



System User

Manual  
Operation

Manual Input  
Interface/Device

Display  
Interface/Device

Printout  
Interface/Device

Information  
System

Data  
Storage

# ANALISIS DAN PERANCANGAN SISTEM

BAHAR

## Daftar Isi

<b>Modul 01</b>	<b>1.1</b>
Konsep Pengembangan Sistem	
<b>Kegiatan Belajar 1</b>	1.4
Konsep Dasar Pengembangan Sistem	
<b>Kegiatan Belajar 2</b>	1.27
Metodologi Pengembangan Sistem	
<b>Modul 02</b>	<b>2.1</b>
Perencanaan Sistem	
<b>Kegiatan Belajar 1</b>	2.4
Mendefinisikan Proyek Sistem	
<b>Kegiatan Belajar 2</b>	2.26
Membuat Rencana Proyek	
<b>Modul 03</b>	<b>3.1</b>
Analisis Sistem	
<b>Kegiatan Belajar 1</b>	3.4
Penentuan Kebutuhan (Persyaratan)	
<b>Kegiatan Belajar 2</b>	3.23
Teknik Elicitasi dan Strategi Analisis Kebutuhan	
<b>Modul 04</b>	<b>4.1</b>
Pemodelan Analisis dengan Pendekatan Terstruktur	
<b>Kegiatan Belajar 1</b>	4.5
Data Flow Diagram	
<b>Kegiatan Belajar 2</b>	4.40
Bagan Alir	

<b>Modul 05</b>	<b>5.1</b>
Pemodelan Analisis dengan Pendekatan Berorientasi Objek	
<b>Kegiatan Belajar 1</b> Paradigma Berorientasi Objek	5.4
<b>Kegiatan Belajar 2</b> Pemodelan Fungsional dengan <i>Use Case Diagram</i>	5.26
<b>Modul 06</b>	<b>6.1</b>
Pemodelan Proses Bisnis dan Struktural Sistem	
<b>Kegiatan Belajar 1</b> Pemodelan Proses Bisnis dengan <i>Activity Diagram</i>	6.5
<b>Kegiatan Belajar 2</b> Pemodelan Struktural dengan <i>Class Diagram dan Object Diagram</i>	6.18
<b>Modul 07</b>	<b>7.1</b>
Pemodelan Perilaku ( <i>Behavioral</i> ) Objek	
<b>Kegiatan Belajar 1</b> Diagram Interaksi ( <i>Sequence Diagram dan Communication Diagram</i> )	7.5
<b>Kegiatan Belajar 2</b> <i>Behavioral State Machines Diagram</i>	7.26
<b>Modul 08</b>	<b>8.1</b>
Transisi dari Analisis Sistem ke Desain/Perancangan Sistem	
<b>Kegiatan Belajar 1</b> Transformasi <i>Entity Relationship Diagram</i> ke Diagram Relasi	8.4

<b>Kegiatan Belajar 2</b>	8.33
Strategi Akuisisi	
Sistem Informasi	
<b>Modul 09</b>	
Desain/Perancangan Sistem	<b>9.1</b>
<b>Kegiatan Belajar 1</b>	9.5
Desain Arsitektur Sistem	
<b>Kegiatan Belajar 2</b>	9.26
Desain Antarmuka Pengguna	

## Tinjauan Mata Kuliah

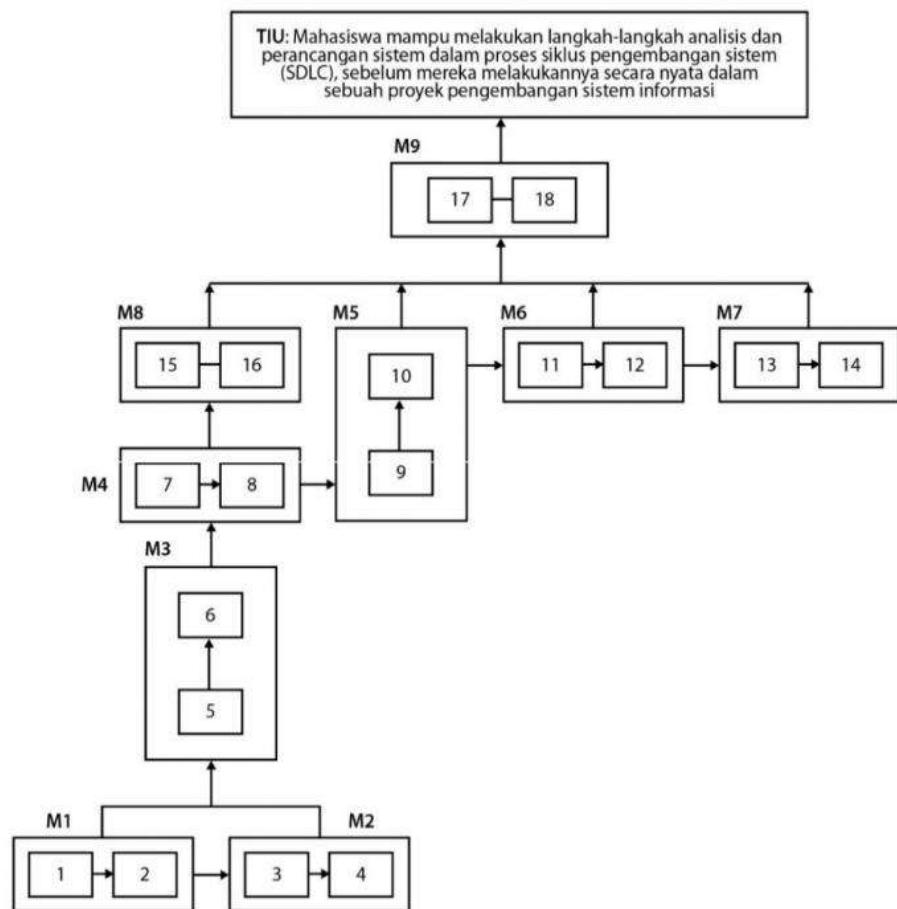
Mata kuliah ini membahas mengenai cara menganalisis dan merancang Sistem Informasi berdasarkan siklus pengembangan sistem (*System Development Life Cycle/SDLC*), dengan menggunakan alat-alat bantu pemodelan sistem. Pembahasan diawali dengan penyajian konsep dasar pengembangan sistem informasi, yang meliputi prinsip, alasan, dan pendekatan pengembangan sistem. Selanjutnya menyajikan berbagai metodologi dalam pengembangan sistem informasi, teknik dan strategi menganalisis kebutuhan, teknik memodelkan kebutuhan dengan menggunakan notasi pemodelan terstruktur maupun pemodelan berorientasi objek. Bagian akhir membahas mengenai mekanisme melakukan translasi hasil analisis ke perancangan sistem yang meliputi: strategi akuisisi sistem informasi, perancangan arsitektur sistem, dan perancangan antarmuka pengguna.

Setelah mempelajari mata kuliah ini mahasiswa mampu:

1. menjelaskan konsep pengembangan sistem informasi;
2. menjelaskan berbagai metode dalam pengembangan sistem informasi;
3. menggunakan berbagai perangkat dan metode untuk menganalisis aliran dan struktur informasi dalam proses organisasi;
4. menggunakan tools pendekatan terstruktur *Data Flow Diagram* (DFD) dan tools pendekatan berorientasi objek *Unified Modeling Language* (UML) untuk membangun model konseptual/logik (model fungsional, model proses, model struktural, dan model perilaku) sistem informasi;
5. melakukan perancangan fisik sistem berupa perancangan arsitektur sistem dan antarmuka pengguna.

Untuk mengambil mata kuliah ini, mahasiswa pernah mengambil atau dapat mengambil bersamaan dengan mata kuliah: Sistem Informasi Manajemen, Basis Data, Interaksi Manusia dan Komputer, Algoritma dan Pemograman, dan Rekayasa Perangkat Lunak.

**PETA KOMPETENSI**  
**MSIM4302/Analisis dan Perancangan Sistem/3 sks**



**Keterangan Peta Kompetensi:**

1. Menjelaskan konsep dasar pengembangan sistem dan peran analis
2. Mengklasifikasikan metode pengembangan sistem
3. Menjelaskan bagaimana mendefinisikan dan menilai proyek sistem
4. Menjelaskan bagaimana membuat rencana proyek
5. Mengklasifikasikan dan menentukan kebutuhan sistem
6. Menggunakan teknik elisitasi untuk menentukan kebutuhan sistem
7. Menggunakan diagram arus data (*Data Flow Diagram/DFD*) untuk melakukan pemodelan analisis
8. Menggunakan bagan alir sistem (*flowchart*) untuk mendeskripsikan proses secara lebih detail dalam pemodelan analisis
9. Menjelaskan paradigma berorientasi objek
10. Menggunakan *use case diagram* pada UML untuk merancang model fungsional sistem
11. Menggunakan *activity diagram* pada UML untuk merancang model proses bisnis

12. Menggunakan *class diagram* dan *object diagram* pada UML untuk merancang model struktural sistem
13. Menggunakan *sequence diagram* dan *communication diagram* pada UML untuk merancang model perilaku sistem
14. Menggunakan *behavioral state machines diagram* pada UML untuk merancang model perilaku sistem
15. Melakukan proses transformasi ERD ke diagram relasi database dalam transisi fase analisis ke fase desain/perancangan
16. Menjelaskan strategi-strategi akuisisi sistem dalam transisi fase analisis ke fase desain/perancangan
17. Menjelaskan model-model desain/rancangan arsitektur sistem
18. Menjelaskan dan membuat desain/rancangan antarmuka pengguna

**Keterangan Modul (M):**

Modul 1 Konsep Pengembangan Sistem

- Kegiatan Belajar 1: Konsep Dasar Pengembangan Sistem
- Kegiatan Belajar 2: Metodologi Pengembangan Sistem

Modul 2 Perencanaan Sistem

- Kegiatan Belajar 1: Mendefinisikan Proyek Sistem
- Kegiatan Belajar 2: Membuat Rencana Proyek

Modul 3 Analisis Sistem

- Kegiatan Belajar 1: Penentuan Kebutuhan (Persyaratan)
- Kegiatan Belajar 2: Teknik Elisitasi dan Strategi Analisis Kebutuhan

Modul 4 Pemodelan Analisis dengan Pendekatan Terstruktur

- Kegiatan Belajar 1: *Data Flow Diagram* (DFD)
- Kegiatan Belajar 2: Bagan Alir (*Flowchart*)

Modul 5 Pemodelan Analisis dengan Pendekatan Berorientasi Objek

- Kegiatan Belajar 1: Paradigma Berorientasi Objek
- Kegiatan Belajar 2: Pemodelan Fungsional dengan *Use Case Diagram*

Modul 6 Pemodelan Proses Bisnis dan Struktural Sistem

- Kegiatan Belajar 1: Pemodelan Proses Bisnis dengan *Activity Diagram*
- Kegiatan Belajar 2: Pemodelan Struktural dengan *Class Diagram* dan *Object Diagram*

Modul 7 Pemodelan Perilaku (*Behavioral*) Objek

- Kegiatan Belajar 1: Diagram Interaksi (*Sequence Diagram* dan *Communication Diagram*)
- Kegiatan Belajar 2: *Behavioral State Machines Diagram*

Modul 8 Transisi dari Analisis Sistem ke Desain/Perancangan Sistem

- Kegiatan Belajar 1: Transformasi *Entity Relationship Diagram* ke Diagram Relasi
- Kegiatan Belajar 2: Strategi Akuisisi Sistem Informasi

Modul 9 Desain/Perancangan Sistem

- Kegiatan Belajar 1: Desain Arsitektur Sistem
- Kegiatan Belajar 2: Desain Antarmuka Pengguna

**MSIM4302**  
**Edisi 1**

**MODUL 01**

# **Konsep Pengembangan Sistem**

Bahar, S.T., M.Kom.

## Daftar Isi

<b>Modul 01</b>	
Konsep Pengembangan Sistem	
<b>Kegiatan Belajar 1</b>	1.4
Konsep Dasar	
Pengembangan Sistem	
Latihan	1.21
Rangkuman	1.23
Tes Formatif 1	1.24
<b>Kegiatan Belajar 2</b>	1.27
Metodologi Pengembangan Sistem	
Latihan	1.47
Rangkuman	1.49
Tes Formatif 2	1.50
<b>Kunci Jawaban Tes Formatif</b>	1.53
Glosarium	1.54
Daftar Pustaka	1.56



## Pendahuluan

**S**ebagai upaya untuk memenangkan persaingan atau mempertahankan keberlangsungan organisasi, sistem informasi perlu dikembangkan atau disempurnakan mengikuti perkembangan teknologi yang terus melaju dengan cepat. Namun demikian keputusan untuk mengembangkan sistem tidak serta merta memberikan jaminan keberhasilan, sebaliknya dapat menemui kegagalan sehingga perlu dipertimbangkan dengan matang sebelum proses pengembangan sistem dilakukan. Dalam mengembangkan sistem informasi, manajemen organisasi perlu memperhatikan berbagai faktor, seperti kebutuhan yang harus dipenuhi, tujuan yang harus dicapai, keberadaan sumber daya dalam organisasi, serta metodologi yang tersedia.

Setelah mempelajari modul ini mahasiswa diharapkan secara lebih rinci mampu menjelaskan:

1. pengertian pengembangan sistem;
2. alasan-alasan yang menyebabkan perlunya pengembangan sistem;
3. prinsip-prinsip dalam pengembangan sistem dan faktor-faktor yang menyebabkan kegagalan mencapai tujuan dalam pengembangan sistem;
4. tim pengembang sistem dan peran masing-masing anggota tim dalam pengembangan sistem;
5. siklus hidup pengembangan sistem;
6. ragam metodologi pengembangan sistem;
7. pentingnya penggunaan metodologi dalam pengembangan sistem;
8. pengaruh penggunaan metodologi dalam pengembangan sistem.

## Konsep Dasar Pengembangan Sistem

Sistem informasi terkomputerisasi telah dikembangkan dalam berbagai bentuk untuk beberapa dekade terakhir. Pengembangan sistem mencakup aplikasi sederhana untuk mendukung suatu proses atau fungsi, hingga pada model aplikasi ERP (*Enterprise Resource Planning*) yang luas untuk mendukung berbagai aspek pengelolaan manajemen dalam organisasi.

Pada masa-masa awal pengembangan sistem informasi, pemrogram (*programmer*) memegang peran penting dalam menangani tugas pengembangan sistem ini, karena pengembangan sistem dianggap sebagai masalah teknis. Pada awalnya sistem tidak terdokumentasi dengan baik dan pemrogram yang membuatnya merupakan satu-satunya orang yang memiliki pengetahuan tentang bagaimana sistem tersebut dibangun dan bagaimana sistem bekerja. Situasi ini menjadi tidak bisa dipertahankan ketika sistem yang dibuat akan dipelihara, dikembangkan lebih lanjut, dan dikombinasikan dengan sistem yang lain. Kebutuhan akan konsep kerja yang lebih terstruktur menjadi semakin diperlukan.

Kegiatan belajar ini akan membahas mengenai pengertian pengembangan sistem, perlunya pengembangan sistem, prinsip pengembangan sistem, dan tim pengembangan sistem.

### A. PENGERTIAN PENGEMBANGAN SISTEM

Pengembangan Sistem Informasi sering hanya disebut dengan istilah Pengembangan Sistem (*system development*). Terdapat beberapa definisi mengenai pengembangan sistem, di antaranya adalah:

1. aktivitas untuk menghasilkan sistem informasi berbasis komputer untuk menyelesaikan permasalahan (*problem*) organisasi atau memanfaatkan kesempatan (*opportunities*) yang timbul;
2. kumpulan kegiatan para analis sistem, perancang, dan pengguna yang mengembangkan dan mengimplementasikan sistem informasi;
3. tahapan kegiatan yang dilakukan selama masa pembangunan sistem informasi;
4. proses merencanakan, mengembangkan, dan mengimplementasikan sistem informasi dengan menggunakan metode, teknik, dan alat bantu pengembangan tertentu.

Pengembangan sistem (*system development*) dapat diartikan sebagai suatu kegiatan menyusun suatu sistem yang baru untuk menggantikan sistem yang lama secara keseluruhan atau memperbaiki sistem yang telah ada (Jogiyanto, 2008). Sistem lama dapat berupa sistem-sistem informasi yang masih manual, berupa sistem pencatatan transaksi pada lembaran-lembaran arsip. Sistem lama juga dapat berarti sistem-sistem informasi yang telah menggunakan teknologi komputer (aplikasi sistem database terkomputerisasi), baik yang berupa sistem berbasis *desktop* maupun sistem basis data terdistribusi.

## B. PERLUNYA PENGEMBANGAN SISTEM

Sebuah sistem perlu dikembangkan (diperbaiki atau diganti) karena beberapa hal:

1. **Terdapat permasalahan yang timbul pada sistem yang lama. Permasalahan yang timbul dapat berupa:**

- a. *Ketidakberesan*

Ketidakberesan dalam sistem yang lama menyebabkan sistem yang lama tidak dapat beroperasi sesuai dengan yang diharapkan. Ketidakberesan ini dapat berupa:

- 1) kecurangan-kecurangan disengaja yang menyebabkan kebenaran dari data menjadi kurang terjamin;
  - 2) kesalahan-kesalahan yang tidak disengaja yang juga dapat menyebabkan kebenaran dari data kurang terjamin;
  - 3) tidak efisiennya operasi sistem.

*Contoh Kasus 1.1:*

E-KUANGAN: Menuju *Good Governance* (Hardiansyah, 2019)

Kasus korupsi dalam pemerintahan pusat dan daerah menimbulkan kerugian negara yang berujung pada kesejahteraan masyarakat. Berbagai modus yang digunakan oleh para koruptor untuk menjalankan aksinya seperti: *markup* laporan keuangan, laporan fiktif, penyalahgunaan wewenang, anggaran ganda, dan lain-lain. Masyarakat tidak tahu detil anggaran pelaksanaan dan laporan pertanggungjawaban program-program pembangunan yang ada. Dalam kasus seperti ini, porsi pengawasan masyarakat dalam kontrol anggaran masih sangat kurang. Diperlukan keterbukaan dan pengawasan publik melalui aksesibilitas informasi, sehingga masyarakat dapat melakukan pengawasan.

e-Keuangan adalah sistem penyusunan anggaran yang di dalamnya termasuk aplikasi program komputer berbasis web untuk memfasilitasi proses penyusunan anggaran belanja daerah. Dalam aplikasi tersebut terdapat beberapa fitur untuk mendukung keberhasilan di antaranya adalah: *e-planing* dan *e-controlling*. Sistem ini dibuat secara *online* agar dapat diakses oleh instansi di mana pun lokasinya, dapat diakses pada saat pembahasan dengan Dewan Perwakilan Rakyat Daerah (DPRD) dan

dapat diakses oleh masyarakat jika ingin mengetahui kinerja pemerintah dan sirkulasi keuangan daerah. e-Keuangan juga digunakan untuk meningkatkan efisiensi dan efektivitas dalam proses penganggaran. Sistem e-Keuangan memberikan peranan yang penting dalam pembuatan anggaran. Sistem tersebut mampu mempersingkat waktu yang diperlukan dalam proses pembuatan anggaran karena dilakukan secara *online* dan dapat diakses kapan saja dan di mana saja.

*Contoh Kasus 1.2:*

Pengembangan Sistem Informasi Geografis dalam Manajemen Tata Ruang Wilayah (Radliya, 2018)

Berdasarkan Undang-Undang Republik Indonesia Nomor 26 Tahun 2007, tata ruang merupakan susunan pusat-pusat permukiman, sistem jaringan sarana dan prasarana yang berfungsi untuk mendukung kegiatan sosial ekonomi masyarakat yang secara hierarki memiliki hubungan fungsional. Manajemen tata ruang di suatu wilayah telah diatur dalam bentuk Peraturan Pemerintah. Ketidaaan fasilitas yang dapat digunakan sebagai media penyebaran informasi mengenai Rencana Tata Ruang Wilayah (RT/RW) yang mudah diakses oleh masyarakat (masyarakat perorangan atau kelompok masyarakat, seperti masyarakat hukum adat dan korporasi) yang akan memanfaatkan ruang, menyebabkan kesalahan dalam peruntukan wilayah/tata ruang. Kesalahan pemanfaatan ruang misalnya, pembabatan hutan untuk pembangunan perumahan, dapat mengakibatkan berkurangnya lahan hijau untuk daerah resapan air, sehingga dapat menimbulkan banjir.

Pemanfaatan Teknologi Informasi dan Komunikasi yang dapat dilakukan adalah dengan membangun Sistem Informasi Geografis. Sistem Informasi Geografis dapat digunakan untuk penyediaan informasi mengenai sarana dan prasarana yang ada di suatu wilayah dalam bentuk data spasial (peta). Informasi peta yang disediakan adalah peta administratif (mencakup kecamatan, pusat pemerintahan, sungai, jalan, desa), peta rancang pola (mencakup jalur kereta api, area pariwisata, energi induk, tambang), peta pengguna lahan, peta rencana sistem jaringan sarana dan prasarana (mencakup peta rencana prasarana transportasi darat, peta rencana prasarana perkeretaapian, peta rencana prasarana lingkungan permukiman, peta rencana jaringan energi, peta rencana prasarana sumberdaya air, peta rencana prasarana telekomunikasi, serta peta rencana prasarana transportasi udara). Penerapan Sistem Informasi Geografis tata ruang wilayah, memungkinkan terjadinya partisipasi seluruh lapisan masyarakat dalam pengambilan keputusan penggunaan suatu area sesuai peruntukannya.

*b. Pertumbuhan dan Perubahan Organisasi*

Pertumbuhan dan perubahan organisasi adalah suatu proses membesar atau meluasnya sebuah organisasi ke arah yang lebih baik. Pertumbuhan dan perubahan organisasi dipengaruhi oleh faktor internal dan faktor eksternal. Faktor internal dapat berupa perubahan kebijakan organisasi, perubahan struktur organisasi, dan volume

kegiatan yang bertambah banyak. Faktor eksternal dapat berupa sumber daya alam, kompetisi yang semakin tajam antar organisasi, dan perubahan lingkungan, baik lingkungan fisik maupun lingkungan sosial.

*Contoh Kasus 1.3:*

PT. XYZ adalah perusahaan yang bergerak dalam bidang penyediaan jasa konstruksi yang berhubungan dengan pembangunan, seperti pembangunan rumah, pembangunan gedung, dan pembangunan pabrik. Setiap proyek yang ditangani mempunyai data perencanaan pekerjaan. Pada periode waktu tertentu, data perencanaan pekerjaan dibandingkan dengan data pekerjaan yang telah diselesaikan, sehingga dapat diketahui perkembangan pekerjaan proyek.

Perkembangan pekerjaan proyek secara periodik dilaporkan ke pimpinan proyek sebagai bahan pengambilan keputusan bagi jajaran direksi perusahaan. Mekanisme pelaporan dilakukan dengan cara: setiap mandor mencatat data perkembangan proyek, kemudian *supervisor* memeriksa kembali data yang ada dan menentukan *volume* perkembangan proyek. Selanjutnya setiap *supervisor* menghubungi administrator sebagai pusat pengolahan data melalui telepon untuk menyampaikan data perkembangan proyek pada bidang yang ditanganinya. Data tersebut dientri ke dalam aplikasi sistem informasi berbasis desktop, dan menghasilkan informasi bagi manager proyek.

Seiring dengan semakin membaiknya citra perusahaan, lokasi pembangunan proyek menjadi semakin luas, tersebar pada daerah-daerah yang berjauhan. Sistem pelaporan dan pengolahan data untuk menghasilkan informasi perkembangan pekerjaan proyek tidak lagi dapat dilakukan di satu tempat, melainkan harus dilakukan di setiap lokasi proyek. Permasalahan muncul ketika sistem pengolahan data menggunakan aplikasi berbasis desktop, yaitu integrasi data yang sulit dilakukan untuk keperluan pemantauan perkembangan pekerjaan proyek secara keseluruhan dan terintegrasi.

Untuk mengatasi permasalahan tersebut, perusahaan melakukan migrasi sistem dari *platform* database berbasis *desktop* menjadi sistem database terdistribusi melalui sistem informasi berbasis web. Dengan sistem informasi berbasis web, pihak perusahaan dapat mengintegrasikan sistem komunikasi dan pengolahan data untuk memantau perkembangan pekerjaan proyek secara terintegrasi.

## 2. Untuk Meraih Kesempatan

Teknologi informasi berupa perangkat keras komputer, perangkat lunak, dan teknologi komunikasi telah begitu cepat berkembang. Organisasi mulai merasakan bahwa teknologi informasi ini perlu digunakan untuk meningkatkan penyediaan informasi sehingga dapat mendukung dalam proses pengambilan keputusan yang akan dilakukan oleh manajemen. Dalam keadaan pasar bersaing, kecepatan informasi atau efisiensi waktu sangat menentukan berhasil atau tidaknya strategi dan rencana-rencana yang telah disusun untuk meraih kesempatan dan keunggulan kompetitif. Bila pesaing dapat memanfaatkan teknologi ini, maka kesempatan-kesempatan akan jatuh ke tangan

pesaing. Kesempatan-kesempatan ini dapat berupa peluang-peluang pasar, pelayanan yang meningkat kepada pelanggan, dan lain sebagainya.

Sistem Informasi Strategis adalah sistem informasi yang menggunakan teknologi informasi untuk membantu perusahaan dalam hal mendapatkan keunggulan bersaing, meminimalkan hal yang tidak menguntungkan, sehingga tercapai tujuan strategis perusahaan. Sistem Informasi Strategis membantu perusahaan dengan menyediakan produk dan layanan yang memberikan keuntungan lebih strategis dibandingkan pesaingnya dalam pasar yang kompetitif. Sistem Informasi Strategis dapat juga diasumsikan sebagai sistem informasi yang mempromosikan inovasi bisnis, meningkatkan proses bisnis, dan membangun sumber daya informasi bagi sebuah perusahaan.

*Contoh Kasus 1.4:*

STIMI ANTA BERANTA adalah sebuah Perguruan Tinggi bidang Ilmu Manajemen di Kota Banjar Lama. Dengan menggunakan sistem marketing konvensional, informasi mengenai profil lembaga disampaikan melalui brosur dan *leaflet* kepada calon mahasiswa di sekolah-sekolah menengah atas di wilayah sekitar Banjar Lama. Konsep marketing konvensional yang diterapkan saat ini mampu menjaring mahasiswa baru dari tahun ke tahun sesuai kuota yang ditargetkan. Untuk mendukung pengelolaan administrasi akademik, institusi telah mengembangkan aplikasi Sistem Informasi Akademik berbasis jaringan lokal. Mahasiswa dapat memanfaatkan sistem informasi ini untuk berpartisipasi langsung dalam perencanaan dan registrasi mata kuliah setiap semester melalui koneksi jaringan lokal berbasis Wifi di lingkungan kampus. Dengan konsep seperti ini, manajemen akademik dapat mempersiapkan kegiatan perkuliahan secara efektif, sehingga awal perkuliahan dapat dilaksanakan tepat pada waktunya.

Walaupun target penerimaan mahasiswa baru dari tahun ke tahun selalu terpenuhi dan layanan akademik dapat berjalan baik dengan menggunakan Sistem Informasi Akademik berbasis jaringan lokal, institusi bermaksud mengembangkan sistem yang ada menjadi Sistem Informasi Akademik berbasis web, yang selain dapat difungsikan sebagai Sistem Layanan Akademik berbasis *online*, juga dapat difungsikan sebagai Sistem Informasi Marketing dan Penerimaan Mahasiswa Baru. Sikap manajemen institusi mengembangkan/memperluas sistem informasi ini bertujuan untuk meraih peluang pasar yang lebih besar, dengan cara meningkatkan citra institusi melalui layanan marketing, penerimaan mahasiswa baru, dan layanan administrasi akademik secara online. Semakin membaiknya citra institusi diharapkan memberikan dampak positif berupa peningkatan kepercayaan masyarakat, khususnya calon mahasiswa baru kepada institusi, sehingga institusi dapat meningkatkan jumlah kuota penerimaan mahasiswa di tahun-tahun mendatang.

### 3. Adanya Instruksi

Pengembangan sistem yang baru dapat juga terjadi karena adanya instruksi dari luar organisasi, misalnya peraturan pemerintah mengenai perubahan sistem perpajakan, kebijakan pemerintah yang mewajibkan semua perguruan tinggi untuk mengembangkan

sistem informasi secara internal sebagai *repository* untuk menampung setiap hasil penelitian mahasiswa, dan lain-lain.

Berikut ini dapat digunakan sebagai indikator adanya permasalahan-permasalahan dan kesempatan-kesempatan yang dapat diraih, sehingga menyebabkan sistem yang lama harus diperbaiki, ditingkatkan, bahkan diganti keseluruhannya. Indikator-indikator ini di antaranya adalah sebagai berikut:

- a. infomasi sulit diakses;
- b. informasi tidak relevan dengan kebutuhan;
- c. informasi yang dihasilkan tidak *up-to-date*;
- d. pelaporan tidak tepat waktu;
- e. isi informasi/laporan yang sering salah;
- f. produktivitas tenaga kerja rendah;
- g. terlalu banyak informasi sehingga membingungkan;
- h. proses pengolahan dan penyajian informasi lamban;
- i. penyusunan dan penyajian informasi tidak sistematis;
- j. banyak informasi ilegal;
- k. jangkauan distribusi informasi pendek;
- l. tidak dapat dilakukan pengolahan informasi secara tersebar;
- m. tanggung jawab antar bagian yang tidak jelas;
- n. waktu kerja yang berlebihan;
- o. kegiatan yang tumpang tindih;
- p. tanggapan yang lambat terhadap langganan;
- q. kehilangan kesempatan kompetisi pasar;
- r. kesalahan-kesalahan proses manual yang tinggi;
- s. biaya operasi yang tinggi;
- t. file-file yang kurang teratur;
- u. peramalan penjualan dan produksi tidak tepat;
- v. kapasitas produksi yang menganggur (*idle capacities*);
- w. pekerjaan manajer yang terlalu teknis.

Proses pengembangan sistem dapat diilustrasikan seperti pada Gambar 1.1. Pada Gambar 1.1 terlihat bahwa sistem dikembangkan hanya jika terdapat permasalahan atau peluang atau instruksi. Jika tidak, sistem tidak perlu dikembangkan. Sistem juga dikembangkan hanya jika terdapat tujuan yang jelas, yaitu untuk memecahkan masalah atau meraih peluang menuju ke arah yang lebih menguntungkan, atau untuk memenuhi kebijakan.

Dengan dikembangkannya sistem yang baru, maka diharapkan akan terjadi peningkatan-peningkatan yang berhubungan dengan:

- a. kinerja organisasi;
- b. kualitas penyajian informasi;

## 1.10 Konsep Pengembangan Sistem

- c. manfaat-manfaat atau keuntungan-keuntungan atau penurunan-penurunan biaya yang terjadi (ekonomis);
- d. pengendalian untuk mendeteksi dan memperbaiki kesalahan-kesalahan serta kecurangan-kecurangan yang akan terjadi;
- e. efisiensi operasi, yaitu bagaimana sumber daya tersebut digunakan dengan pemberoran yang paling minimum;
- f. pelayanan yang diberikan oleh sistem.



Gambar 1.1  
Proses Pengembangan Sistem

### C. PRINSIP PENGEMBANGAN SISTEM

Dalam proses pengembangan sistem terdapat beberapa hal yang berpotensi menyebabkan tidak dapat tercapainya tujuan utama pengembangan (Jogiyanto, 2008), yaitu sebagai berikut.

1. Tidak Mendapatkan Dukungan Penuh dari Pihak Manajemen.  
Banyak organisasi yang ketika mengembangkan sistem tidak mempunyai komite pengarah sistem informasi. Kalaupun ada, komite pengarah bersifat tidak formal, sehingga komite pengarah tidak dapat bekerja secara efisien dan efektif sehingga

- jika ada rencana strategis sistem informasi, sering kali tidak berhubungan dengan rencana strategis jangka panjang organisasi.
2. Terjadi Perubahan Kebutuhan Pemakai.  
Pada saat manajemen tingkat atas berada pada situasi yang semakin kompleks, para manajer senior menuntut sistem informasi untuk memberikan keuntungan kompetitif strategis. Semakin strategis nilai sebuah sistem informasi, semakin besar resiko kegagalan mencapai tujuan-tujuan sistem informasi tersebut.
  3. Kemunculan Teknologi Baru.  
Ketika organisasi menggunakan teknologi yang ada, organisasi dapat lebih mudah mencapai tujuan-tujuan pengembangan sistem, karena personel-personel telah menguasai teknologi tersebut. Namun ketika organisasi berusaha membuat keuntungan kompetitif dengan menggunakan teknologi yang baru atau yang lebih canggih, organisasi tidak dapat dengan mudah mencapai tujuan-tujuan pengembangan sistem, karena personel-personel kemungkinan tidak menguasai penggunaan teknologi baru tersebut.
  4. Standar Metodologi Pengembangan Sistem yang Tidak *Up-to-Date*.  
Beberapa organisasi tidak mempunyai standar metodologi pengembangan sistem. Jika ada, seringkali organisasi tidak menjaga keterkinian manual standar metodologi pengembangan sistem yang dimiliki.
  5. Kelebihan Beban Kerja atau Kurangnya Keahlian Personel dalam Pengembangan Sistem.  
Sebuah survei memperkirakan bahwa personel-personel dalam tim pengembangan sistem menghadapi keterlambatan dalam pengembangan sistem mulai dari enam bulan sampai dengan lima tahun. Selain kelebihan beban kerja, personel-personel dalam tim pengembangan sistem sering tidak memiliki keahlian, karena organisasi tersebut tidak mempunyai rencana training.

Oleh karena itu beberapa prinsip berikut ini harus diperhatikan sewaktu akan mengembangkan suatu sistem informasi.

1. Sistem dikembangkan untuk mendukung manajemen.  
Setelah sistem selesai dikembangkan, maka yang akan menggunakan sistem ini adalah manajemen, sehingga sistem harus dapat mendukung kebutuhan yang diperlukan oleh manajemen. Prinsip ini selalu harus diperhatikan pada waktu mengembangkan sistem.
2. Sistem yang dikembangkan adalah investasi modal yang besar.  
Sistem informasi yang dikembangkan membutuhkan dukungan pendanaan yang besar, apalagi jika menggunakan teknologi yang mutakhir. Pada saat menginvestasikan modal harus mempertimbangkan dua hal.
  - a. Investigasi semua alternatif  
Bila lalai memperhitungkan suatu alternatif potensial dan sudah terlanjur menanamkan dana ke suatu proyek investasi tertentu, maka investor akan

kehilangan kesempatan untuk menanamkan dananya ke investasi yang lain. Oleh karena itu beberapa alternatif investasi yang ada harus diinvestigasi untuk menentukan alternatif yang terbaik.

- b. Investasi yang terbaik harus bernilai  
Belum tentu alternatif terbaik merupakan investasi yang menguntungkan. Investasi terbaik memang sering menguntungkan dibandingkan dengan alternatif yang lainnya, namun demikian investasi terbaik harus juga diukur. Investasi dapat dikatakan menguntungkan bila bernilai, artinya manfaat atau hasil baliknya lebih besar dari biaya untuk memperolehnya.
3. Sistem yang dikembangkan memerlukan orang yang terdidik.  
Faktor manusia sangat menentukan berhasil tidaknya suatu sistem, baik dalam proses pengembangannya, penerapannya, maupun dalam proses operasinya. Oleh karena itu, orang yang terlibat dalam pengembangan maupun penggunaan sistem ini harus merupakan orang yang terdidik (pendidikan formal atau pelatihan kerja) dan mengerti tentang permasalahan-permasalahan yang ada dan solusi-solusi yang mungkin dilakukan. Analis harus mempunyai pendidikan sesuai dengan masalah yang dihadapinya. Tidaklah mungkin seorang analis akan mengembangkan suatu sistem informasi bisnis tanpa mempunyai pengetahuan sedikit pun tentang bisnis atau akan mengembangkan sistem informasi akuntansi tanpa memiliki pengetahuan sedikit pun tentang akuntansi dan teknologi komputer. Bagaimana mungkin nantinya analis ini akan berkomunikasi dengan pihak manajemen dan pemrogram yang akan membuat programnya. Demikian juga dengan pengguna sistem harus merupakan orang yang terdidik tentang sistem ini, dapat dilakukan dengan memberikan pelatihan kepada mereka tentang cara menggunakan sistem yang diterapkan.
4. Terdapat tahapan kerja dan tugas-tugas yang jelas yang harus dilakukan dalam proses pengembangan sistem.  
Proses pengembangan sistem pada umumnya terdiri dari beberapa tahapan kerja dan melibatkan beberapa personel dalam bentuk suatu tim kerja. Pengalaman menunjukkan bahwa tanpa adanya perencanaan dan koordinasi yang baik, proses pengembangan sistem tidak akan berhasil dengan memuaskan. Untuk maksud ini sebelum proses pengembangan sistem dilakukan, terlebih dahulu dibuat jadwal kerja yang menunjukkan tahapan-tahapan kerja dan tugas-tugas pekerjaan yang akan dilakukan, sehingga proses pengembangan sistem dapat dilakukan dan selesai, dan berhasil sesuai dengan waktu dan anggaran yang direncanakan. Siklus hidup pengembangan sistem umumnya menunjukkan tahapan-tahapan kerja dan tugas-tugas kerja yang harus dilakukan. Beberapa metodologi pengembangan

sistem juga menyediakan lebih terinci konsep kerja yang harus dilakukan dalam proses pengembangan sistem.

5. Proses pengembangan sistem tidak harus dilakukan secara berurutan.

Prinsip ini kelihatannya bertentangan dengan prinsip nomor 4, tetapi tidaklah demikian. Tahapan kerja dari pengembangan sistem di prinsip nomor 4 menunjukkan langkah-langkah yang harus dilakukan secara bersama-sama. Misalnya di dalam pengembangan sistem, perancangan output merupakan tahapan yang harus dilakukan sebelum melakukan perancangan file. Ini tidak berarti bahwa semua output harus dirancang semuanya terlebih dahulu baru dapat dilakukan perancangan file, tetapi dapat dilakukan secara serentak, yaitu pada saat proses perancangan output masih dilakukan, hasil perancangan output yang telah selesai dapat digunakan untuk merancang file. Contoh lain, pada saat proses pengadaan *hardware* dilakukan, perancangan lainnya yang tidak tergantung pada keberadaan *hardware* dapat dilakukan secara paralel.

6. Jangan takut membatalkan proyek.

Umumnya membatalkan suatu proyek yang sedang berjalan merupakan pantangan. Keputusan untuk meneruskan suatu proyek atau membatalkannya memang harus dievaluasi dengan cermat. Untuk kasus-kasus tertentu, di mana suatu proyek terpaksa harus dihentikan atau dibatalkan karena sudah tidak layak lagi, maka harus dilakukan dengan tegas. Keraguan untuk terus melanjutkan proyek yang tidak layak lagi karena sudah terserapnya dana ke dalam proyek ini hanya akan membuang dana yang sia-sia. Jika proyek yang tidak layak masih terus dilanjutkan, maka dana berikutnya yang terserap akan sia-sia juga.

7. Dokumentasi harus ada sebagai pedoman dalam pengembangan sistem.

Kegagalan membuat suatu dokumentasi kerja merupakan salah satu hal yang sering terjadi dan merupakan kesalahan kritis yang dibuat oleh analis. Tim analis sering membuat dokumentasi hasil analisis setelah mereka selesai mengembangkan sistemnya dan bahkan ada yang tidak membuat dokumentasi. Dokumentasi seharusnya dibuat pada saat proses pengembangan sistem sedang berlangsung, karena dokumentasi ini dapat dihasilkan dari hasil kerja tiap-tiap langkah selama pengembangan sistem. Dokumentasi yang dibuat selama proses pengembangan sistem dapat digunakan sebagai alat komunikasi antara analis dengan pengguna sistem dan dapat digunakan untuk mendorong keterlibatan pengguna sistem.

#### D. TIM PENGEMBANGAN SISTEM

Dalam proyek pengembangan sistem yang kecil dan sederhana, kemungkinan hanya ada seorang analis sistem yang merangkap sebagai pemrogram (analisis/pemrogram) atau seorang pemrogram yang merangkap sebagai analis sistem (pemrogram/analisis). Akan tetapi untuk proyek pengembangan sistem yang besar atau kompleks, pekerjaan ini biasanya dilakukan oleh sejumlah orang dalam bentuk tim. Anggota dari tim pengembangan sistem ini tergantung dari besar-kecilnya ruang lingkup proyek yang akan ditangani. Tim ini secara umum dapat terdiri dari beberapa personel yang akan dibahas selanjutnya.

Dari berbagai fase dan langkah yang dilakukan dalam siklus pengembangan sistem, tim proyek membutuhkan beragam keterampilan. Anggota tim adalah agen perubahan yang mengidentifikasi cara untuk meningkatkan organisasi, membangun sistem informasi untuk mendukung mereka, dan melatih serta memotivasi pengguna sistem. Memimpin upaya perubahan organisasi menuju sukses dalam pemanfaatan sistem informasi adalah salah satu pekerjaan paling sulit yang dapat dilakukan seseorang. Memahami apa yang harus diubah dan bagaimana mengubahnya, meyakinkan orang lain tentang perlunya perubahan membutuhkan berbagai keterampilan. Keterampilan ini dapat berupa: keterampilan teknis, bisnis, analitis, interpersonal, manajemen, dan etika.

Analisis harus memiliki keterampilan teknis untuk memahami lingkungan teknis organisasi yang ada, teknologi yang akan membentuk sistem baru, dan kolaborasi keduanya sebagai solusi teknis yang terintegrasi. Keterampilan bisnis diperlukan untuk memahami bagaimana teknologi informasi dapat diterapkan pada situasi bisnis dan untuk memastikan bahwa teknologi informasi memberikan nilai bisnis yang nyata. Analis adalah pemecah masalah yang berkesinambungan, baik di tingkat proyek maupun di tingkat organisasi.

Analisis sering perlu berkomunikasi satu per satu dengan pengguna dan manajer bisnis (yang sering memiliki hanya sedikit pengalaman terkait teknologi) dan dengan pemrogram (yang sering memiliki lebih banyak keahlian teknis daripada keahlian analisis). Mereka harus mampu melakukan presentasi kepada kelompok-kelompok besar maupun kelompok-kelompok kecil dan menulis laporan. Mereka tidak hanya perlu memiliki kemampuan interpersonal yang kuat, akan tetapi mereka juga perlu mengelola orang-orang yang bekerja dengan mereka, serta mengelola risiko yang terkait dengan situasi yang tidak jelas.

Singkat kata, analis harus berurusan dengan anggota tim proyek lainnya, manajer, dan pengguna sistem secara adil, jujur, dan etis. Analis sering berurusan dengan informasi rahasia atau informasi yang jika dibagikan kepada orang lain, dapat menyebabkan kerugian. Penting untuk menjaga kepercayaan diri dan kepercayaan dengan semua orang.

Selain beberapa keterampilan umum yang telah dipaparkan sebelumnya, analis memerlukan banyak keterampilan khusus yang terkait dengan peran yang dilakukan pada suatu proyek. Pada masa-masa periode awal pengembangan sistem, sebagian besar organisasi mengharapkan satu orang untuk memiliki semua keterampilan khusus yang diperlukan untuk melakukan proyek pengembangan sistem, tetapi karena organisasi dan teknologi saat ini telah menjadi lebih kompleks, kebanyakan organisasi besar sekarang membangun tim proyek yang berisi beberapa individu dengan tanggung jawab yang jelas. Organisasi yang berbeda membagi peran secara berbeda. Sebagian besar tim pengembang sistem informasi mencakup banyak individu, seperti pemrogram yang benar-benar menulis program untuk membentuk sistem, dan penulis laporan teknis yang menyiapkan bantuan dan dokumentasi lainnya (misalnya buku pedoman pengguna dan manual sistem).

Jogiyanto (2008) membagi tim dalam pengembangan sistem informasi menjadi personel-personel yang terdiri atas berikut ini.

1. Manajer Analis Sistem

Manajer analisis sistem (*systems analysis manager*) ini disebut juga sebagai koordinator proyek, dan mempunyai tugas dan tanggung jawab sebagai berikut:

- a. sebagai ketua/koordinator tim pengembangan sistem;
- b. mengarahkan, mengontrol dan mengatur anggota tim pengembangan sistem lainnya;
- c. membuat jadwal pelaksanaan proyek pengembangan sistem yang akan dilakukan;
- d. bertanggungjawab dalam mendefinisikan masalah, studi kelayakan, desain sistem dan penerapannya;
- e. memberikan rekomendasi-rekomendasi perbaikan sistem;
- f. mewakili tim untuk berhubungan dengan pengguna sistem dalam hal perundingan-perundingan dan pemberian-pemberian nasehat kepada manajemen dan pengguna sistem;
- g. membuat laporan-laporan kemajuan proyek (*progress report*);
- h. mengkaji ulang dan memeriksa kembali hasil kerja dari tim.

2. Ketua Analis Sistem

Ketua analis sistem (*lead systems analyst*) biasanya menjabat sebagai wakil dari manajer analisis sistem. Tugasnya adalah membantu tugas dari manajer analisis sistem dan mewakilinya bila manajer analisis sistem berhalangan.

3. Analis Sistem Senior

Analisis sistem senior (*senior systems analyst*) merupakan analis sistem yang sudah berpengalaman.

4. Analis Sistem

Analis sistem (*systems analyst*) merupakan analis sistem yang cukup berpengalaman dan dapat bekerja sendiri tanpa bimbingan dari analis sistem senior.

5. Analis Sistem Yunior

Analis sistem yunior (*junior systems analyst*) merupakan analis sistem yang belum berpengalaman dan masih membutuhkan bimbingan-bimbingan dari analis sistem yang lebih senior. Analis sistem yunior ini sering juga disebut dengan analis sistem yang masih dilatih (*systems analyst trainee*).

6. Pemrogram Aplikasi Senior

Pemrogram aplikasi senior (*senior applications programmer*) merupakan pemrogram komputer yang sudah berpengalaman dengan tugas merancang spesifikasi dari program aplikasi dan mengkoordinasi kerja dari pemrogram yang lainnya. Pemrogram aplikasi senior ini kadang-kadang juga disebut dengan pemrogram/analisis.

7. Pemrogram Aplikasi

Pemrogram aplikasi (*applications programmer*) merupakan pemrogram komputer yang cukup berpengalaman dan dapat melakukan tugasnya tanpa harus dibimbing secara langsung.

8. Pemrogram Aplikasi Yunior

Pemrogram aplikasi yunior (*junior applications programmer*) merupakan pemrogram komputer yang belum berpengalaman dan masih di bawah bimbingan langsung dari pemrogram yang lebih senior. Pemrogram aplikasi yunior biasanya hanya dilibatkan dalam pembuatan modul-modul program yang sederhana, seperti pembuatan bentuk-bentuk I/O. Pemrogram aplikasi yunior ini sering juga disebut dengan pemrogram aplikasi yang masih dilatih (*applications programmer trainee*).

Jogiyanto (2008) juga mengemukakan bahwa seorang analis harus mempunyai pengetahuan yang luas dan keahlian yang khusus, berikut ini.

1. Pengetahuan dan keahlian tentang teknik pengolahan data, teknologi komputer, dan pemrograman komputer.

Keahlian teknis yang harus dimiliki termasuk keahlian dalam penggunaan alat dan teknik untuk pengembangan perangkat lunak aplikasi, serta keahlian dalam menggunakan komputer. Pengetahuan teknis lainnya yang harus dimiliki meliputi pengetahuan tentang perangkat keras komputer, teknologi komunikasi data,

bahasa-bahasa komputer, sistem operasi, *utilities* dan paket-paket perangkat lunak lainnya.

2. Pengetahuan tentang bisnis secara umum.

Aplikasi bisnis merupakan aplikasi yang sekarang paling banyak diterapkan, oleh sebab itu analis sistem harus mempunyai pengetahuan tentang ini. Pengetahuan ini dibutuhkan supaya analis sistem dapat berkomunikasi dengan pengguna sistem. Pengetahuan tentang bisnis ini meliputi akuntansi keuangan, akuntansi biaya, akuntansi manajemen, sistem pengendalian manajemen, pemasaran, produksi, manajemen personalia, keuangan, tingkah laku organisasi, kebijaksanaan perusahaan, dan aspek-aspek bisnis lainnya.

3. Pengetahuan tentang metode kuantitatif.

Dalam membangun model-model aplikasi, analis sistem banyak menggunakan metode-metode kuantitatif, misalnya pemrograman linier (*linear programming*), pemrograman dinamik (*dynamic programming*), regresi (*regression*), *network*, pohon keputusan (*decision tree*), *trend*, simulasi, dan lain sebagainya.

4. Keahlian pemecahan masalah.

Analis sistem harus mempunyai kemampuan untuk meletakkan permasalahan-permasalahan kompleks yang dihadapi oleh bisnis, memecah-mecah masalah tersebut ke dalam bagian-bagiannya, menganalisisnya dan kemudian merangkainya kembali menjadi suatu sistem yang dapat mengatasi permasalahan-permasalahan tersebut.

5. Keahlian komunikasi antar personel.

Analis sistem harus mempunyai kemampuan untuk mengadakan komunikasi baik secara lisan maupun secara tertulis. Keahlian ini diperlukan di dalam wawancara, presentasi, rapat, dan pembuatan laporan-laporan.

6. Keahlian membina hubungan antar personel.

Manusia merupakan faktor yang kritis di dalam sistem dan watak manusia satu dengan yang lainnya berbeda. Analis sistem yang kaku dalam membina hubungan kerja dengan personel-personel lainnya yang terlibat, akan membuat pekerjaannya menjadi tidak efektif. Apalagi bila analis sistem tidak dapat membina hubungan yang baik dengan pemakai sistem, maka akan tidak mendapat dukungan dari pemakai sistem atau manajemen dan ada kecenderungan pengguna sistem akan mempersulitnya.

PEMROGRAM	ANALIS
1. Tanggungjawab pemrogram terbatas pada pembuatan program komputer.	1. Tanggungjawab analis sistem tidak hanya pada pembuatan program komputer saja, tetapi pada sistem secara keseluruhan
2. Pengetahuan pemrogram cukup terbatas pada teknologi komputer, sistem komputer, utilities dan bahasa-bahasa pemrograman yang diperlukan	2. Pengetahuan analis sistem harus luas, tidak hanya pada teknologi komputer, tetapi juga pada bidang aplikasi yang ditanganinya.
3. Pekerjaan pemrogram sifatnya teknis dan harus tepat dalam pembuatan instruksi-instruksi program.	3. Pekerjaan analis sistem dalam pembuatan program terbatas pada pemecahan masalah secara garis besar.
4. Pekerjaan pemrogram tidak menyangkut hubungan dengan banyak orang, terbatas pada sesama pemrogram dan analis sistem yang mempersiapkan rancang bangun (spesifikasi) programnya.	4. Pekerjaan analis sistem melibatkan hubungan banyak orang, tidak terbatas pada sesama analis sistem, pemrogram, tetapi juga pemakai sistem dan manajer.

**Gambar 1.2**  
Perbandingan Peran Pemrogram dan Analis

Analisis sistem berbeda dengan pemrogram (*programmer*). Pemrogram adalah orang yang menulis kode program untuk suatu aplikasi tertentu berdasarkan rancang bangun yang telah dibuat oleh analis sistem. Perbedaan mendasar antara tugas dan tanggung jawab analis sistem dan pemrogram dapat dilihat pada Gambar 1.2.

Menurut Dennis (2009), pada umumnya sekelompok tim proyek yang biasa terlibat dalam pengembangan sistem disajikan pada Gambar 1.3.

PERAN	TANGGUNGJAWAB
<b>Analis Bisnis</b>	<ul style="list-style-type: none"> <li>- Menganalisa aspek bisnis utama dari sistem</li> <li>- Mengidentifikasi bagaimana sistem akan memberikan nilai bisnis</li> <li>- Merancang kebijakan bisnis baru</li> </ul>
<b>Analis Sistem</b>	<ul style="list-style-type: none"> <li>- Mengidentifikasi bagaimana teknologi dapat meningkatkan proses bisnis</li> <li>- Merancang proses bisnis baru</li> <li>- Merancang sistem informasi</li> <li>- Memastikan bahwa sistem tersebut sesuai dengan standar sistem informasi</li> </ul>
<b>Analis Infrastruktur</b>	<ul style="list-style-type: none"> <li>- Memastikan sistem sesuai (kompatibel) dengan standar infrastruktur.</li> <li>- Mengidentifikasi perubahan infrastruktur yang diperlukan untuk mendukung sistem</li> </ul>
<b>Analis Manajemen Perubahan</b>	<ul style="list-style-type: none"> <li>- Mengembangkan dan melaksanakan rencana manajemen perubahan</li> <li>- Mengembangkan dan melaksanakan rencana pelatihan pengguna</li> </ul>
<b>Manajer Proyek</b>	<ul style="list-style-type: none"> <li>- Memimpin tim analis, programer, perancang teknis, dan spesialis lainnya</li> <li>- Mengembangkan dan memantau rencana proyek</li> <li>- Menetapkan sumber daya</li> <li>- Sebagai titik kontak utama dalam pengembangan proyek</li> </ul>

**Gambar 1.3**  
Peran dan Tanggung Jawab Tim Proyek

Keterangan:

1. Analis Bisnis

Seorang analis bisnis berfokus pada masalah-masalah bisnis di sekitar sistem yang akan dikembangkan. Masalah-masalah ini termasuk mengidentifikasi nilai/manfaat bisnis yang akan dihasilkan oleh sistem, mengembangkan ide mengenai bagaimana proses bisnis dapat ditingkatkan, dan merancang proses, serta kebijakan baru dalam hubungannya dengan analis sistem. Individu ini kemungkinan akan memiliki pengalaman bisnis dan beberapa jenis pelatihan profesional (misalnya, pelatihan analis bisnis untuk sistem akuntansi akan memperoleh *CPA/Certified Public Accountant* di Amerika Serikat, atau *CA/Chartered Accountants* di Kanada). Ia mewakili kepentingan penyelenggara (sponsor) proyek dan pengguna akhir sistem. Seorang analis bisnis membantu dalam fase perencanaan dan desain, tetapi paling aktif dalam fase analisis.

2. Analis Sistem

Seorang analis sistem berfokus pada masalah sistem informasi di sekitar sistem. Dia mengembangkan gagasan bagaimana teknologi informasi dapat meningkatkan proses bisnis, merancang proses bisnis baru dengan bantuan dari analis bisnis, merancang sistem informasi baru, dan memastikan bahwa semua standar sistem informasi telah dipatuhi. Seorang analis sistem memiliki pengalaman pelatihan yang signifikan dan pengalaman dalam analisis dan desain, pemrograman, dan bahkan bidang bisnis. Dia mewakili kepentingan departemen Sistem Informasi dan bekerja secara intensif sepanjang fase-fase sebelum fase implementasi. Sebutan lain untuk analis sistem ini adalah analis informasi (*information analyst*), analis bisnis (*business analyst*), perancang sistem (*systems designer*), konsultan sistem (*systems consultant*), dan ahli teknik sistem (*systems engineer*).

3. Analis Infrastruktur

Seorang analis infrastruktur berfokus pada masalah teknis seputar bagaimana sistem akan berinteraksi dengan infrastruktur teknis organisasi (misalnya, perangkat keras, perangkat lunak, jaringan komunikasi, dan basis data). Tugas seorang analis infrastruktur termasuk memastikan bahwa sistem informasi baru yang dikembangkan sesuai dengan standar organisasi dan mengidentifikasi perubahan infrastruktur yang diperlukan untuk mendukung sistem. Individu ini memiliki pengalaman pelatihan yang signifikan dalam jaringan, administrasi basis data, dan berbagai produk perangkat keras dan perangkat lunak. Dia mewakili kepentingan organisasi dan kelompok sistem informasi pada saat pengoperasian sistem dan memberikan dukungan pada sistem baru setelah diinstal. Seorang analis infrastruktur terlibat di seluruh fase proyek, akan tetapi mungkin kurang terlibat selama fase perencanaan dan analisis.

4. Analis Manajemen Perubahan

Seorang analis manajemen perubahan berfokus pada orang-orang dan masalah manajemen di lingkungan instalasi sistem. Peran orang ini termasuk memastikan ketersediaan dokumentasi dan dukungan yang memadai untuk pengguna, memberikan pelatihan kepada pengguna terkait sistem baru, dan mengembangkan strategi untuk mengatasi resistensi pada masa perubahan sistem. Individu ini harus memiliki pelatihan dan pengalaman yang signifikan dalam perilaku organisasi secara umum dan manajemen perubahan pada khususnya. Seorang analis manajemen perubahan bekerja paling aktif selama fase implementasi, akan tetapi mulai mengkaji dasar untuk perubahan sistem selama fase analisis dan desain.

## 5. Manajer Proyek

Seorang manajer proyek bertanggung jawab untuk memastikan bahwa proyek diselesaikan tepat waktu, sesuai anggaran, dan bahwa sistem memberikan semua manfaat yang dimaksudkan oleh penyelenggara proyek. Peran manajer proyek termasuk mengelola anggota tim, mengembangkan rencana proyek, menugaskan sumber daya, dan menjadi titik kontak utama ketika orang-orang di luar tim memiliki pertanyaan tentang proyek. Individu ini memiliki pengalaman yang signifikan dalam manajemen proyek dan mungkin telah bekerja selama bertahun-tahun sebagai analis sistem sebelumnya. Dia mewakili kepentingan departemen Sistem Informasi dan penyelenggara proyek. Manajer proyek bekerja secara intens di sepanjang fase proyek.



### Latihan

Untuk memperdalam pemahaman Anda mengenai materi di atas, kerjakanlah latihan berikut!

- 1) Jelaskan beberapa alasan yang menyebabkan organisasi perlu mengembangkan sistem informasi!
- 2) Jelaskan tiga hal yang menyebabkan kegagalan organisasi dalam mencapai tujuan pengembangan sistem informasi!
- 3) Jelaskan hal-hal yang harus dipertimbangkan oleh manajemen organisasi ketika melakukan investasi modal dalam rangka pengembangan sistem informasi!
- 4) Jelaskan peran utama tim analis dalam pengembangan sistem informasi!
- 5) Misalkan dalam sebuah proyek pengembangan sistem informasi, Manajer Proyek sedang berhalangan dalam menjalankan tugas-tugas keseharian, sedangkan dalam waktu yang singkat belum dapat dilakukan pergantian. Siapa yang dipandang paling layak menggantikan tugas-tugas harian yang dibebankan kepada Manajer Proyek? Jelaskan alasan Anda!

#### *Petunjuk Jawaban Latihan*

- 1) Sistem perlu dikembangkan dengan beberapa alasan.
  - a) Terdapat permasalahan pada sistem yang lama, sehingga operasi sistem menjadi tidak efisien dan sistem tidak dapat mencegah terjadinya kesalahan operasi, baik yang disengaja maupun yang tidak disengaja.
  - b) Terjadi pertumbuhan pada organisasi menuju ke arah yang lebih baik, sehingga mengakibatkan volume pekerjaan menjadi lebih banyak.
  - c) Adanya kompetisi antar organisasi, sehingga kinerja sistem perlu lebih dioptimalkan

- d) Terjadi perubahan lingkungan, baik lingkungan fisik maupun lingkungan sosial, sehingga organisasi perlu menyesuaikan diri mengikuti pola perubahan tersebut.
  - e) Ada peluang yang memungkinkan untuk mempertahankan eksistensi organisasi atau menuju ke arah yang lebih baik jika memanfaatkan sistem informasi.
  - f) Ada kebijakan pemerintah atau pihak yang terkait yang mengharuskan sistem perlu dikembangkan (pengembangan sistem baru atau perbaikan pada sistem yang ada)
- 2) Tiga penyebab utama kegagalan organisasi mencapai tujuan pengembangan sistem informasi.
- a) Tidak didukung oleh kebijakan manajemen, misalnya karena konsep pengembangan yang tidak bersinergi dengan rencana strategi jangka panjang organisasi.
  - b) Perubahan kebutuhan pengguna (manajemen tingkat atas) yang menuntut adanya sistem informasi yang dapat memberikan keuntungan kompetitif dengan menggunakan teknologi yang lebih canggih, sementara secara teoritis dinyatakan bahwa semakin strategis nilai sebuah sistem informasi, semakin besar resiko kegagalan mencapai tujuan sistem informasi tersebut akibat personel tidak memiliki keahlian yang cukup dalam menguasai teknologi yang ada.
  - c) Organisasi tidak memiliki standar metodologi pengembangan sistem.
- 3) Beberapa hal yang harus dipertimbangkan oleh organisasi ketika melakukan investasi modal dalam pengembangan sistem informasi.
- a) Melakukan investigasi terhadap alternatif-alternatif yang ada, agar dapat menentukan alternatif yang terbaik dalam keterbatasan modal investasi.
  - b) Melakukan penilaian terhadap alternatif yang terpilih, apakah memang merupakan investasi yang menguntungkan. Penilaian dilakukan dengan menggunakan *Cost-Benefit Analysis* (hasil yang diperoleh lebih besar dari biaya untuk memperolehnya).
- 4) Analis adalah pemecah masalah (bisnis, sistem, infrastruktur, dan lain-lain) yang berkesinambungan dalam siklus hidup sistem, baik di tingkat proyek maupun di tingkat organisasi. Analis menjembatani kepentingan pengguna sistem (manajemen dan *end user*) dengan pembuat sistem (*programmer*). Di tingkat proyek, analis juga sebagai perantara komunikasi antara manajer proyek dengan tim proyek lainnya.

- 5) Salah satu dari tim analis (khususnya analis sistem), dengan pertimbangan berikut.
- Analis sistem yang paling mengerti seluk beluk proyek dengan keterlibatannya mewakili departemen sistem informasi secara berkesinambungan dalam siklus hidup pengembangan sistem (terkecuali pada fase implementasi yang melibatkan hanya sedikit peran bagi analis sistem).
  - Analis sistem dibekali dengan berbagai pengetahuan dan keterampilan, baik berupa keterampilan analitis, desain, pemrograman, maupun di bidang bisnis (dengan pengalaman bekerjasama dengan analis bisnis dan analis lainnya).
  - Untuk menjadi Manajer Proyek, seseorang harus memiliki pengalaman bekerja selama bertahun-tahun sebagai analis sistem.



## Rangkuman

Pengembangan sistem (*system development*) dapat diartikan sebagai suatu kegiatan mengembangkan sistem yang baru untuk menggantikan sistem yang lama secara keseluruhan atau memperbaiki sistem yang telah ada.

Suatu sistem perlu dikembangkan tidak hanya karena terdapat masalah pada sistem tersebut, akan tetapi adanya perintah dan kebijakan, baik dari dalam organisasi maupun dari luar organisasi. Pertumbuhan organisasi serta adanya peluang bagi organisasi untuk menuju ke arah perkembangan yang baik juga dapat menjadi alasan sebuah sistem dapat dikembangkan.

Proses pengembangan proyek sistem yang besar memerlukan sejumlah orang dalam bentuk tim kerja. Tim tersebut terdiri atas tim analis pada berbagai bidang, seperti bidang bisnis, bidang sistem, bidang infrastruktur, dan bidang manajemen perubahan, serta tim pemrogram. Tim kerja tersebut dipimpin oleh seorang Manajer Proyek.

Analis memiliki peran sentral dalam pengembangan sistem. Analis sebagai pemecah masalah (bisnis, sistem, infrastruktur, manajemen, dan etika) yang berkesinambungan dalam siklus hidup sistem, baik di tingkat proyek maupun di tingkat organisasi. Analis menjembatani kepentingan pengguna sistem (manajemen dan *end user*) dengan pembuat sistem (*programmer*). Di tingkat proyek, analis juga sebagai perantara komunikasi antara manajer proyek dengan tim proyek lainnya. Dengan demikian, dalam lingkup proyek pengembangan sistem yang kecil dan sederhana, kemungkinan hanya terdapat seorang analis sistem yang merangkap sebagai manajer proyek dan pemrogram.



## Tes Formatif 1

Pilihlah satu jawaban yang paling tepat!

- 1) Dalam pengembangan sistem, sistem lama dapat diartikan sebagai ....
  - A. sistem informasi yang telah menggunakan teknologi/aplikasi database berbasis komputer
  - B. sistem-sistem informasi yang masih berupa pencatatan transaksi pada lembar-lembar arsip
  - C. sistem informasi yang masih menggunakan aplikasi seperti Microsoft Word dan Microsoft Excell dalam pencatatan transaksi
  - D. semua jawaban A, B, dan C benar
- 2) Indikator-indikator berikut ini mengindikasikan adanya permasalahan pada sistem lama sehingga perlu dikembangkan, *kecuali* ....
  - A. hilangnya peluang perusahaan mendapatkan keuntungan penjualan yang lebih besar lagi, walaupun target penjualan telah tercapai
  - B. waktu kerja yang berlebihan dalam menyelesaikan suatu pekerjaan tertentu
  - C. jangkauan distribusi informasi pendek, sehingga banyak pelanggan yang tidak memperoleh informasi
  - D. terjadi kesalahan dalam menghitung gaji bulanan karyawan
- 3) Berikut adalah beberapa prinsip yang harus diperhatikan sewaktu mengembangkan sistem, *kecuali* ....
  - A. sistem yang dikembangkan memerlukan modal yang besar, sehingga perlu menetapkan prioritas pada projek yang lebih menguntungkan
  - B. proyek yang sudah direncanakan, harus terus dijalankan walaupun dinyatakan tidak layak berdasarkan hasil evaluasi, agar peluang keuntungan dapat diperoleh di masa mendatang.
  - C. pengembangan sistem perlu didahului oleh perencanaan yang baik agar dapat berhasil dengan memuaskan
  - D. dokumentasi harus ada untuk pedoman dalam pengembangan sistem
- 4) Analis yang berfokus pada masalah teknis seputar bagaimana sistem akan berinteraksi dengan infrastruktur teknis organisasi (misalnya, perangkat keras, perangkat lunak, jaringan komunikasi, dan basis data) adalah....
  - A. Analis Sistem
  - B. Analis Bisnis
  - C. Analis Teknis
  - D. Analis Infrastruktur

- 5) Perhatikan pernyataan berikut:
- bertugas memastikan ketersediaan dokumentasi dan dukungan yang memadai untuk pengguna sistem
  - memberikan pelatihan kepada pengguna sistem terkait sistem baru
  - mengembangkan strategi untuk mengatasi masalah yang muncul ketika terjadi perubahan sistem
  - paling aktif bekerja selama fase implementasi sistem
- Pernyataan-pernyataan di atas paling relevan dengan tugas ....
- A. Manajer Proyek
  - B. Tim Tester
  - C. Analis Manajemen Perubahan
  - D. Analis Administrasi Sistem
- 6) Sistem hanya dapat dikembangkan jika terdapat permasalahan pada sistem yang lama.
- A. Benar
  - B. Salah
- 7) Kehadiran teknologi baru tidak berpotensi mengakibatkan kegagalan dalam pencapaian tujuan pengembangan sistem, justru sebaliknya selalu dapat mendukung kelancaran pengembangan sistem.
- A. Benar
  - B. Salah
- 8) Pengembangan sistem mesti dilakukan dengan prosedur kerja yang berurutan, untuk menjamin tercapainya pekerjaan secara efisien.
- A. Benar
  - B. Salah
- 9) Manajer Proyek adalah seseorang yang bertugas sebagai pemecah masalah (bisnis, sistem, infrastruktur, manajemen, dan etika) yang berkesinambungan dalam siklus hidup pengembangan sistem, baik di tingkat proyek maupun di tingkat organisasi.
- A. Benar
  - B. Salah
- 10) Sebutan lain untuk analis sistem adalah perancang sistem (*systems designer*).
- A. Benar
  - B. Salah

## 1.26 Konsep Pengembangan Sistem

Cocokkanlah jawaban Anda dengan Kunci Jawaban Tes Formatif 1 yang terdapat di bagian akhir modul ini. Hitunglah jawaban yang benar. Kemudian, gunakan rumus berikut untuk mengetahui tingkat penguasaan Anda terhadap materi Kegiatan Belajar 1.

$$\text{Tingkat Penguasaan} = \frac{\text{Jumlah Jawaban yang Benar}}{\text{Jumlah Soal}} \times 100$$

Arti tingkat penguasaan

<70%	70% - 79%	80% - 89%	90% - 100%
kurang	cukup	baik	baik sekali

Apabila mencapai tingkat penguasaan 80% atau lebih, Anda dapat meneruskan dengan Kegiatan Belajar 2. **Bagus!** Jika masih di bawah 80%, Anda harus mengulangi materi Kegiatan Belajar 1, terutama bagian yang belum dikuasai.

# Metodologi Pengembangan Sistem

Pengembangan sistem informasi berbasis komputer merupakan suatu kegiatan yang sangat kompleks, memerlukan biaya dan sumber daya yang besar, dan merupakan kegiatan yang sangat penting bagi organisasi (Hoffer, 2011). Kompleksitas ini muncul disebabkan oleh beberapa hal:

1. sebuah sistem yang akan dikembangkan melibatkan pemangku kepentingan (*stakeholder*) yang beragam dengan keinginan, kebutuhan, dan agenda yang berbeda-beda;
2. rumitnya struktur proses bisnis dalam organisasi di mana sistem informasi (berbasis komputer) tersebut akan dikembangkan. Sebagaimana diketahui bahwa sistem informasi berbasis komputer dikembangkan merujuk pada sistem bisnis (proses bisnis) organisasi;
3. rumit dan lamanya proses pengembangan sistem informasi itu sendiri, terutama pada sistem-sistem yang berskala besar.

Dalam situasi yang kompleks ini, suatu proyek pengembangan sistem informasi harus dijalankan dengan suatu prosedur kerja yang terstruktur dan terukur agar efisien dalam mencapai tujuan yang telah ditetapkan.

## A. SIKLUS HIDUP PENGEMBANGAN SISTEM

Pengembangan sistem informasi yang berbasis komputer merupakan suatu kegiatan yang cukup kompleks yang membutuhkan banyak sumber daya dan dapat memakan waktu yang lama untuk menyelesaiannya. Proses pengembangan sistem melewati beberapa tahapan (fase), dimulai dari sistem itu direncanakan sampai dengan sistem tersebut diterapkan, dioperasikan, dan dipelihara. Bila operasi sistem yang sudah dikembangkan masih timbul permasalahan-permasalahan yang kritis serta tidak dapat diatasi dalam tahap pemeliharaan sistem, maka sistem tersebut perlu ditinjau kembali untuk dikembangkan dengan mengimplementasikan kembali tahap awal, yaitu tahap perencanaan sistem. Siklus ini disebut dengan siklus hidup suatu sistem (*system life cycle*) yang biasa disebut dengan istilah Siklus Hidup Pengembangan Sistem (*System Development Life Cycle/SDLC*). Daur atau siklus hidup dari pengembangan sistem

merupakan suatu bentuk yang digunakan untuk menggambarkan tahapan-tahapan utama dan langkah-langkah di dalam tahapan tersebut dalam proses pengembangannya.

### 1. Tahapan SDLC

Dalam banyak hal, membangun (mengembangkan) sistem informasi mirip dengan membangun rumah. Pertama, pemilik menjelaskan visi rumah kepada pengembang. Kedua, ide ini ditransformasikan menjadi sketsa dan gambar yang diperlihatkan kepada pemilik (seringkali melalui beberapa gambar) sampai pemilik setuju bahwa gambar-gambar tersebut telah menggambarkan apa yang pemilik inginkan. Ketiga, serangkaian cetak biru (*blueprint*) terperinci dikembangkan yang menyajikan lebih banyak informasi spesifik tentang rumah (misalnya tata letak kamar, penempatan perlengkapan pipa dan jaringan listrik, dan sebagainya). Akhirnya, rumah dibangun mengikuti cetak biru, dan seringkali dengan beberapa perubahan yang dibuat oleh pemilik ketika rumah itu didirikan.

Ide dari *system life cycle* adalah sederhana dan logis. Tiap-tiap bagian dari pengembangan sistem dibagi menjadi beberapa tahapan kerja dan setiap tahapan mempunyai karakteristik tersendiri. Tahapan utama siklus hidup pengembangan sistem terdiri dari tahapan perencanaan sistem (*system planning*), analisis sistem (*system analysis*), desain sistem (*system design*), implementasi sistem (*system implementation*), dan pemeliharaan sistem (*system maintenance*). Tahapan-tahapan seperti ini sebenarnya merupakan tahapan di dalam pengembangan sistem teknik (*engineering system*), misalnya teknik pengembangan konstruksi gedung, mesin, dan teknik pengembangan perangkat lunak (*software engineering*). Istilah *software engineering* merupakan proses pengembangan perangkat lunak yang merupakan sub sistem dari pengembangan sistem informasi. Dengan demikian terdapat kesamaan metode-metode yang dibahas pada pengembangan sistem informasi dengan metode-metode pada pengembangan perangkat lunak (*software*).

Ada banyak varian tahapan dari SDLC sejak tercatat pertama kali digunakan di Inggris pada akhir dekade 1960-an (Avison & Fitzgerald, 2006). Penggunaan tersebut diusulkan oleh *National Computing Centre* (NCC) yang disponsori oleh pemerintah Inggris. Usulan ini sebagai bentuk standardisasi metodologi bagi pengembangan sistem informasi di kalangan pemerintah Inggris maupun pihak swasta.

Pada era tahun 1970-an, beberapa siklus pengembangan sistem dikemukakan oleh beberapa ahli di bidang pengembangan sistem (Jogiyanto, 2008). Pada umumnya tahapan-tahapan utama proses pengembangan sistem yang dikemukakan adalah sama.

*Contoh 1.5:*

Siklus pengembangan sistem yang dikemukakan oleh J.F. Kelly, *Computerized Management Information System* (MacMilan, 1970):

1. Penelitian sistem (*system survey*)
  - a. Definisi ruang lingkup
  - b. Studi penelitian

2. Analisis dan desain sistem (*system analysis and design*)
  - a. Studi penelitian
  - b. Pengumpulan data dan analisis
  - c. Desain sistem
  - d. Rencana implementasi
3. Pengembangan sistem (*system development*)
  - a. Pengembangan
  - b. Pengetesan
  - c. Pengoperasian sistem baru
  - d. Perawatan

*Contoh 1.6:*

Siklus pengembangan sistem yang dikemukakan oleh Charles L. Biggs, Evan G. Birks, William Atkins, *Managing the System Development Process* (Prentice-Hall, 1980):

1. Perencanaan Sistem (*System Planning*)
  - a. Investigasi awal
  - b. Studi kelayakan
2. Kebutuhan-kebutuhan Sistem (*System Requirements*)
  - a. Operasi dan analisis sistem
  - b. Kebutuhan-kebutuhan pemakai
  - c. Pendekatan dukungan secara teknik
  - d. Desain secara konsep dan kaji ulang paket
  - e. Penilaian alternatif dan perencanaan
3. Pengembangan Sistem (*System Development*)
  - a. Rancang bangun sistem secara teknis
  - b. Rancang bangun aplikasi
  - c. Pemrograman aplikasi dan pengetesan
  - d. Prosedur pemakai dan pengendalian
  - e. Latihan untuk pemakai
  - f. Perencanaan implementasi
  - g. Perencanaan konversi
  - h. Pengetesan sistem
4. Implementasi Sistem (*System Implementation*)
  - a. Konversi dan tahap implementasi
  - b. Perbaikan dan pembetulan
  - c. Kaji ulang setelah implementasi
  - d. Perawatan sistem (*system maintenance*)

*Contoh 1.7:*

Siklus pengembangan sistem yang dikemukakan oleh A. Ziya Aktas, *Structured Analysis & Design of Information System* (Prentice-Hall, 1987):

1. Perencanaan (*Planning*)
  - a. Permintaan untuk studi suatu sistem
  - b. Investigasi awal
  - c. Studi kelayakan
2. Analisis (*Analysis*)
  - a. Mendefinisikan kembali masalah
  - b. Memahami sistem yang ada
  - c. Menentukan kebutuhan-kebutuhan pengguna dan hambatan-hambatan pada suatu sistem baru
  - d. Model logika dari pemecahan yang direkomendasikan
3. Desain secara fisik (*physical design*)
  - a. Desain sistem secara umum atau rancang bangun sistem
  - b. Desain terinci atau desain khusus
4. Implementasi atau konstruksi (*implementation or construction*)
  - a. Pembangunan sistem
  - b. Pengetesan
  - c. Instalasi/konversi
  - d. Operasi
  - e. Kajian ulang setelah operasi
  - f. Perawatan dan peningkatan-peningkatan (*maintenance and enhancement*)

Dari beberapa siklus pengembangan sistem yang dikemukakan oleh para ahli, terdapat tiga siklus utama yaitu analisis sistem, desain sistem, dan implementasi sistem. Beberapa penulis memasukkan proses kebijakan dan perencanaan sistem dalam tahapan pengembangan sistem. Tahapan ini sebenarnya adalah sebagai awal terjadinya proyek sistem (*initiation of system project*). Beberapa penulis juga membagi tahapan desain sistem menjadi desain sistem secara umum atau desain sistem secara konsep atau desain sistem secara logika dan desain sistem secara rinci atau desain sistem secara fisik.

Tahap perawatan sistem (*system maintenance*) sebenarnya juga merupakan tahapan setelah pengembangan sistem selesai dilakukan dan sistem dioperasikan. Beberapa penulis menyebutnya sebagai tahap manajemen sistem, karena yang melakukan proses ini sudah bukan analis sistem, akan tetapi manajemen.

Uraian berikut ini menyajikan gambaran secara umum proses yang berlangsung pada setiap tahapan pada SDLC (Sarosa, 2017):

a. Perencanaan dan Studi Kelayakan Proyek Sistem

Studi kelayakan pada intinya mencoba meninjau apakah kebutuhan akan pengembangan sistem informasi baru (baik sistem baru sama sekali maupun pengganti sistem yang lama) layak secara ekonomis maupun dari sisi kelayakan lainnya. Kriteria kelayakan yang perlu diperhatikan dalam suatu studi kelayakan adalah sebagai berikut (Romney, 2012).

- 1) Kelayakan secara hukum, yaitu tidak melanggar aturan maupun undang-undang yang berlaku.
- 2) Kelayakan teknis, yaitu tersedia teknologi untuk membangun sistem serta tersedia sumber daya manusia (tenaga ahli) dalam membangun sistem.
- 3) Kelayakan ekonomi, yaitu biaya membangun sistem lebih kecil daripada manfaat yang diperoleh.
- 4) Kelayakan organisasi dan sosial, yaitu sistem baru dapat diterima oleh para anggota organisasi.

Laporan hasil studi kelayakan menjadi dasar bagi manajemen untuk memutuskan apakah akan meneruskan upaya pengembangan sistem baru atau tidak. Jika keputusannya adalah melanjutkan berarti akan masuk ke tahap berikutnya, yaitu investigasi sistem.

b. Penyelidikan dan Penelitian Sistem

Pada tahap ini dilakukan penelitian dalam rangka pencarian fakta yang lebih terperinci dengan cara menelusuri secara lebih mendetail sistem seperti apa yang dibutuhkan. Penelitian dilakukan terhadap hal-hal, yaitu:

- 1) permasalahan pada sistem yang ada saat ini;
- 2) kebutuhan fungsional dari sistem yang sedang berjalan saat ini, apakah telah terpenuhi atau tidak;
- 3) kebutuhan sistem baru yang mungkin muncul seiring dengan perkembangan waktu, perubahan situasi dan lingkungan bisnis, maupun perkembangan teknologi;
- 4) kendala sistem dan batasan-batasan yang tidak boleh dilampaui;
- 5) jenis dan volume data yang akan diolah dan disimpan;
- 6) pengecualian-pengecualian terhadap kondisi normal yang mungkin akan dihadapi sistem di masa mendatang.

Pencarian fakta dilakukan dengan menggunakan alat pengumpulan data berupa berikut ini.

- 1) Pengamatan terhadap cara kerja dan perilaku pengguna sistem yang ada. Hasil pengamatan dapat memberikan pemahaman akan masalah, kondisi kerja, metode kerja, dan kemampatan.

- 2) Wawancara terhadap individu maupun kelompok pengguna. Wawancara merupakan kesempatan baik untuk bertemu dengan pengguna dari berbagai tingkatan organisasi, mendengarkan apa kebutuhan dan keluhan mereka. Wawancara juga dapat digunakan untuk mengkomunikasikan perubahan yang akan terjadi dan mengurangi resistensi pengguna.
- 3) Kuesioner untuk mendapatkan informasi dari responden dalam jumlah besar dan secara geografis tersebar di berbagai lokasi yang berbeda.
- 4) Studi Pustaka dengan cara menelaah dokumen dan catatan-catatan yang ada untuk mendapatkan pemahaman sistem dan menemukan masalah yang terjadi. Analis sistem harus sadar bahwa terkadang dokumen dan catatan yang ditemukan sudah terlalu tua dan tidak lagi mencerminkan kondisi yang ada saat ini.

Beberapa penulis ahli di bidang pengembangan sistem mengkategorikan kegiatan penyelidikan dan penelitian sistem menjadi bagian dari kegiatan analisis sistem.

c. Analisis Sistem

Data yang diperoleh dari kegiatan investigasi sistem dianalisis untuk menentukan;

- 1) Domain informasi, yang berisi tiga hal yaitu:
  - a) muatan dan hubungan informasi
  - b) aliran informasi
  - c) struktur informasi
- 2) Fungsi-fungsi yang akan dilakukan oleh sistem baru
- 3) Tingkah laku sistem
- 4) Model-model yang menggambarkan informasi, fungsi, dan tingkah laku

Dalam tahapan analisis sistem, pengembang mencoba memahami sistem lama, mengapa dan bagaimana sistem tersebut dibuat, dan bagaimana sistem lama dapat diperbaiki atau dikembangkan.

d. Perancangan (Desain) Sistem

Tujuan dari fase analisis adalah untuk mengetahui kebutuhan bisnis sistem, sedangkan tujuan dari fase desain adalah untuk memutuskan bagaimana membangunnya. Selama bagian awal desain, tim proyek mengubah kebutuhan bisnis sistem menjadi kebutuhan sistem yang menggambarkan detail teknis untuk membangun sistem. Tidak seperti kebutuhan bisnis, yang dikomunikasikan melalui penggunaan model logis dan model data, kebutuhan sistem dikomunikasikan melalui kumpulan dokumen desain dan proses fisik serta model data. Secara bersama-sama, dokumen desain dan model fisik membentuk cetak biru untuk sistem baru yang dikembangkan (Dennis, 2012). Analis bisnis sering

mengalihkan perhatian mereka ke desain bisnis sistem pada tahap proyek ini, sementara analis sistem fokus pada elemen desain yang lebih teknis.

e. **Implementasi Sistem**

Pada tahap implementasi, rancangan yang dihasilkan pada tahap perancangan sistem diwujudkan. Program komputer ditulis, dikompilasi, dan diujicoba. Selanjutnya perangkat keras dan perangkat lunak baru secara terintegrasi dipasang, di-*install* dan diujicoba. Semua aspek sistem informasi yang baru harus teruji dan dalam kondisi baik sebelum terjadi perpindahan (peralihan) sistem.

Salah satu kegiatan pada tahapan penerapan adalah pengendalian kualitas. Semua manual dan dokumentasi sistem diperiksa ulang dan dikonsultasikan dengan para pemangku kepentingan. Para calon pengguna sistem yang baru menjalani pelatihan dan sosialisasi untuk mengakrabkan diri. Jika calon pengguna tidak familiar dengan sistem yang baru, besar kemungkinan proses peralihan akan berjalan dengan sulit.

Untuk menguji sistem informasi, data yang sesungguhnya dapat digunakan untuk uji coba. Data induk mulai dipindahkan dan ditransformasikan dari sistem lama ke sistem baru. Prosedur keamanan sistem juga menjalani uji coba untuk meyakinkan bahwa sistem baru akan terjaga integritas dan keandalannya dari serangan maupun kesalahan yang tidak disengaja. Prosedur pemulihan kembali jika ada gangguan sistem juga diuji coba.

f. **Peninjauan Ulang dan Perawatan Sistem**

Tahapan peninjauan ulang dan perawatan dilakukan setelah sistem yang dibangun telah diimplementasikan dan telah berjalan. Ketika sistem sedang dioperasikan pada kondisi yang riil, sering kali ditemui situasi di mana sistem harus dimodifikasi atau diperbaiki untuk menyesuaikan dengan kondisi saat ini. Suatu sistem harus ditinjau untuk suatu perubahan sebagai akibat dari adanya permasalah (*error*) yang ditemukan pengguna, temuan dari hasil pemeriksaan atau audit, adanya peningkatan kapasitas kerja yang menuntut kebutuhan kapasitas penyimpanan maupun pengolahan data, perubahan lingkungan bisnis (misalnya kebijakan pemerintah), pemanfaatan kemajuan teknologi, dan lain-lain.

*Contoh 1.8:*

Microsoft mengeluarkan kebijakan pembaharuan bagi perangkat lunak produknya secara rutin pada tengah minggu kedua setiap bulannya. Pembaharuan dapat berupa perbaikan untuk menutup celah keamanan, peningkatan kinerja sistem, maupun penyesuaian terhadap kondisi yang dijumpai pengguna pada saat menggunakan sistem.

Perawatan sistem dilaksanakan terus-menerus sampai pada akhirnya sistem tersebut tidak lagi dapat memenuhi kebutuhan organisasi, sehingga harus diganti dengan sistem yang baru. Untuk mendukung kegiatan peninjauan ulang dan pemeliharaan sistem, dibutuhkan tim yang bertanggungjawab terhadap perawatan sistem ini.

## 2. Kelebihan dan Kelemahan SDLC

Sebagai sebuah metode pengembangan yang telah diujicoba dan digunakan dalam berbagai proyek pengembangan sistem informasi dengan berbagai ukuran dan tingkat kompleksitasnya, SDLC memiliki kelebihan dan kelemahan. Beberapa kelebihan SDLC sebagai sebuah metode pengembangan sistem (Avison & Fitzgerald, 2006) adalah:

- a. adanya dokumentasi standar yang dapat mempermudah para pemangku kepentingan dalam berkomunikasi. dokumentasi juga menjamin bahwa spesifikasi sistem yang dikembangkan sesuai dengan kebutuhan pengguna;
- b. dokumentasi yang lengkap juga memudahkan edukasi bagi pengguna sistem;
- c. SDLC memiliki fase-fase dengan batasan dan hasil kegiatan yang jelas. Fase-fase tersebut mempermudah dalam mengelola proyek pengembangan sistem karena tiap fase didefinisikan dengan jelas. Kemajuan dan pencapaian proyek dapat dilihat dan dikendalikan dengan mudah karena sudah terbagi dalam tahapan yang jelas.

Meskipun SDLC memiliki kelebihan, dalam praktiknya sejak diperkenalkan pada era tahun 1970-an, ditemui beberapa kelemahan. Kelemahan-kelemahan tersebut antara lain berikut ini (Avison & Fitzgerald, 2006).

- a. Hanya mampu memenuhi kebutuhan manajemen tingkat bawah.  
Pada umumnya sistem yang dibangun menggunakan SDLC adalah sistem pemrosesan transaksi. Sistem ini memenuhi kebutuhan level manajerial operasional di bagian paling bawah. Kebutuhan manajerial level menengah dan atas (sistem pendukung keputusan atau sistem cerdas) kurang terpenuhi. Meskipun data transaksi yang diolah dapat diringkas dan dilaporkan ke level di atasnya, akan tetapi informasi tersebut kurang memadai untuk pengambilan keputusan taktis dan strategis.
- b. Pemodelan proses yang tidak dinamis.  
Dalam SDLC analisa dilakukan dengan mengamati proses bisnis yang berjalan dan kemudian merancang proses bisnis baru yang lebih baik. Permasalahannya adalah proses bisnis bersifat dinamis dan terus berubah mengikuti perkembangan. Akibatnya mungkin saja ketika sistem baru dikembangkan proses bisnis yang mendasarinya sudah tidak relevan dengan kondisi organisasi, sehingga sistem tidak dapat digunakan.

c. Perancangan yang tidak fleksibel.

Dalam SDLC perancangan sebuah sistem baru didasarkan pada keluaran sistem yang diharapkan (perancangan berbasis keluaran). Dari keluaran yang dikehendaki baru diturunkan menjadi desain masukan, desain basis data, desain proses, dan pengendalian. Jika terjadi perubahan proses bisnis maka desain keluaran dan desain lainnya harus berubah. Perubahan kebutuhan keluaran akan berdampak pada semua bagian sistem dan menyebabkan penundaan penyelesaian proyek.

d. Munculnya ketidakpuasan pengguna.

Munculnya ketidakpuasan pengguna atas spesifikasi sistem yang disepakati pada tahap awal antara para pemangku kepentingan sistem, terutama pengguna. Pengembang sistem sering berasumsi kesepakatan itu tidak akan berubah, namun pada kenyataannya spesifikasi dari pengguna sering berubah seiring berjalannya waktu. Dampaknya ketika sistem dengan spesifikasi awal selesai, pengguna yang kebutuhannya telah berubah tidak lagi terakomodasi.

e. Permasalahan dengan pendekatan ideal.

Jika melihat pada model SDLC, terdapat asumsi bahwa pengembangan sistem berjalan linear, yakni satu tahap diselesaikan sebelum melangkah ke tahap berikutnya. Dalam kenyataannya kondisi tersebut sangat jarang dijumpai. Adanya kebutuhan sistem baru yang tiba-tiba muncul pada saat SDLC dijalankan perlu diakomodasi, padahal dalam SDLC tidak dimungkinkan adanya iterasi (pengulangan langkah) untuk mengakomodasi penambahan fitur atau perubahan rancangan.

Kelemahan-kelemahan SDLC di atas adalah bersifat potensial, artinya ketika menggunakan SDLC pengembang perlu tetap memperhatikan dan mengantisipasi kelemahan yang ada.

## B. PENGERTIAN DAN RAGAM METODOLOGI PENGEMBANGAN SISTEM

Munculnya berbagai istilah yang digunakan dalam pengembangan sistem informasi seringkali membuat bingung bagi pemakainya. Sering terjadi kerancuan, distorsi atau kekaburan makna antara pengertian metode (*method*), metodologi (*methodology*), dan pendekatan (*approach*). Pengertian metode dan metodologi bahkan sering saling dipertukarkan (bermakna sama) atau mempunyai makna yang berbeda pada situasi yang berbeda. Oleh karena itu, sebelum membahas lebih jauh tentang pendekatan dan metodologi pengembangan sistem informasi, terlebih dahulu perlu

kesamaan persepsi tentang pengertian metode, metodologi, dan pendekatan. Pengertian dari setiap istilah akan disarikan dari berbagai sumber sehingga bisa saling melengkapi.

Dalam *WordNet Dictionary* (Prasetyo, 2010), metode (*method*) didefinisikan sebagai: cara untuk mengerjakan atau berbuat; cara untuk mengerjakan sesuatu, khususnya cara yang sistematis yang mengakibatkan pengaturan (biasanya tahapan) dengan urutan yang logis.

Definisi yang lebih detil dipaparkan dalam *Webster's Dictionary* (Prasetyo, 2010) di mana metode didefinisikan sebagai:

1. prosedur atau proses yang teratur, cara yang teratur untuk mengerjakan sesuatu;
2. pengaturan, penjelasan/uraian, pengembangan, atau klasifikasi secara teratur; penyusunan yang jelas dan mudah dimengerti; pengaturan khusus yang sistematis;
3. klasifikasi, mode, atau sistem untuk mengklasifikasi obyek alamiah berdasarkan karakteristik yang umum.

Masih dari *WordNet Dictionary*, metodologi (*methodology*) didefinisikan sebagai: sistem dari metode-metode yang diikuti pada bidang tertentu; cabang dari ilmu filosofi yang menganalisis prinsip-prinsip dan prosedur yang dibutuhkan di suatu bidang tertentu.

*Webster's Dictionary* mendefinisikan metodologi sebagai ilmu pengetahuan dari metode (*science of method*). Dalam *Computer Dictionary* metodologi didefinisikan sebagai kumpulan prosedur dan petunjuk terdokumentasi yang terorganisasi untuk satu atau beberapa tahapan dalam siklus hidup *software*, seperti analisis atau desain. Beberapa metodologi mencantumkan notasi dalam bentuk diagram untuk mendokumentasikan hasil prosedur, pendekatan bertahap (*step by step*) untuk melakukan prosedur, dan tujuan (*objective*) untuk kriteria dalam menentukan apakah kualitas hasil dari prosedur dapat diterima.

Berdasarkan definisi-definisi di atas terlihat jelas perbedaan antara istilah metode dan metodologi. Metode merupakan cara untuk mencapai sesuatu yang diinginkan melalui serangkaian aksi atau tahapan, sedangkan pengertian metodologi lebih luas jika dibandingkan dengan pengertian metode. Suatu metodologi berisi tahapan-tahapan “apa yang harus dilaksanakan”, “bagaimana melaksanakan” tahapan-tahapan tersebut, dan yang paling penting adalah alasan “mengapa” tahapan tersebut harus diambil dalam urutan tertentu. Jadi, metodologi tidak hanya berisi aspek yang langsung terlihat dari sebuah konsep seperti tahapan, prosedur, teknik, alat bantu (*tools*), dan dokumentasi yang diperlukan dalam mencapai suatu tujuan, akan tetapi di dalamnya juga berisi aspek yang tidak bisa langsung dirasakan yaitu filosofi. Filosofi bisa diartikan sebagai teori dan asumsi yang dipercayai oleh pengarang/pemrakarsa metodologi ketika membangun/memodelkan metodologi (Avison & Fitzgerald, 2006).

Keberadaan aspek filosofi inilah yang membedakan pengertian metodologi dengan pengertian metode. Walaupun kedua pengertian ini jelas berbeda, namun di

bidang sistem informasi secara pragmatis pengertian metodologi dapat dianggap sama dengan pengertian metode.

Menurut Avison dan Fitzgerald (2006), definisi dari metodologi pengembangan sistem informasi atau yang sering disebut sebagai *information systems development method* (ISDM) adalah kumpulan prosedur, teknik, alat, dan alat bantu pendokumentasian yang membantu para pengembang membangun sistem informasi. Iivari (2000) mendefinisikan metodologi pengembangan sistem informasi sebagai kumpulan prosedur berorientasi tujuan yang didukung dengan teknik dan alat bantu serta prinsip-prinsip, yang digunakan sebagai pedoman dalam bekerja atau bekerja sama di antara berbagai pihak (*stakeholder*) yang terlibat dalam pengembangan aplikasi sistem informasi. Whitten (2001) mendefinisikan metodologi pengembangan sistem sebagai suatu proses pengembangan sistem yang formal dan presisi yang mendefinisikan serangkaian aktivitas, metode, *best practices*, dan tool yang terautomatisasi bagi para pengembang dalam rangka mengembangkan dan merawat sebagian besar atau keseluruhan sistem informasi atau software.

Terlihat bahwa definisi-definisi tentang metodologi pengembangan sistem informasi di atas saling melengkapi, tidak ada pertentangan antara satu dan yang lainnya. Setiap metodologi memasukkan komponen teknik dan alat bantu sebagai bagian yang tak terpisahkan dari sebuah metodologi.

Teknik (*technique*) adalah cara mengerjakan aktivitas tertentu dalam proses pengembangan sistem informasi. Teknik juga dapat diartikan sebagai cara memodelkan (*a way of modelling*). Suatu metodologi mungkin memasukkan sejumlah teknik yang dipilih berdasarkan kecocokan dengan kerangka tujuan umum dari metodologi.

Contoh:

- *Data Flow Diagram* (DFD)
- *Entity Relationship Diagram* (ERD)
- *Use-Case Diagram*
- *Decision Table*, dan sebagainya.

Alat bantu (*tools*) adalah alat bantu komputer (normalnya berbentuk aplikasi berbasis komputer) yang biasanya digunakan untuk mendukung aktivitas dalam suatu metodologi. Alat bantu memungkinkan beberapa tugas pengembangan bisa dikerjakan secara otomatis atau semi otomatis. Fungsionalitas pemodelan yang didukung oleh alat bantu di antaranya adalah abstraksi dari obyek sistem ke dalam model, pengecekan konsistensi dari model, konversi hasil dari satu bentuk model atau representasi ke bentuk yang lain, dan menyediakan spesifikasi untuk keperluan kajian ulang.

Contoh:

- Paket aplikasi *Computer-Aided Software Engineering* (CASE),
- Aplikasi Visio

- Aplikasi *Rational Rose* untuk pemodelan berorientasi objek
- Aplikasi *Enterprise Architect* untuk pemodelan berorientasi objek
- Aplikasi *Macromedia Dreamweaver*, dan sebagainya

Bersumber beberapa definisi di atas, maka dapat dikatakan bahwa metodologi disusun oleh sejumlah konsep beserta hubungan antara satu dan lainnya. Konsep ini diaplikasikan melalui teknik pemodelan untuk merepresentasikan model dari sistem informasi. Bagaimana model tersebut dibuat, dimanipulasi, dan digunakan diwujudkan melalui serangkaian proses/tahapan. Pilihan dan pengaturan dari teknik dan proses yang digunakan didasarkan pada tujuan pengembangan serta nilai dan asumsi (filosofi) tertentu.

Terdapat beberapa cara dalam mengklasifikasikan metodologi pengembangan sistem informasi, di antaranya dengan mengetahui latar belakang maupun pendekatan (*approach*) yang digunakan. Beberapa metodologi dikembangkan dengan latar belakang *science* dan yang lainnya dikembangkan berdasarkan pada pengalaman praktis atau dengan pendekatan sebuah sistem. Pendekatan yang digunakan juga bermacam-macam, ada yang menggunakan pendekatan *process-oriented*, *data-oriented*, *user-oriented*, dan sebagainya. Wajar jika kemudian muncul berbagai cara dalam mengelompokkan metodologi pengembangan sistem informasi.

Avison dan Fitzgerald (2006) mengklasifikasikan metodologi pengembangan sistem informasi menjadi dua bagian berdasarkan pendekatan filosofi yang digunakan, yaitu *science paradigm* dan *system paradigm*. Istilah lain yang sering digunakan untuk menyebut *science paradigm* adalah *hard approach* sedangkan untuk *system paradigm* sering disebut sebagai *soft approach* atau *holistic approach*.

Metodologi *hard approach*: mengasumsikan fakta yang kongkrit dan menggunakan teknik *top-down*.

Contoh:

- SSAD (*Structured System Analysis and Design*)
- YSM (*Yourdon System Methods*)
- IE (*Information Engineering*)
- Merise

Metodologi *soft approach*: mengasumsikan persepsi yang berbeda di antara user serta pengembangan mini-system akan membuat perbedaan sistem secara menyeluruh.

Contoh:

- SSM (*Soft Systems Methodology*)
- ETICHS (*Effective Technical and Human Implementation of Computer-Based Systems*)

Dungaria (Prasetyo, 2010) menambahkan dua kategori metodologi ke dalam pendapat Avison (2006) sehingga menjadi empat kelompok, yaitu:

1. *Hard methodology*, merupakan metodologi terstruktur yang didasarkan dekomposisi fungsional;
2. *Soft methodology*, menekankan pada aspek sosial dan manusia;
3. *Hybrid methodology*, merupakan kombinasi dari gabungan *hard methodology* dan *soft methodology*;
4. *Specialised methodology*, merupakan metodologi yang digunakan untuk pengembangan sistem informasi khusus (spesial).

Kategori *hard methodology* dapat dikelompokkan lagi menjadi empat kelompok, yaitu:

1. *Structured methodology*, menekankan cara yang terstruktur dalam melakukan analisis dan perancangan.

Contoh:

- a. SSAD (*Structured System Analysis and Design*)
- b. IE (*Information Engineering*)
- c. STRADIS (*Structured Analysis, Design, and Implementation of Information Systems*)
- d. YSM (*Yourdon System Method*)
- e. JSD (*Jackson System Development*)

2. *Formal methodology*, menekankan penggunaan notasi matematika dalam spesifikasi dan perancangan.

Contoh:

- a. Z-numbers method
- b. VDM (*Vienna Development Method*)

3. *Object Oriented (OO) methodology*, menggunakan konsep *object* dalam melakukan analisis, perancangan dan implementasi.

Contoh:

- a. OMT (*Object Modelling Technique*)
- b. Booch method

Beynon-Davis (2003) mempunyai pandangan lain tentang pengelompokan metodologi pengembangan sistem informasi. Dalam pandangan Beynon-Davis, metodologi pengembangan sistem informasi dapat dibagi menjadi tiga kelompok utama.

1. Metode terstruktur (*structured methods*)

Menggunakan model linear dalam proses pengembangan. Setiap tahapan diidentifikasi dengan jelas, termasuk input dan output dari setiap tahapan. Pemodelan data dan proses dilakukan dengan kerangka kerja yang terstruktur.

Contoh:

- a. STRADIS (*Structured Analysis, Design and Implementation of Information System*)
- b. SSAD (*Structured System Analysis and Design*)

2. Metode *Rapid Application Development* (RAD)

Menggunakan model iterasi di dalam proses pengembangan dan secara umum menspesifikasikan tahapan level tinggi (*high-level phase*) berdasar beberapa bentuk prototype. Metode RAD secara umum dapat disesuaikan dengan situasi yang ada karena tidak memberikan detail teknik yang digunakan.

Contoh: DSDM (*Dynamic Systems Development Method*)

3. Metode berorientasi obyek (*object-oriented methods*)

Fokus pada penggunaan obyek secara konsisten mulai dari tahap analisis, perancangan, sampai implementasi sistem informasi. Proses pengembangan dengan pendekatan berorientasi obyek biasanya memanfaatkan model kontingensi, di mana kadang-kadang menggunakan model linear, kadang-kadang menggunakan model iteratif.

Contoh:

- a. UML (*Unified Modelling Language*)
- b. RUP (*Rational Unified Process*)

Teknik pengelompokan yang lebih luas dilakukan oleh Avison dan Fitzgerald (2006) dengan mengelompokkan metodologi pengembangan sistem informasi berdasarkan pada tema (*theme*) dalam pengembangan sistem informasi. Dalam pandangan Avison dan Fitzgerald, tema pengembangan sistem informasi bisa didasarkan pada:

1. pendekatan sistem atau organisasinya (*organizational theme*);
2. pendekatan pemodelannya (*modelling theme*);
3. pendekatan perekayaan perangkat lunaknya (*engineering theme*);
4. pendekatan orang (*stake holder*) yang terlibat dalam pengembangan sistem informasi (*people theme*).

Berdasarkan pada pendekatan tema pengembangan, maka metodologi pengembangan sistem informasi bisa dikelompokkan menjadi enam kategori, yaitu:

1. Metodologi berorientasi proses (*process-oriented methodologies*), yang fokus pada aspek pemodelan proses.

Contoh:

- a. STRADIS (*Structured Analysis, Design and Implementation of Information Systems*)
- b. YSM (*Yourdon Systems Method*)

2. Metodologi campuran (*blended methodologies*), yang fokus pada campuran antara aspek pemodelan proses dan aspek pemodelan data.

Contoh:

- a. SSAD (*Structured System Analysis and Design*)
- b. Merise
- c. IE (*Information Engineering*)

3. Metodologi berorientasi obyek (*object-oriented methodologies*), yang fokus pada aspek pemodelan berorientasi obyek

Contoh:

- a. OOA (*Object Oriented Analysis*)
- b. RUP (*Rational Unified Process*)

4. Metodologi pengembangan cepat (*Rapid Development Methodologies*), yang lebih fokus pada aspek rekayasa dan konstruksi softwarenya.

Contoh:

- a. JMRAD (*James Martin's*)
- b. XP (*Extreme Programming*)
- c. DSDM (*Dynamic Systems Development Method*)

5. Metodologi berorientasi orang (*people-oriented methodologies*), yang lebih fokus pada aspek sosial (orang yang terlibat dalam pengembangan sistem informasi).

Contoh:

- a. ETHICS (*Effective Technical and Human Implementation of Computer-Based Systems*)
- b. KADS

6. Metodologi berorientasi organisasi (*organizational-oriented methodologies*), yang lebih fokus pada sistem/organisasinya.

Contoh:

- a. SSM (*Soft Systems Methodology*)
- b. PRINCE (*Project in Controlled Environments*)

Teknik klasifikasi lainnya yang mendasarkan pada strategi-strategi alternatif yang dapat dipilih dalam pengembangan sistem informasi disampaikan oleh Whitten (2001). Pengembangan sistem informasi bisa menggunakan dua opsi (pilihan), yaitu membangun solusi *software* sendiri (*model-driven*) atau dengan membeli paket *software* komersial yang sudah jadi (*product-driven*).

Pendekatan *model-driven* menekankan pada:

1. pemodelan proses (process-centric model);
2. pemodelan data (data-centric model);
3. pemodelan berorientasi obyek (object-oriented model).

Pendekatan *product-driven* menekankan pada:

1. penggunaan prototyping;
2. penulisan kode program secepat mungkin, contoh: Extreme Programming (XP).

Aktaz (Jogiyanto, 2008) mengelompokkan metodologi pengembangan sistem terstruktur ke dalam dua bagian, yaitu: metodologi pengembangan sistem yang dibuat atau diusulkan oleh para peneliti/penulis buku/konsultan dan metodologi yang dikembangkan oleh *system house* dan pabrik-pabrik perangkat lunak. Metodologi yang dibuat oleh *system house* dan pabrik-pabrik perangkat lunak umumnya tersedia secara komersial dalam bentuk paket program. Metodologi ini diklasifikasikan sebagai *prescriptive methodologies*. Metodologi yang diusulkan oleh para peneliti/penulis/konsultan diklasifikasikan dalam metodologi pemecahan fungsional (*functional decomposition methodologies*) dan metodologi orientasi data (*data-oriented methodologies*). Ketiga kelompok metodologi ini dapat dirinci sebagai berikut.

1. *Prescriptive Methodologies*

*Contoh:*

- a. ISDOS (*Information System Design and Optimization System*), merupakan perangkat lunak yang dikembangkan untuk mengotomatisasi proses pengembangan sistem informasi.
- b. PRIDE, merupakan suatu perangkat lunak terpadu yang baik untuk analisis/desain sistem terstruktur, manajemen data, manajemen proyek, dan pendokumentasian.
- c. SDM/70 (*System Development Methodology/70*), merupakan suatu perangkat lunak berisi kumpulan metode, estimasi, dokumentasi dan petunjuk administrasi guna membantu pengguna untuk mengembangkan dan merawat sistem secara efektif.

2. *Functional Decomposition Methodologies*

Menekankan pada pemecahan sistem ke dalam subsistem-subsistem yang lebih kecil, sehingga akan lebih mudah untuk dipahami.

*Contoh:* HIPO (*Hierarchy plus Input-Process-Output*), SR (*Stepwise Refinement*) atau ISR (*Iterative Stepwise Refinement*), Information-hiding

3. *Data-Oriented Methodologies*

Menekankan pada karakteristik data yang akan diproses.

a. *Data-flow Oriented Methodologies*

Pemecahan sistem ke dalam modul-modul berdasarkan tipe elemen data dan tingkah laku logika modul dalam sistem.

*Contoh:* SADT (*Structured Analysis and Design Technique*), Composite Design, SSAD (*Structured System Analysis and Design*)

b. *Data Structure Oriented Methodologies*

Menekankan pada struktur dari input dan output di sistem.

Contoh: JSD (*Jackson's System Development*), W/O (*Warnier/Orr*)

Metodologi pengembangan sistem informasi juga dapat diklasifikasikan menjadi:

1. metodologi pengembangan sistem melalui pendekatan siklus hidup pengembangan sistem (*system development life cycle/SDLC*) (Sulistanta, 2017).  
*Contoh: Waterfall, RAD (Rapid Application Development), Prototyping, Spiral, Incremental, Agile Development;*
2. metodologi pengembangan sistem melalui metode alternatif (Jogiyanto, 2010)  
*Contoh: Paket, Outsourcing, End-User Development.*

Metodologi pengembangan sistem berbasis siklus hidup pengembangan sistem dan metode alternatif akan dibahas lebih jauh pada Modul 2 (Perencanaan Sistem).

### C. PENTINGNYA METODOLOGI PENGEMBANGAN SISTEM

Pada masa-masa awal digunakannya komputer sebagai pendukung pengembangan sistem informasi, perhatian dicurahkan pada bagaimana menggunakan komputer untuk kegiatan pembuatan program aplikasi. Penekanan pada pemrograman aplikasi (perangkat lunak) menyebabkan perhatian besar diarahkan pada kemampuan pemrogram (*programmer*) dalam menyusun program aplikasi (perangkat lunak). Pendidikan dan pelatihan untuk pemrogram difokuskan pada aspek teknis pemrograman, seperti bahasa pemrograman, struktur data, algoritma, dan lain-lain, dengan harapan dapat menghasilkan program aplikasi yang berkualitas. Sayangnya pelatihan peningkatan sumber daya manusia tersebut belum memikirkan bahwa kemampuan pemrogram dalam menciptakan program aplikasi harus dibarengi dengan aspek kemampuan berkomunikasi yang baik kepada pemangku kepentingan lainnya dalam sistem informasi yang akan dikembangkan. Kurangnya kualitas komunikasi antara pemrogram dengan para pemangku kepentingan (terutama pengguna sistem atau *user*) mengakibatkan kepentingan-kepentingan maupun kebutuhan mereka tidak terakomodasi dengan baik dalam sistem informasi yang dikembangkan. Hal ini berdampak pada terciptanya sebuah aplikasi perangkat lunak yang tidak sesuai dengan kebutuhan pengguna. Perangkat lunak yang secara teknis dihasilkan dengan baik menjadi tidak terpakai karena tidak sesuai dengan kebutuhan pengguna.

Terlepas dari kurangnya pendidikan dan pelatihan formal yang didapatkannya, banyak pemrogram yang tidak menggunakan metodologi formal dalam proyek pengembangan perangkat lunak. Para pemrogram lebih mengandalkan pengalaman pribadi atau *best practice* dalam mengembangkan perangkat lunak, sehingga sangat sulit membuat estimasi kapan proyek pengembangan perangkat lunak akan berakhir untuk selanjutnya dapat diterapkan (digunakan). Kesalahan dalam estimasi dapat

mengakibatkan penyelesaian suatu proyek sistem menjadi mundur dari jadwal yang telah ditetapkan dan disepakati. Terjadi antrian pada proyek pengembangan aplikasi lainnya yang akan diselesaikan.

Adanya metodologi pengembangan sistem secara formal akan mendorong terjadinya hal-hal berikut (Avison & Fitzgerald, 2006).

1. Meningkatnya kesadaran akan perlunya analisa permasalahan dan kebutuhan sistem, serta perancangan sistem yang lebih baik sebelum sistem dibuat.
2. Terciptanya solusi sistem yang terintegrasi bagi manajemen organisasi yang besar dan kompleks.
3. Meningkatnya kesadaran akan perlunya suatu metode yang dapat memandu para pengembang sistem untuk menghasilkan aplikasi dan sistem informasi yang berkualitas.

Sebagai upaya untuk meminimalkan permasalahan yang timbul dalam proses pengembangan sistem informasi, diciptakan suatu metodologi pengembangan sistem. Metodologi ini tidak hanya memuat sekumpulan alat, teknik, dan cara/metode, akan tetapi juga dilandasi oleh filosofi tertentu. Dengan filosofi yang berbeda, tahapan, dan alat bantu, serta metode yang terkandung dalam metodologi pengembangan sistem menjadi beragam pula. Meskipun berbeda, pada dasarnya setiap metodologi pengembangan sistem berusaha mencapai suatu tujuan yang sama, yaitu:

1. memperoleh secara akurat apa yang menjadi kebutuhan pengguna dalam sistem yang dikembangkan;
2. adanya metode pengembangan yang sistematis, yang dapat memantau dan mengendalikan kemajuan proyek pengembangan dengan baik;
3. menghasilkan sistem yang menggunakan waktu pengembangan, sumber daya, dan biaya pengembangan yang efisien;

Ada banyak proyek pengembangan sistem yang karena penyelesaiannya tidak sesuai dengan jadwal yang sesungguhnya, menyebabkan timbulnya biaya pengembangan yang besar, serta memunculkan masalah yang lain.

*Contoh:*

Ketika Microsoft mengembangkan perangkat lunak Sistem Operasi Windows Vista, Microsoft banyak mengalami kendala sehingga terlambat diluncurkan ke pasaran. Windows Vista yang semestinya dikembangkan dan diluncurkan 2 atau 3 tahun setelah Windows XP, ternyata baru dapat diluncurkan setelah lebih dari 4 tahun dari pendahulunya Windows XP, yaitu pada tahun 2006. Imbas memiliki kendala dalam proses pengembangannya, selepas diluncurkan, Vista memperoleh banjir kritikan, dimulai dari masalah-masalah teknis hingga pada masalah waktu pengembangan yang cukup lama yang tidak dibarengi dengan kinerja sistem yang lebih baik dari produk yang telah ada sebelumnya (Windows XP). Kegagalan

yang dialami Windows Vista adalah sebuah siklus yang diderita Microsoft pada saat itu, yaitu biaya pengembangan yang besar, di lain pihak Windows Vista hanya memperoleh *net market share* sekitar 0,78 persen pengguna di seluruh dunia hingga dirilisnya Windows 7 dua tahun setelah peluncuran Windows Vista. Jika dibandingkan dengan pendahulunya Windows XP, Windows XP memperoleh *net market share* 8,45 persen pengguna di seluruh dunia (Zaenuddin, 2017).

4. menghasilkan sistem informasi yang terdokumentasi dengan baik dan mudah dipelihara;
5. mengakomodasi perubahan yang harus dilakukan selama proses pengembangan sistem.

Metodologi pengembangan sistem merupakan suatu cara yang dapat membantu para pengembang mengendalikan proyek pengembangan sistem sehingga proyek sistem dapat diselesaikan tepat waktu, menghabiskan anggaran dengan efisien sesuai estimasi penganggaran, dan berfungsi dengan baik sesuai dengan spesifikasi teknis yang dituangkan dalam dokumen rancangan sistem. Metodologi juga dapat menyelaraskan antara penggunaan alat dan teknik yang tepat dengan aspek perilaku manusia sebagai pemangku kepentingan.

Alasan rasional lainnya mengenai perlunya menggunakan metodologi dalam pengembangan sistem dikemukakan oleh Avison dan Fitzgerald (2006).

1. Menghasilkan Produk Akhir Berupa Sistem Yang Lebih Baik, dengan beberapa indikator:
  - a. waktu pengembangan yang cepat dan ekonomi,
  - b. sistem yang dikembangkan dapat memenuhi kebutuhan fungsional bagi pengguna,
  - c. mudah dipelajari dan dioperasikan,
  - d. tersedia dokumentasi yang memadai untuk membantu komunikasi antar pengguna, pengembang sistem, dan pihak manajemen,
  - e. modular, sehingga walaupun terintegrasi satu sama lainnya jika terjadi kerusakan atau modifikasi pada suatu komponen, pengaruhnya kecil terhadap komponen lain atau sistem itu sendiri,
  - f. sistem yang dikembangkan kompatibel dengan sistem lainnya di dalam maupun di luar organisasi,
  - g. dapat dijalankan di berbagai *platform* teknologi yang berbeda,
  - h. sistem yang fleksibel (mudah dimodifikasi),
  - i. mudah dipelihara,
  - j. sistem mampu bertahan dalam kondisi tertentu yang tidak menguntungkan (*fail-safe* dan *fault-tolerant*),
  - k. aman.

2. Kematangan Proses Pengembangan Sistem

Merujuk pada *IEEE Standard Glossary of Software Engineering Technology* (1990), kualitas sebuah sistem (perangkat lunak) diukur tidak hanya pada produk jadinya, namun juga diukur pada kematangan proses pengembangannya. Perbaikan proses pengembangan sistem dapat dicapai dengan manajemen proyek yang baik menggunakan suatu metodologi pengembangan tertentu.

3. Standardisasi Proses Pengembangan Sistem

Penggunaan proses yang sama dalam mengembangkan sistem yang berbeda menghasilkan beberapa keuntungan, yaitu:

- a. tenaga pengembang dapat berpindah dari satu proyek pengembangan ke proyek lainnya tanpa perlu pelatihan ulang;
- b. standar kerja dan hasil yang setara dapat dicapai oleh siapapun selama mengikuti metodologi yang standar;
- c. pengembangan sistem yang terintegrasi dapat dicapai dengan lebih mudah.

## D. PENGARUH METODOLOGI TERHADAP PENGEMBANGAN SISTEM

### Studi Kasus

*Standish Group* dalam CHAOS Report pada tahun 2009 mengungkapkan data statistik yang terkait dengan proyek-proyek pengembangan sistem informasi, seperti berikut.

1. 32% proyek berhasil (tepat waktu, tidak melebihi anggaran, dan fungsionalitas yang dijanjikan terwujud sepenuhnya).
2. 44% proyek bermasalah (terlambat, melebihi anggaran, dan fungsionalitas yang dijanjikan tidak sepenuhnya terwujud).
3. 24% proyek gagal (dibatalkan atau tidak pernah digunakan).

Temuan di atas tentu mengundang pertanyaan, mengapa kegagalan dan permasalahan pengembangan sistem masih cukup tinggi? Bukankah lahirnya metodologi sebagai suatu upaya untuk mengurangi kegagalan dan meningkatkan kualitas? Apa yang terjadi dengan metodologi pengembangan sistem informasi?

Davis (2004) telah mengungkapkan fakta mengenai penggunaan metodologi pengembangan sistem untuk memberikan dukungan terhadap pertanyaan yang terkait dengan laporan *Standish Group*. Dimulai dari dekade 1960-an sampai dekade awal 2000-an terdapat beberapa terobosan dalam metodologi pengembangan sistem informasi, antara lain:

1. Metode Terstruktur (tahun 1955 – 1986)
2. Teknologi Berorientasi Objek (tahun 1980 – 2000)
3. Kematangan Proses, terkait ISO dan CMMI (tahun 1990 – 2005)

4. Prototyping (tahun 1985 – sekarang)
5. CASE Tools (tahun 1988 – sekarang)
6. Daur Ulang Modul/*Reuse* (mulai tahun 1988)
7. Comquats (mulai tahun 1988)

Selain metode-metode tersebut saat ini dikenal juga berbagai jenis metode *agile system development* seperti *Extreme Programming*, *Scrum*, *Crystal*, dan lain sebagainya. Dampak berbagai terobosan tersebut ternyata tidak terlalu signifikan dalam upaya menyempurnakan kualitas sistem informasi yang dihasilkan. Tiap terobosan memang membawa dampak yang baik terhadap sebagian proyek pengembangan sistem informasi, namun tidak pada yang lainnya. Penyebabnya antara lain sebagai berikut.

1. Permasalahan disebabkan oleh pemilihan metodologi pengembangan sistem yang kurang tepat. Tiap metodologi dibangun dengan landasan filosofi dan teori tertentu untuk mengatasi atau menyelesaikan masalah yang spesifik.

*Contoh:*

*Agile system development* terbukti baik dan cepat untuk menghasilkan sistem informasi atau program aplikasi (perangkat lunak) dalam skala kecil atau individual, akan tetapi akan mengalami kesulitan ketika digunakan untuk mengembangkan sistem dalam skala besar.

2. Meski telah memilih metodologi yang tepat, akan tetapi jika tidak mempraktikkan dengan benar akan mengalami permasalahan. Perbedaan yang signifikan sering terjadi antara bagaimana metodologi seharusnya dijalankan dan kenyataan di lapangan (Kautz, 2004). Perbedaan ini sering terjadi karena adanya variasi dalam pemahaman dan penguasaan metodologi karena kesalahan pemilihan metodologi.

Hal yang dapat dipelajari dari kedua permasalahan tersebut adalah: ketika menggunakan suatu metodologi pengembangan sistem haruslah dipahami dengan benar latar belakang, asumsi, landasan teoritis dan filosofi metodologi, alat, teknik dan cara penggunaannya. Kesalahan dalam pemilihan metodologi dan kurangnya penguasaan atas metodologi oleh pengembang sistem dapat menjadi masalah yang serius.



### Latihan

Untuk memperdalam pemahaman Anda mengenai materi di atas, kerjakanlah latihan berikut!

- 1) Terdapat banyak varian tahapan pada siklus hidup pengembangan sistem (*system development life cycle*) yang dikemukakan oleh para ahli pengembangan sistem informasi, namun pada umumnya memiliki kesamaan pada banyak hal. Sebutkan

dan jelaskan tiga tahapan yang paling utama dalam siklus hidup pengembangan sistem!

- 2) Jelaskan alasan mengapa metodologi pengembangan diperlukan dalam proses pengembangan sistem informasi!
- 3) Sebutkan dan jelaskan lima contoh metodologi pengembangan sistem berdasarkan kategori pendekatan tema pengembangan sistem!

*Petunjuk Jawaban Latihan*

- 1) Tiga tahapan utama dalam siklus hidup pengembangan sistem.
  - a) Analisis Sistem, merupakan kegiatan yang bertujuan untuk memahami permasalahan pada sistem yang lama, mengusulkan kebutuhan-kebutuhan pada sistem yang baru, serta memodelkan kebutuhan sistem secara logis.
  - b) Desain/perancangan sistem, merupakan kegiatan yang bertujuan untuk mengubah atau menerjemahkan model-model logis kebutuhan sistem menjadi kebutuhan sistem yang menggambarkan detail teknis (model fisik) untuk membentuk cetak biru sistem yang akan dikembangkan.
  - c) Implementasi, merupakan kegiatan yang bertujuan untuk menerjemahkan rancangan-rancangan fisik menjadi wujud sistem informasi melalui kegiatan penulisan program, pengetesan program, dan ujicoba penerapan sistem.
- 2) Alasan yang menyebabkan perlunya penggunaan metodologi pengembangan dalam proses pengembangan sistem informasi:  
Pengembangan sistem adalah suatu kegiatan yang sangat kompleks, yang melibatkan banyak pemangku kepentingan dengan beragam keinginan, kebutuhan, dan agenda masing-masing. Sering kali sebuah organisasi memiliki struktur proses bisnis yang rumit, di mana proses bisnis organisasi menjadi acuan dalam proses pengembangan sistem informasi. Demikian juga dengan kompleksitas proyek sistem yang melibatkan banyak pekerjaan besar dan melibatkan banyak sumber daya, serta waktu pelaksanaan yang panjang. Dalam situasi yang kompleks seperti ini, suatu proyek pengembangan sistem harus direncanakan dan dijalankan menggunakan suatu prosedur kerja yang terstruktur dan terukur agar dapat efektif dan efisien dalam mencapai tujuan proyek yang telah ditetapkan.
- 3) Tiga contoh metodologi pengembangan sistem berdasarkan kategori pendekatan tema pengembangan sistem.
  - a) Metodologi Berorientasi Proses, yang berfokus pada aspek pemodelan proses. Contoh: *Structured Analysis, Design and Implementation of Information Systems* (STRADIS), dan *Yourdon Systems Method* (YSM).

- b) Metodologi Berorientasi Obyek, yang berfokus pada aspek pemodelan berorientasi obyek. Contoh: *Object Oriented Analysis* (OOA) dan *Rational Unified Process* (RUP).
- c) Metodologi Pengembangan Cepat, yang berfokus pada aspek rekayasa dan konstruksi. Contoh: *Extreme Programming* (XP) dan *Dynamic Systems Development Method* (DSDM).



## Rangkuman

Proses pengembangan sistem melewati beberapa tahapan (fase), dimulai dari fase perencanaan, analisis, desain, sampai dengan sistem tersebut diimplementasikan, dioperasikan, dan dipelihara. Bila dalam operasi sistem masih timbul permasalahan-permasalahan yang tidak dapat diatasi dalam tahap pemeliharaan sistem, maka sistem tersebut perlu ditinjau kembali untuk dikembangkan dengan mengimplementasikan kembali tahap awal. Proses ini membentuk sebuah siklus yang dikenal dengan nama siklus hidup pengembangan sistem (*system development life cycle/SDLC*). SDLC menjadi dasar munculnya berbagai ragam metodologi pengembangan lain yang telah disesuaikan dengan kebutuhan pengembangan sistem.

Munculnya berbagai istilah yang digunakan dalam pengembangan sistem informasi seringkali membuat bingung bagi pemakainya, terutama kerancuan dan kekaburuan makna dari istilah metode dan metodologi. Pengertian metode dan metodologi bahkan sering saling dipertukarkan (bermakna sama) atau mempunyai makna yang berbeda pada situasi yang berbeda. Namun beberapa pakar di bidang sistem informasi berpandangan bahwa secara pragmatis pengertian metodologi dapat dianggap sama dengan pengertian metode. Demikian juga dengan ragam metodologi pengembangan sistem, seringkali membuat bingung dalam mempelajari dan memahaminya. Munculnya ragam metodologi pengembangan sistem informasi diakibatkan oleh adanya perbedaan sudut pandang para penyusun metodologi dalam merumuskan metodologi tersebut, seperti sudut pandang latar belakang maupun pendekatan yang digunakan dalam pengembangan sistem, sudut pandang filosofi, tema pengembangan, serta objek pengembangan (model proses atau objek proses).

Meskipun berbeda, pada dasarnya setiap metodologi pengembangan sistem berusaha mencapai suatu tujuan yang sama yaitu harapan untuk mendefinisikan secara akurat apa yang menjadi kebutuhan pengguna, memantau dan mengendalikan proyek pengembangan sistem dengan baik, serta menghasilkan sistem dengan sumber daya pengembangan yang efisien.

Tes Formatif 2

---

Pilihlah satu jawaban yang paling tepat!

- 1) Secara umum urutan fase yang tepat pada siklus hidup pengembangan sistem adalah ....
  - A. Analisis, Perencanaan, Desain, Pemeliharaan, dan Implementasi
  - B. Perencanaan, Analisis, Desain, Pemeliharaan, Implementasi
  - C. Perencanaan, Analisis, Desain, Implementasi, Pemeliharaan
  - D. Analisis, Desain, Implementasi, Perencanaan, dan Pemeliharaan
- 2) Studi yang pada intinya mencoba meninjau apakah kebutuhan akan pengembangan sistem informasi baru layak secara ekonomis dan layak dari sisi kelayakan lainnya, termasuk fase .... dalam pengembangan sistem.
  - A. Perencanaan Sistem
  - B. Analisis dan Desain Sistem
  - C. Implementasi Sistem
  - D. Pemeliharaan Sistem
- 3) Beberapa kelebihan *System Development Life Cycle* (SDLC) sebagai metode pengembangan sistem, *kecuali* ....
  - A. dokumentasi standar yang mempermudah *stakeholder* (pemangku kepentingan) berkomunikasi antar satu dengan yang lainnya
  - B. terdapat fase-fase dengan batasan dan hasil kegiatan yang jelas, sehingga mempermudah pengelolaan proyek
  - C. fleksibel, yaitu jika terjadi perubahan proses bisnis maka desain keluaran dan desain lainnya seketika dapat langsung diubah
  - D. jadwal setiap proses dapat ditentukan secara pasti, sehingga dapat dilihat jelas target penyelesaian pengembangan sistem.
- 4) Yang termasuk metodologi pengembangan sistem melalui pendekatan Siklus Hidup Sistem adalah ....
  - A. Waterfall
  - B. Paket
  - C. End-User Development
  - D. Jawaban A, B, dan C benar

- 5) Penerapan metodologi pengembangan sistem secara formal akan berdampak pada....
- mendapat penolakan dari para pemrogram yang lebih senang mengandalkan pengalaman pribadi atau best practice dalam mengembangkan sistem
  - mendorong terciptanya solusi yang terintegrasi bagi manajemen organisasi yang besar dan kompleks
  - menyebabkan proses pengembangan proyek sistem menjadi tidak fleksibel.
  - semua jawaban A, B dan C salah
- 6) Bila dalam fase Desain Sistem terdapat masalah yang diakibatkan oleh adanya kebutuhan sistem yang tidak terpenuhi, maka proyek sistem perlu ditinjau kembali dengan cara mengimplementasikan kembali tahap Analisis Sistem sebelum projek dilanjutkan pada fase Implementasi. Prinsip ini adalah salah satu prinsip yang diterapkan dalam SDLC.
- Benar
  - Salah
- 7) Pada umumnya sistem yang dibangun menggunakan SDLC adalah sistem pemrosesan transaksi yang diperuntukkan memenuhi kebutuhan level manajerial operasional di bagian paling bawah.
- Benar
  - Salah
- 8) Dalam pengukuran kualitas sebuah sistem informasi berbasis komputer (perangkat lunak), pengukuran kualitas sistem dilakukan terhadap kualitas produk jadinya, sehingga kematangan proses pengembangannya menjadi hal yang dapat dihiraukan.
- Benar
  - Salah
- 9) Beragam metodologi alternatif yang dapat digunakan dalam pengembangan sistem informasi, antara lain *end-user development*, *extreme programming*, *outsourcing*, dan spiral.
- Benar
  - Salah
- 10) Penggunaan metodologi pengembangan sistem secara formal bukanlah suatu jaminan bahwa suatu proyek pengembangan sistem sepenuhnya akan dapat mencapai tujuan.
- Benar
  - Salah

Cocokkanlah jawaban Anda dengan Kunci Jawaban Tes Formatif 2 yang terdapat di bagian akhir modul ini. Hitunglah jawaban yang benar. Kemudian, gunakan rumus berikut untuk mengetahui tingkat penguasaan Anda terhadap materi Kegiatan Belajar 2.

$$\text{Tingkat Penguasaan} = \frac{\text{Jumlah Jawaban yang Benar}}{\text{Jumlah Soal}} \times 100$$

Arti tingkat penguasaan

<70%	70% - 79%	80% - 89%	90% - 100%
kurang	cukup	baik	baik sekali

Apabila mencapai tingkat penguasaan 80% atau lebih, Anda dapat meneruskan dengan modul selanjutnya. **Bagus!** Jika masih di bawah 80%, Anda harus mengulangi materi Kegiatan Belajar 2, terutama bagian yang belum dikuasai.

## Kunci Jawaban Tes Formatif

*Tes Formatif 1*

- 1) D
- 2) A
- 3) B
- 4) D
- 5) C
- 6) B
- 7) B
- 8) B
- 9) B
- 10) A

*Tes Formatif 2*

- 1) C
- 2) A
- 3) C
- 4) A
- 5) B
- 6) B
- 7) A
- 8) B
- 9) B
- 10) A

## Glosarium

Analis	: Individu yang diberi tugas untuk menganalisis permasalahan dan mendesain kebutuhan pada berbagai aspek (bisnis, sistem, infrastruktur, dan manajemen perubahan, dan lain-lain) dalam siklus pengembangan sistem informasi
Analisis sistem	: Kegiatan mengidentifikasi permasalahan dan kebutuhan sistem informasi
Desain sistem	: Kegiatan menerjemahkan model-model logis sistem menjadi model-model fisik sistem informasi
Fase	: Tahapan-tahapan dalam pengembangan sistem informasi
Implementasi sistem	: Kegiatan menulis program komputer dengan berpedoman pada desain sistem, menguji program komputer yang telah ditulis, instalasi, dan integrasi program komputer dengan perangkat keras pendukung sistem informasi, serta menerapkan sistem pada situasi yang sebenarnya
Metodologi	: Kumpulan prosedur, teknik, alat, dan alat bantu pendokumentasian yang membantu para pengembang dalam mengembangkan sistem informasi
Model fisik	: Gambar teknis yang merepresentasikan fungsi-fungsi sistem informasi
Model logis	: Gambar non teknis yang merepresentasikan prosedur bisnis organisasi
Pemodelan sistem	: Kegiatan menerjemahkan prosedur bisnis organisasi menjadi model-model logis sistem informasi
Pengembangan sistem	: Kegiatan menyusun suatu sistem informasi yang baru untuk menggantikan sistem yang lama secara keseluruhan atau memperbaiki sistem yang telah ada
Perangkat lunak (software)	: Program aplikasi sistem informasi berbasis komputer

- Perawatan sistem : Kegiatan menjaga, memperbaiki, atau menanggulangi permasalahan yang muncul ketika sistem informasi sedang diimplementasikan
- Programmer (pemrogram) : Individu yang diberi tugas untuk menerjemahkan desain sistem informasi menjadi program aplikasi sistem informasi
- Proses bisnis : Rangkaian aktivitas atau kegiatan tertentu yang terstruktur yang dilakukan dalam organisasi, yang menjadi acuan dalam pengembangan sistem informasi
- Sistem berbasis desktop : Aplikasi sistem informasi berbasis komputer yang berjalan di atas komputer yang berdiri sendiri (tidak terkoneksi ke jaringan komputer)
- Sistem informasi berbasis komputer : Sistem informasi yang menggunakan komputer sebagai alat pengolah data dan penyaji informasi
- Sistem informasi : Sistem berbasis komputer yang mengolah data dan menghasilkan informasi yang berguna bagi manajemen organisasi
- Sistem lama : Sistem informasi yang telah digunakan sebelum sistem baru dikembangkan, baik berupa sistem informasi yang sudah terkomputerisasi maupun sistem yang masih manual
- Sistem manual : Sistem pengolahan, pemrosesan transaksi, dan penyajian informasi yang belum menggunakan komputer sebagai alat bantu
- Sistem terdistribusi : Aplikasi sistem informasi berbasis komputer yang basis datanya berbeda di beberapa tempat, namun saling terhubung melalui jaringan komputer
- Stakeholder : Semua pihak yang memiliki kepentingan atau terkena dampak pengembangan sistem informasi
- System Development Life Cycle : Sebuah siklus dalam proses pengembangan sistem informasi, yang terdiri dari tahapan merencanakan membuat, menguji, dan meluncurkan sistem informasi

## Daftar Pustaka

- Avison, D. E., & Fitzgerald, G. (2006). *Information system development: Methodologies, techniques and tools* (3rd edition). London: McHraw-Hill.
- Beynon, D. P., & Williams, M. D. (2003). The diffusion of information systems development methods. *The journal of strategic information systems*, 12(1), 29-46.
- Dennis, A., Wixom, B. H., & Roth, R. M. (2012). *Systems analysis & design* (fifth edition). United States of America: John Wiley & Sons, Inc.
- Dennis, A., Wixom, B. H., & Tegarden, D. (2009). *Systems analysis & design with UML Version 2.0* (3rd edition). United States of America: John Wiley & Sons, Inc.
- Hardiansyah, F. (2019). E-KEUANGAN: Menuju *good governance*. *Jurnal Sains Penelitian & Pengabdian*, 1(2): 17-24.
- Hoffer, J. A., George, J. F., & Valacich, J. S. (2011). *Modern system analysis and design* (6th edition). Boston: Pearson.
- IEEE Standard Glossary of Software Engineering Technology. (1990). *IEEE Std 610.12-1990*. New York: Institute of Electrical and Electronics Engineers.
- Iivari, J., Hirschheim, R., & Klein, H. K. (2000). A dynamic framework for classifying information systems development methodologies and approaches. *Journal of management information systems*, 17(3), 179-218.
- Jogiyanto, H. M. (2008). *Analisis & desain sistem informasi: Pendekatan terstruktur teori dan praktik aplikasi bisnis* (edisi 3). Yogyakarta: Penerbit ANDI.
- Jogiyanto, H. M. (2010). *Sistem teknologi informasi* (edisi 3). Yogyakarta: Penerbit ANDI.
- Kautz, K. (2004). The enactment of methodology: The case of developing a multimedia information system. *ICIS 2004 Proceedings*, 55.

Prasetyo, B. (2010). Kajian tentang metodologi pengembangan sistem informasi. *Risalah Lokakarya Komputasi dalam Sains dan Teknologi Nuklir*, 97-122.

Radliya, N. R., Fauzan, R., & Irmayanti, H. (2018). Pengembangan sistem informasi geografis dalam manajemen tata ruang wilayah. *Jurnal Teknologi dan Informasi*, 8(2): 73-81.

Romney, M. B., & Steinbart, P. J. (2012). *Accounting information system* (12th edition). Boston: Pearson.

Sarosa, A. (2017). *Metodologi pengembangan sistem informasi*. Jakarta: Penerbit Indeks.

Sulianta, F. (2017). *Teknik perancangan arsitektur sistem informasi*. Yogyakarta: Penerbit ANDI.

Whitten, J. L., Bentley, L. D., & Dittman, K. C. (2001). *System analysis and design methods* (5th edition). Boston: McGraw-Hill.

Zaenudin, A. (2017). *Akhir perjalanan dari kegagalan Windows Vista*. Diakses dari <https://tirto.id/akhir-perjalanan-dari-kegagalan-windows-vista-clqH>: 4 April 2020.

**MSIM4302**  
**Edisi 1**

**MODUL 02**  
**Perencanaan Sistem**

Bahar, S.T., M.Kom.

## Daftar Isi

Modul 02	
2.1	
Perencanaan Sistem	
<b>Kegiatan Belajar 1</b>	2.4
Mendefinisikan Proyek Sistem	
Latihan	2.20
Rangkuman	2.22
Tes Formatif 1	2.23
<b>Kegiatan Belajar 2</b>	2.26
Membuat Rencana Proyek	
Latihan	2.43
Rangkuman	2.46
Tes Formatif 2	2.47
Kunci Jawaban Tes Formatif	2.50
Glosarium	2.51
Daftar Pustaka	2.54



## Pendahuluan

Proses manajemen proyek sistem informasi dimulai dengan serangkaian aktivitas yang secara kolektif disebut dengan Perencanaan Sistem. Aktivitas yang pertama dilakukan adalah estimasi atau perkiraan. Tujuannya adalah melihat ke masa depan mengenai ketidakpastian. Meskipun estimasi merupakan suatu seni seperti juga pada sains, aktivitas ini tidak dilakukan secara serampangan. Terdapat teknik-teknik yang berguna untuk mengestimasi waktu dan pelaksanaan kegiatan.

Pembahasan pada modul ini diasumsikan bahwa manajeman puncak telah menetapkan kebijakan untuk mengembangkan sistem informasi, dan perencanaan awal atas proyek-proyek sistem telah dilakukan oleh staf perencana sistem. Dengan demikian, perencanaan sistem yang dimaksudkan dalam pembahasan ini adalah perencanaan sistem yang dilakukan oleh tim analis sebelum fase analisis kebutuhan (*requirement analysis*) dilaksanakan.

Setelah mempelajari modul ini mahasiswa diharapkan secara lebih rinci mampu menjelaskan:

1. cara melakukan studi kelayakan sistem;
2. jenis-jenis dan aspek-aspek yang perlu diperhatikan dalam studi kelayakan sistem;
3. opsi-opsi metodologi pengembangan sistem yang dapat dipilih dalam membuat rencana proyek sistem dan karakteristik setiap metodologi yang ada;
4. berbagai kriteria penting yang menjadi pertimbangan dalam memilih metodologi pengembangan sistem;
5. cara memperkirakan waktu yang dibutuhkan dalam pengembangan sistem;
6. cara mengembangkan rencana kerja proyek.

## Mendefinisikan Proyek Sistem

Tujuan perencanaan proyek sistem informasi adalah untuk menyediakan sebuah kerangka kerja yang memungkinkan manajer proyek membuat estimasi yang dapat dipertanggungjawabkan mengenai sumber daya, biaya, dan jadwal. Estimasi dibuat dengan sebuah kerangka waktu yang terbatas pada awal sebuah proyek sistem informasi dan diperbarui secara teratur selagi proyek sedang berjalan. Estimasi akan berusaha mendefinisikan skenario kasus terbaik dan kasus terburuk (Pressman, 2014).

Setelah suatu proyek jangka panjang disetujui oleh manajemen dan sistem prioritas telah ditentukan oleh komite pengarah, Dennis (2012) menyebutnya sebagai komite persetujuan, serta tim analis telah ditunjuk, maka tahap perencanaan sistem berikutnya adalah mendefinisikan proyek-proyek sistem prioritas yang akan dikembangkan ini. Mendefinisikan proyek-proyek sistem ini berarti melakukan suatu studi untuk mencari alternatif-alternatif pemecahan terbaik yang paling layak untuk dikembangkan. Studi ini dilakukan oleh tim analis (Jogiyanto, 2008).

### A. MENDEFINISIKAN KEMBALI RUANG LINGKUP DAN SASARAN PROYEK

Ruang lingkup dan sasaran dari proyek-proyek sistem yang akan dikembangkan dapat diperoleh dari laporan perencanaan awal sistem yang telah dibuat dan disetujui oleh manajemen. Yang menentukan ruang lingkup dan sasaran proyek sistem ini adalah staf perencana sistem. Tim analis tidak terlibat dalam penentuan ruang lingkup dan sasaran ini, sehingga ruang lingkup dan sasaran proyek-proyek sistem ini perlu didefinisikan kembali oleh analis. Analis dapat berdiskusi terlebih dahulu dengan manajemen untuk mengkaji kembali mengenai ruang lingkup dan sasaran sistem ini, dan jika perlu memberikan usulan-usulan atau saran-saran tambahan bilamana ruang lingkup proyek kurang tepat dan sasaran proyek dapat ditingkatkan lagi. Ruang lingkup dan sasaran dari proyek sistem harus betul-betul dipahami terlebih dahulu oleh tim analis, karena hal ini sangat erat hubungannya dalam perencanaan biaya dan waktu pengembangan sistem yang akan diperkirakan oleh analis.

## B. MELAKUKAN STUDI KELAYAKAN

Menurut Tantra (2012), studi kelayakan adalah tindakan yang dilakukan untuk menentukan apakah suatu proyek layak untuk direalisasikan atau tidak. Proyek tidak layak jika hanya menghasilkan produk atau jasa yang hanya berfungsi satu kali saja, atau jika proyek yang akan dilaksanakan itu tidak memberi manfaat apapun selain hanya menambah panjang birokrasi.

Menurut Jogiyanto (2008), studi kelayakan adalah suatu studi yang akan digunakan untuk menentukan kemungkinan apakah pengembangan proyek sistem layak diteruskan atau dihentikan, sedangkan menurut Sommerville (2011) studi kelayakan adalah studi yang merekomendasikan apakah kegiatan yang telah direncanakan layak diteruskan pada rekayasa kebutuhan dan proses pengembangan sistem.

Berdasarkan beberapa definisi studi kelayakan yang dikemukakan oleh para ahli, dapat disimpulkan bahwa inti dari kegiatan studi kelayakan adalah menilai apakah proyek-proyek sistem yang akan dikembangkan dapat diteruskan ke fase pengembangan atau harus dihentikan.

Waktu yang digunakan untuk melaksanakan studi kelayakan tergantung dari ruang lingkup proyek yang akan dikembangkan. Untuk proyek sistem yang kecil dengan sedikit perubahan-perubahan, studi kelayakan mungkin hanya membutuhkan waktu beberapa hari, namun untuk proyek sistem yang menelan biaya besar dan akan diterapkan pada perusahaan besar, mungkin studi kelayakan akan memakan waktu berbulan-bulan. Demikian juga dengan biaya yang dikeluarkan untuk melakukan studi kelayakan tergantung pada ruang lingkup proyek, namun dapat diperkirakan sekitar 5 sampai 10 persen dari seluruh nilai proyek yang diperkirakan (Jogiyanto, 2008).

Studi kelayakan dalam tahap ini dilakukan oleh tim analis dengan melakukan penelitian pendahuluan. Penelitian pendahuluan merupakan penelitian yang dilakukan tidak terlalu rinci, yaitu hanya untuk mendapatkan jawaban apa yang dikerjakan oleh sistem yang lama dan apa yang dinginkan oleh sistem yang baru. Artinya, tim analis hanya mengumpulkan informasi kelayakan dari proyek yang akan dilaksanakan, dan bukan untuk menyusun *requirement* (kebutuhan) sistem. Penelitian yang terinci akan dilakukan di tahap analisis sistem.

Penelitian pendahuluan ini bertujuan untuk:

### 1. Memahami operasi dari sistem yang lama

Sistem yang lama merupakan sumber data untuk menyusun sistem yang baru. Dengan memahami sistem yang lama, analis akan mempunyai gambaran seperti apa sistem yang seharusnya akan dikembangkan. Cara yang paling baik dan menghemat waktu untuk mempelajari sistem yang lama adalah mendapatkan dokumen-dokumen sistem yang telah ada di perusahaan, misalnya:

- a. bagan alir dokumen (*paperwork flowchart*),
- b. bagan alir sistem (*system flowchart*),

- c. struktur organisasi,
  - d. deskripsi jabatan,
  - e. salinan dokumen-dokumen input /dasar,
  - f. salinan laporan-laporan.
- 2. Menentukan kebutuhan-kebutuhan pengguna sistem secara garis besar untuk dapat mencapai sasaran sistem**

Setelah sistem yang lama dipahami, yaitu apa yang dikerjakan oleh sistem yang lama, siapa yang mengoperasikan dan yang memakainya, analis kemudian dapat mengumpulkan data mengenai kebutuhan-kebutuhan pengguna sistem untuk mencapai sasaran yang akan direncanakan. Analis dapat menggunakan beberapa teknik pengumpulan data seperti wawancara, observasi, dan pengambilan sampel.

- 3. Menentukan permasalahan-permasalahan yang terjadi sehingga sistem yang lama belum dapat mencapai sasaran yang diinginkan**

Analis harus dapat menemukan jawaban mengenai apa yang menyebabkan sasaran sistem yang direncanakan saat ini belum dapat dicapai oleh sistem yang lama.

Hasil penelitian pendahuluan ditujukan untuk menjawab sejumlah pertanyaan berikut.

- a. Apakah sistem yang akan dikembangkan dapat memberikan kontribusi bagi tujuan organisasi secara keseluruhan?
- b. Apakah sistem dapat diimplementasikan dengan menggunakan teknologi terbaru dan dalam batasan biaya dan jadwal?
- c. Apakah sistem dapat diintegrasikan dengan sistem lain yang sudah ada?

Untuk dapat menjawab tiga pertanyaan tersebut, analis juga melakukan pengumpulan dan penilaian informasi dengan menanyai sumber informasi untuk mendapatkan jawaban atas pertanyaan-pertanyaan ini. Sumber informasi bisa melibatkan manajer departemen di mana sistem akan dipakai, perekayasa sistem yang mengenal sistem yang diajukan, pakar teknologi, dan *end-user* sistem. Beberapa contoh pertanyaan yang mungkin diberikan adalah sebagai berikut.

- a. Bagaimana organisasi mengatasi masalah jika sistem ini tidak diimplementasikan?
- b. Apa masalah dengan proses pada saat ini dan bagaimana sistem yang baru bisa membantu meringankan masalah ini?
- c. Apa kontribusi langsung yang akan diberikan sistem bagi tujuan bisnis?
- d. Dapatkan informasi ditransfer ke dan dari sistem organisasi?
- e. Apakah sistem membutuhkan teknologi yang sebelumnya tidak dipakai pada organisasi?
- f. Apa yang harus didukung oleh sistem dan apa yang tidak perlu?

Ketika informasi telah tersedia, disiapkan laporan studi kelayakan. Laporan ini harus memberikan rekomendasi mengenai apakah pengembangan sistem harus diteruskan atau harus dihentikan. Jika hasil kajian merekomendasikan sistem dapat diteruskan, tahapan selanjutnya yang dilakukan adalah Membuat Rencana Proyek.

### C. MENILAI KELAYAKAN PROYEK SISTEM

Analisis kelayakan mengidentifikasi risiko penting yang terkait dengan proyek yang harus dikelola setelah proyek dinyatakan disetujui. Seperti permintaan sistem, setiap organisasi memiliki proses dan format sendiri untuk analisis kelayakan, namun sebagian besar mencakup teknik untuk menilai tiga bidang yaitu: kelayakan teknis, kelayakan ekonomi, dan kelayakan organisasi; dan yang lain memasukkan kelayakan operasi, kelayakan jadwal, dan kelayakan hukum. Hasil dari evaluasi kelayakan ini digabungkan ke dalam dokumen studi kelayakan untuk disampaikan kepada komite persetujuan pada akhir inisiasi proyek.

<b>Kelayakan Teknis: Dapatkah kita membangunnya?</b>
<ul style="list-style-type: none"> <li>- Familiar (Keakraban) dengan aplikasi: Kurang familiar menghasilkan lebih banyak risiko.</li> <li>- Familiar (Keakraban) dengan teknologi: Kurang familiar menyebabkan lebih banyak risiko.</li> <li>- Ukuran proyek: Proyek besar memiliki risiko lebih besar.</li> <li>- Kesesuaian: Semakin sulit mengintegrasikan sistem dengan teknologi yang ada di perusahaan, semakin tinggi risikonya.</li> </ul>
<b>Kelayakan Ekonomi: Haruskah kita membangunnya?</b>
<ul style="list-style-type: none"> <li>- Biaya pengembangan</li> <li>- Biaya operasi tahunan</li> <li>- Manfaat tahunan (penghematan biaya dan / atau peningkatan pendapatan)</li> <li>- Manfaat dan biaya tidak berwujud</li> </ul>
<b>Kelayakan Organisasi: Jika kita membangunnya, akankah mereka menyambutnya?</b>
<ul style="list-style-type: none"> <li>- Penggagas proyek</li> <li>- Manajemen senior</li> <li>- Pengguna</li> <li>- Pemangku kepentingan lainnya</li> <li>- Apakah proyek secara strategis selaras dengan bisnis?</li> </ul>

**Gambar 2.1**  
**Faktor Penilaian Kelayakan**

Meskipun pembahasan analisis kelayakan dilakukan dalam konteks inisiasi proyek, sebagian besar tim proyek akan merevisi studi kelayakan mereka di sepanjang fase SDLC dan meninjau kembali isinya di berbagai pos pemeriksaan selama proyek. Jika pada suatu saat risiko dan kerugian proyek lebih besar daripada manfaatnya, tim proyek dapat memutuskan untuk membatalkan proyek atau membuat revisi besar.

### 1. Kelayakan Teknis

Teknik pertama dalam analisis kelayakan adalah menilai kelayakan teknis proyek, sejauh mana sistem berhasil dirancang, dikembangkan, dan diinstalasi oleh personel Teknologi Informasi. Analisis kelayakan teknis, pada dasarnya, adalah analisis risiko teknis yang berupaya menjawab pertanyaan: "Dapatkan kita membangunnya?".

Banyak risiko yang dapat mengancam keberhasilan penyelesaian proyek. Pertama dan terpenting adalah keakraban pengguna dan analis dengan sistem yang akan dikembangkan. Ketika analis tidak familiar pada area aplikasi bisnis, mereka berpeluang untuk salah memahami kebutuhan pengguna atau kehilangan peluang untuk melakukan perbaikan sistem. Risiko meningkat secara dramatis ketika pengguna sendiri kurang terbiasa dengan sistem aplikasi, seperti pengembangan sistem untuk mendukung inovasi bisnis baru. Secara umum, pengembangan sistem baru lebih berisiko daripada jika tetap melanjutkan menggunakan sistem yang ada, karena sistem yang ada cenderung lebih dipahami.

Keakraban dengan teknologi merupakan sumber risiko teknis penting lainnya. Ketika suatu sistem akan menggunakan teknologi yang belum pernah digunakan sebelumnya dalam organisasi, kemungkinan besar bahwa masalah dan keterlambatan akan terjadi disebabkan karena kebutuhan untuk mempelajari bagaimana menggunakan teknologi tersebut. Risiko meningkat secara dramatis ketika teknologi itu sendiri baru (misalnya teknologi pengembangan web menggunakan Ajax).

Ukuran proyek merupakan pertimbangan penting, apakah diukur sebagai jumlah orang dalam tim pengembangan, lamanya waktu yang dibutuhkan untuk menyelesaikan proyek, atau jumlah fitur berbeda yang akan disediakan dalam sistem yang dikembangkan. Proyek yang lebih besar menghadirkan kemungkinan lebih banyak risiko, karena lebih rumit untuk dikelola dan berpeluang terabaikannya beberapa kebutuhan sistem penting atau disalahpahami. Sejauh mana sebuah proyek yang terintegrasi dengan sistem lain dapat menyebabkan masalah, sebagai akibat dari kompleksitas ketika banyak sistem harus bekerja secara terintegrasi.

Akhirnya, tim proyek perlu mempertimbangkan kompatibilitas sistem baru dengan teknologi yang sudah ada di organisasi. Sistem jarang dibangun dalam kondisi berdiri sendiri. Sistem pada umumnya dibangun dalam organisasi yang sudah memiliki banyak sistem. Teknologi dan aplikasi baru harus dapat diintegrasikan dengan lingkungan yang ada karena berbagai alasan. Sistem baru mungkin mengandalkan data dari sistem yang ada. Sistem baru dapat menghasilkan data untuk memberi makna yang lebih sempurna kepada aplikasi lainnya, dan sistem baru mungkin harus menggunakan infrastruktur komunikasi yang sudah ada. Misalnya sistem CRM (*Customer Relationship Management*) baru, akan memiliki nilai kecil jika tidak menggunakan data pelanggan yang tersedia di seluruh organisasi dalam sistem penjualan, aplikasi pemasaran, dan sistem layanan pelanggan yang ada.

Penilaian kelayakan teknis proyek selayaknya tidak dipotong atau dikurangi, karena dalam banyak kasus, beberapa interpretasi terhadap situasi mendasar diperlukan

(misalnya, seberapa besar pertumbuhan sebuah proyek sebelum menjadi kurang layak). Salah satu pendekatan adalah organisasi membandingkan proyek yang sedang dipertimbangkan dengan proyek sebelumnya yang dilakukan oleh organisasi. Pilihan lain adalah berkonsultasi dengan profesional Teknologi Informasi yang berpengalaman dalam organisasi atau dengan konsultan Teknologi Informasi eksternal. Seringkali konsultan Teknologi Informasi eksternal dapat menilai apakah suatu proyek layak dari segi teknis.

## 2. Kelayakan Ekonomi

Elemen kedua dari analisis kelayakan adalah melakukan analisis kelayakan ekonomi (biasa disebut analisis biaya-manfaat). Analisis kelayakan ini mencoba menjawab pertanyaan "haruskah kita membangun sistem?". Kelayakan ekonomi ditentukan dengan mengidentifikasi biaya dan manfaat yang terkait dengan sistem, menentukan nilai proyeknya, menghitung arus kas masa depan, dan mengukur kelayakan finansial proyek. Hasil yang diharapkan dari analisis ini adalah peluang keuangan dan risiko proyek dapat dipahami. Perlu diingat bahwa organisasi memiliki sumber daya modal yang terbatas dan beberapa proyek akan bersaing untuk mendapatkan pendanaan. Semakin mahal proyeknya, analisisnya harus lebih teliti dan terperinci. Berikut ini adalah beberapa kerangka kerja yang dapat diterapkan untuk mengevaluasi investasi proyek dan langkah-langkah penilaian umum yang digunakan.

### a. Analisis dan Pengukuran Arus Kas (*Cash Flow Analysis and Measures*)

Proyek-proyek Teknologi Informasi biasanya melibatkan investasi awal yang menghasilkan manfaat dari waktu ke waktu, bersamaan dengan dukungan biaya yang berkelanjutan. Oleh karena itu, nilai proyek harus diukur dari waktu ke waktu. Arus kas, baik arus masuk maupun keluar, diperkirakan selama beberapa periode mendatang. Selanjutnya, arus kas dievaluasi menggunakan beberapa teknik untuk menilai apakah manfaat yang diproyeksikan akan sesuai dengan biaya yang akan dikeluarkan.

*Contoh*

	Tahun 0	Tahun 1	Tahun 2	Tahun 3	Total
Total Manfaat		45.000	50.000	57.000	152.000
Total Biaya	100.000	10.000	12.000	16.000	138.000
Keuntungan Bersih (Total Manfaat - Total Biaya)	(100.000)	35.000	38.000	41.000	14.000
Akumulasi Arus Kas Bersih	(100.000)	(65.000)	(27.000)	14.000	

Gambar 2.2  
Proyeksi Arus Kas Sederhana

Dalam contoh sederhana pada Gambar 2.2, sebuah sistem dikembangkan pada Tahun 0 (tahun berjalan) seharga \$ 100.000. Setelah sistem beroperasi, manfaat dan biaya yang berkelanjutan diproyeksikan selama tiga tahun. Pada baris ke-3, keuntungan bersih dihitung dengan mengurangi total biaya setiap tahun dari total manfaatnya. Pada baris 4, dihitung total kumulatif dari arus kas bersih.

Beberapa metode umum untuk mengevaluasi nilai proyek sekarang dapat dijelaskan berikut ini:

1) *Return on Investment*

*Return on Investment* (ROI) adalah perhitungan yang mengukur tingkat pengembalian rata-rata yang diperoleh dari uang yang diinvestasikan dalam proyek. ROI adalah perhitungan sederhana yang membagi manfaat bersih proyek (total manfaat - total biaya) dengan total biaya. Rumus ROI adalah (Dennis, 2012):

$$\text{ROI} = \frac{\text{Total Manfaat} - \text{Total Biaya}}{\text{Total Biaya}}$$

*Contoh*

Berdasarkan kasus pada Gambar 2.2, dapat dihitung nilai ROI sebagai berikut:

$$\text{ROI} = \frac{152.000 - 138.000}{138.000} = \frac{14.000}{138.000} = 10,14\%$$

ROI yang tinggi menunjukkan bahwa manfaat proyek jauh lebih besar daripada biaya proyek, meskipun definisi dari ROI "tinggi" itu tidak jelas. ROI biasanya digunakan dalam praktik, namun sulit untuk menafsirkan dan tidak boleh digunakan sebagai satu-satunya ukuran dari nilai proyek.

2) *Break-Even Point*

Pendekatan umum lainnya yang biasa digunakan untuk mengukur nilai proyek adalah titik impas (*Break-Even Point/BEP*). BEP (juga disebut metode pengembalian) didefinisikan sebagai jumlah tahun yang dibutuhkan suatu perusahaan untuk mengembalikan investasi awalnya. Seperti yang ditunjukkan pada baris 4 dari Gambar 2.2, arus kas kumulatif proyek menjadi positif pada tahun 3, sehingga investasi awal "kembali" selama dua tahun ditambah sebagian kecil dari tahun ketiga. Rumus BEP adalah (Dennis, 2012):

$$\text{BEP} = \text{JTAKN} + \frac{\text{AKB} - \text{AKK}}{\text{AKB}}$$

Keterangan:

JTAKN : Jumlah Tahun Arus Kas Negatif (*Number of years of negative cash flow*)

AKB : Arus Kas Bersih tahun berjalan (*That year's Net Cash Flow*)

AKK : Arus Kas Kumulatif tahun berjalan (*That year's Cumulative Cash Flow*)

**Contoh**

Berdasarkan kasus pada Gambar 2.2, dapat dihitung nilai BEP sebagai berikut:

$$\text{BEP} = 2 + \frac{41.000 - 14.000}{41.000} = 2 + \frac{27.000}{41.000} = 2,65 \text{ tahun}$$

BEP secara intuitif mudah dipahami dan memberikan indikasi likuiditas proyek, atau kecepatan proyek dapat mengembalikan modal investasi. Proyek-proyek yang menghasilkan pengembalian modal yang lebih tinggi di awal pengoperasian proyek dianggap kurang berisiko, karena dapat mengantisipasi peristiwa jangka pendek dengan lebih akurat daripada yang diperoleh dalam jangka waktu yang panjang. BEP mengabaikan arus kas yang terjadi setelah BEP telah tercapai, sehingga hal ini dapat bias pada proyek jangka panjang.

### 3) Discounted Cash Flow Technique

Proyeksi arus kas sederhana yang ditunjukkan pada Gambar 2.2, dan laba atas investasi serta perhitungan titik impas semuanya memiliki kelemahan yang sama yaitu tidak mengenali nilai waktu dari uang. Dalam analisis tersebut, waktu arus kas diabaikan. Misalnya nilai satu dolar pada tahun ke-3 proyek dianggap persis sama dengan satu dolar yang diterima pada tahun ke-1.

*Discounted cash flows* digunakan untuk membandingkan nilai sekarang (*Present Value/PV*) dari semua arus kas masuk dan keluar untuk proyek dalam suatu satuan nilai uang (misal dolar) hari ini. Kunci untuk memahami nilai saat ini adalah mengenali bahwa jika misalnya hari ini memiliki satu dolar, orang dapat menginvestasikannya dan menerima tingkat pengembalian atas investasi tersebut. Oleh karena itu, satu dolar yang diterima di masa depan bernilai kurang dari satu dolar yang diterima hari ini, karena orang melupakan potensi pengembalian. Misalnya, jika anda memiliki teman yang berutang \$ 100 hari ini, dan mengembalikan sejumlah \$ 100 dalam tiga tahun. Dengan asumsi Anda bisa menginvestasikan dolar itu pada tingkat pengembalian 10%, Anda akan menerima yang setara dengan \$ 75 dalam nilai hari ini. Rumus dasar untuk mengkonversi arus kas masa depan ke nilai saat ini adalah:

$$PV = \frac{\text{Jumlah Arus Kas}}{(1 + \text{Tingkat Pengembalian})^n}$$

Di mana n adalah = Tahun saat Arus Kas terjadi

Tingkat pengembalian yang digunakan dalam perhitungan nilai sekarang kadang-kadang disebut tingkat pengembalian yang disyaratkan, atau biaya untuk mendapatkan modal yang diperlukan untuk mendanai proyek. Banyak organisasi akan menentukan tingkat pengembalian yang tepat untuk digunakan ketika menganalisis investasi di bidang Teknologi Informasi. Analis sistem harus berkonsultasi dengan departemen keuangan organisasi.

*Contoh*

Jika \$ 100 yang diterima dalam 3 tahun dengan tingkat pengembalian yang diperlukan sebesar 10%, akan memiliki PV sebesar \$ 75,13.

$$PV = \frac{100}{(1 + 0,10)^3} = \frac{100}{1,331} = 75,13$$

Gambar 2.3 merupakan hasil perhitungan nilai sekarang dari manfaat yang diproyeksikan dan biaya yang ditunjukkan pada Gambar 2.2 (menggunakan tingkat pengembalian yang dipersyaratkan sebesar 10%).

	Tahun 0	Tahun 1	Tahun 2	Tahun 3	Total
Total Manfaat		45.000	50.000	57.000	
PV Total Manfaat		40.909	41.322	42.825	125.056
Total Biaya	100.000	10.000	12.000	16.000	
PV Total Biaya	100.000	9.091	9.917	12.021	131.029

**Gambar 2.3**  
Hasil Perhitungan Proyeksi Arus Kas Sederhana

4) *Net Present Value (NPV)*

NPV secara sederhana dinyatakan sebagai perbedaan antara total nilai sekarang dari manfaat dengan total nilai sekarang dari biaya.

$$NPV = \sum PV \text{ Total Manfaat} - \sum PV \text{ Total Biaya}$$

$$NPV = \$ 125.056 - \$ 131.029 = (\$ 5.973)$$

Selama NPV lebih besar dari nol, proyek dianggap dapat diterima secara ekonomi. Dalam contoh di atas NPV kurang dari nol, menunjukkan bahwa untuk tingkat pengembalian yang diperlukan sebesar 10%, proyek ini tidak layak diterima. Tingkat pengembalian yang disyaratkan harus kurang dari 6,65% sebelum proyek ini mengembalikan NPV positif. Contoh ini menggambarkan fakta bahwa kadang-kadang teknik ROI dan BEP menemukan bahwa proyek tersebut tampak dapat diterima, akan tetapi teknik NPV yang lebih teliti secara finansial menemukan proyek tersebut sebenarnya tidak dapat diterima.

Gambar 2.4 mengulas langkah-langkah dalam melakukan analisis kelayakan secara ekonomi (Dennis, 2012).

1. Mengidentifikasi biaya dan manfaat	Buatlah daftar biaya dan manfaat nyata untuk proyek. Termasuk biaya satu kali penggunaan dan yang berulang.
2. Menentukan nilai biaya dan manfaat	Berkolaborasi dengan pengguna bisnis dan profesional Teknologi Informasi dalam menetukan angka untuk setiap biaya dan manfaat. Bahkan benda tak berwujud harus dinilai jika memungkinkan.
3. Menetapkan Arus Kas	Perkirakan berapa biaya dan manfaatnya selama periode tertentu, biasanya, tiga hingga lima tahun.
4. Menilai Nilai Ekonomi Proyek	<p>Evaluasi pengembalian yang diharapkan proyek dibandingkan dengan biayanya. Gunakan satu atau lebih teknik evaluasi berikut:</p> <ul style="list-style-type: none"> <li>- <i>Return on Investment (ROI)</i> Hitung tingkat pengembalian yang diperoleh dari uang yang diinvestasikan dalam proyek, menggunakan rumus ROI.</li> <li>- <i>Break-Even Point (BEP)</i> Temukan tahun di mana manfaat proyek kumulatif melebihi biaya proyek kumulatif. Terapkan formula BEP, menggunakan angka untuk tahun itu. Perhitungan ini mengukur berapa lama waktu yang dibutuhkan sistem untuk menghasilkan keuntungan yang menutupi biayanya.</li> <li>- <i>Net Present Value (NPV)</i> Nyatakan kembali semua biaya dan manfaat dalam ketentuan nilai mata uang hari ini (nilai sekarang), menggunakan tingkat diskonto yang sesuai. Tentukan apakah total nilai sekarang dari manfaat lebih besar atau kurang dari total nilai sekarang dari biaya</li> </ul>

Gambar 2.4  
Langkah-langkah dalam Melakukan Analisis Kelayakan Ekonomi

#### b. Identifikasi Biaya dan Manfaat

Tugas pertama analis sistem ketika mengembangkan analisis kelayakan ekonomi adalah mengidentifikasi jenis-jenis biaya dan manfaat yang akan dimiliki sistem dan mendaftarkannya di sepanjang kolom sebelah kiri *spreadsheet*. Gambar 2.5 mencantumkan contoh biaya dan manfaat yang mungkin dimasukkan. Biaya dan manfaat dapat dibagi menjadi empat kategori: (1) biaya pengembangan, (2) biaya operasional, (3) manfaat nyata, dan (4) manfaat tidak nyata. Biaya pengembangan adalah biaya nyata yang dikeluarkan selama pembuatan sistem, seperti gaji untuk tim proyek, biaya perangkat keras dan perangkat lunak, biaya konsultan, pelatihan, ruang dan peralatan kantor. Biaya pengembangan biasanya dianggap sebagai biaya satu kali.

Biaya operasional adalah biaya nyata yang diperlukan untuk mengoperasikan sistem, seperti gaji staf operasi, biaya lisensi perangkat lunak, peningkatan peralatan,

## 2.14 Perencanaan Sistem

dan biaya komunikasi. Biaya operasional biasanya dianggap sebagai biaya berkelanjutan.

Biaya Pengembangan	Biaya Operasional
Gaji Tim Pengembangan	Pembaruan Perangkat Lunak
Biaya Konsultan	Biaya Lisensi Perangkat Lunak
Pelatihan Pengembangan	Perbaikan Perangkat Keras
Perangkat Keras dan Perangkat Lunak	Pembaruan Perangkat Keras
Instalasi Vendor	Gaji Tim Operasional
Ruang dan Peralatan Kantor	Biaya Komunikasi
Biaya Konversi Data	Pelatihan Pengguna
Manfaat Nyata (Berwujud)	Manfaat Tidak Nyata
Peningkatan Penjualan	Peningkatan Pangsa Pasar
Pengurangan Staf	Pengurangan Pengakuan Merek
Pengurangan Inventaris	Produk Berkualitas Tinggi
Pengurangan Biaya TI	Peningkatan Layanan Pelanggan
Harga Pemasok yang Lebih Baik	Hubungan Pemasok yang Lebih Baik

**Gambar 2.5**  
Contoh Biaya dan Manfaat pada Kelayakan Ekonomi

Manfaat atau nilai nyata (*tangible benefits*) termasuk pendapatan yang dimungkinkan sistem organisasi terima, seperti peningkatan penjualan. Selain itu, sistem dapat memungkinkan organisasi untuk menghindari biaya tertentu, yang mengarah ke jenis lain dari manfaat nyata, seperti penghematan biaya. Misalnya, jika sistem menghasilkan pengurangan staf yang dibutuhkan, biaya gaji menjadi lebih rendah. Demikian pula, pengurangan tingkat persediaan yang diperlukan karena sistem baru menghasilkan biaya persediaan lebih rendah. Dalam contoh-contoh ini, pengurangan biaya adalah manfaat nyata dari sistem baru.

Tentu saja, sebuah proyek juga dapat memengaruhi jajaran bawah organisasi dengan mendapatkan manfaat (nilai) tidak berwujud (*intangible benefits*) atau menimbulkan biaya tak berwujud (*intangible costs*). Biaya dan manfaat tidak berwujud lebih sulit untuk dimasukkan ke dalam analisis kelayakan ekonomi karena penentuannya didasarkan pada intuisi dan kepercayaan daripada pada "angka nyata". Meskipun demikian, biaya ini harus terdaftar dalam *spreadsheet* bersama dengan barang-barang nyata (*tangible items*).

c. *Penentuan Nilai Biaya dan Manfaat (Assign Values to Costs and Benefits)*

Setelah jenis biaya dan manfaat diidentifikasi, analis perlu menetapkan nilai mata uang tertentu untuknya. Ini terlihat mustahil, bagaimana seseorang bisa menghitung biaya dan manfaat yang belum terjadi, dan bagaimana prediksi itu dapat realistik. Meskipun tugas ini sangat sulit, analis harus melakukan yang terbaik untuk menghasilkan angka yang masuk akal untuk semua biaya dan manfaat. Hanya dengan demikian komite pengesahan proyek sistem dapat membuat keputusan berdasarkan informasi yang ada tentang apakah akan melanjutkan proyek atau tidak.

Strategi yang paling efektif untuk memperkirakan biaya dan manfaat adalah mengandalkan orang-orang yang memiliki pemahaman terbaik tentang hal tersebut. Misalnya, biaya dan manfaat yang terkait dengan teknologi atau proyek itu sendiri dapat disediakan oleh kelompok TI perusahaan atau konsultan eksternal, dan pengguna bisnis dapat menentukan angka-angka yang terkait dengan bisnis (misalnya: proyeksi penjualan dan tingkat pesanan). Perusahaan juga dapat mempertimbangkan proyek masa lalu, laporan industri, dan informasi vendor, meskipun sumber-sumber ini mungkin akan sedikit kurang akurat. Kemungkinan, semua estimasi akan direvisi ketika proyek berlanjut.

Jika proses memprediksi nilai tertentu terkait biaya atau manfaat terbukti sulit dilakukan, mungkin berguna jika memperkirakan kisaran nilai untuk biaya atau manfaat, kemudian menetapkan estimasi kemungkinan (probabilitas) untuk setiap nilai. Dengan informasi ini, nilai yang diharapkan untuk biaya atau manfaat dapat dihitung.

Bagaimana dengan manfaat dan biaya tidak berwujud (*intangible benefits and costs*)? Manfaat dan biaya tak berwujud tersebut terkadang dapat diterima untuk dimasukkan dalam daftar keuntungan tidak berwujud (seperti peningkatan layanan pelanggan) tanpa menetapkan nilai mata uang. Pada kesempatan lain harus dibuat perkiraan mengenai berapa banyak manfaat tidak berwujud (*intangible benefits*) yang “bernilai”. Disarankan agar dilakukan perhitungan biaya atau keuntungan tidak berwujud jika memungkinkan. Jika tidak, bagaimana dapat mengetahuinya pada saat angka-angka itu tiba-tiba muncul.

Sebagai contoh, suatu sistem mengklaim untuk meningkatkan layanan pelanggan. Manfaat ini tidak berwujud, akan tetapi diasumsikan bahwa peningkatan layanan pelanggan akan mengurangi jumlah keluhan pelanggan sebesar 10% setiap tahun selama tiga tahun, dan menghabiskan biaya telepon dan operator telepon yang menangani panggilan keluhan sebesar \$ 200.000. Tiba-tiba muncul angka yang nyata untuk menetapkan tujuan dan mengukur manfaat yang semula tidak berwujud tersebut.

Contoh analisis biaya - manfaat (*cost-benefit analysis*) secara rinci ditunjukkan pada Gambar 2.6. Dalam contoh ini, manfaat bertambah karena proyek diharapkan untuk meningkatkan penjualan, mengurangi panggilan keluhan pelanggan, dan menurunkan biaya persediaan. Secara sederhana dapat digambarkan, semua biaya pengembangan diasumsikan terjadi pada tahun berjalan 2015, dan semua manfaat dan biaya operasional diasumsikan dimulai ketika sistem diimplementasikan pada awal

2016, dan berlanjut hingga 2019. Perhatikan bahwa manfaat layanan pelanggan tidak berwujud (*intangible*) telah dikuantifikasi, berdasarkan penurunan panggilan telepon keluhan pelanggan. Manfaat tidak berwujud (*intangible benefit*) untuk memberikan layanan yang ditawarkan pesaing saat ini tidak dikuantifikasi, akan tetapi terdaftar sehingga komite persetujuan akan mempertimbangkan manfaat tersebut ketika menilai kelayakan ekonomi sistem.

*d. Menentukan Arus Kas (Determine Cash Flow)*

Analisis formal biaya-manfaat biasanya berisi biaya dan manfaat selama beberapa tahun tertentu (biasanya tiga hingga lima tahun) untuk menunjukkan arus kas dari waktu ke waktu (lihat Gambar 2.2 dan 2.6). Misalnya, pada Gambar 2.6 mencantumkan jumlah yang sama untuk panggilan pengaduan pelanggan, biaya inventaris, perangkat keras, dan perangkat lunak selama empat tahun. Seringkali, jumlah ditambah dengan beberapa tingkat pertumbuhan untuk menyesuaikan dengan peningkatan inflasi atau bisnis, seperti yang ditunjukkan oleh kenaikan 6% yang ditambahkan ke angka penjualan dalam sampel yang ada. Demikian pula, biaya tenaga kerja diasumsikan meningkat pada tingkat 4% setiap tahun. Pada kolom terakhir ditambahkan total untuk menentukan manfaat dan biaya keseluruhan.

*Contoh*

	2015	2016	2017	2018	2019	TOTAL
<b>Manfaat</b>						
Peningkatan Penjualan	500,000	530,000	561,800	595,508	2,187,308	
Pengurangan Panggilan Kehuhan Pelanggan <sup>1)</sup>	70,000	70,000	70,000	70,000	280,000	
Pengurangan Biaya Persediaan	68,000	68,000	68,000	68,000	272,000	
<b>Total Manfaat<sup>2)</sup></b>	<b>638,000</b>	<b>668,000</b>	<b>699,800</b>	<b>733,508</b>	<b>2,739,308</b>	
<b>Biaya Pengembangan</b>						
2 Server @ %125,000	250,000	0	0	0	0	250,000
Printer	100,000	0	0	0	0	100,000
Lisensi Software	34,825	0	0	0	0	34,825
Software Server	10,945	0	0	0	0	10,945
Pengembangan Tenaga Kerja	1,236,525	0	0	0	0	1,236,525
<b>Total Biaya Pengembangan</b>	<b>1,632,295</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>1,632,295</b>
<b>Biaya Operasional</b>						
Hardware	50,000	50,000	50,000	50,000	200,000	
Software	20,000	20,000	20,000	20,000	80,000	
Operasional Tenaga Kerja	115,000	119,600	124,384	129,359	488,343	
<b>Total Biaya Operasional</b>	<b>185,000</b>	<b>189,600</b>	<b>194,384</b>	<b>199,359</b>	<b>768,343</b>	
<b>Total Biaya</b>	<b>1,632,295</b>	<b>185,000</b>	<b>189,600</b>	<b>194,384</b>	<b>199,359</b>	<b>2,400,638</b>
<b>Total Manfaat - Total Biaya</b>	<b>(1,632,295)</b>	<b>453,000</b>	<b>478,400</b>	<b>505,416</b>	<b>534,149</b>	<b>338,670</b>
<b>Kumulatif Arus Kas Bersih</b>	<b>(1,632,295)</b>	<b>(1,179,295)</b>	<b>(700,895)</b>	<b>(195,479)</b>	<b>338,670</b>	
<b>Return on Investment (ROI)</b>	<b>14.1%</b>	$(338,670 / 2,400,638)$				
<b>Break-even Point (BEP)</b>	<b>3.37 tahun</b>	$(3 \text{ tahun arus kas kumulatif negatif} + [534,149 - 338,670] / 534,149) = (3 + 0.37)$				

<sup>1)</sup> Nilai layanan Pelanggan didasarkan pada pengurangan biaya penanganan panggilan telepon keluhan pelanggan

<sup>2)</sup> Manfaat penting namun tidak berwujud adalah kemampuan untuk menawarkan layanan yang saat ini ditawarkan oleh pesaing

**Gambar 2.6**  
Arus Kas Sederhana untuk Analisis Biaya-Manfaat

e. *Nilai Ekonomi Proyek (Project's Economic Value)*

1) ROI dan BEP

Pada Gambar 2.6 hasil perhitungan ROI (*Return on Investment*) untuk contoh proyek tersebut mendapatkan angka 14,1%, sedangkan BEP (*Break-even Point*) selama 3,37 tahun.

2) NPV

Dalam Gambar 2.7, nilai sekarang dari biaya dan manfaat telah dihitung dan ditambahkan ke contoh tersebut, menggunakan tingkat pengembalian 6%. NPV (*Net Present Value*) hanyalah perbedaan antara total nilai sekarang dari manfaat dan total nilai sekarang dari biaya. Selama NPV lebih besar dari nol, proyek dianggap layak secara ekonomi. Dalam contoh ini, karena NPV adalah \$ 68.292, proyek harus dinyatakan diterima dari sudut pandang kelayakan ekonomi.

	2015	2016	2017	2018	2019	TOTAL
<b>Manfaat</b>						
Peningkatan Penjualan	500,000	530,000	561,800	595,508		
Pengurangan Panggilan Keluhan Pelanggan <sup>1)</sup>	70,000	70,000	70,000	70,000		
Pengurangan Biaya Persediaan	68,000	68,000	68,000	68,000		
<b>Total Manfaat<sup>2)</sup></b>	<b>638,000</b>	<b>668,000</b>	<b>699,800</b>	<b>733,508</b>		
<b>Nilai Sekarang Total Manfaat</b>	<b>601,887</b>	<b>594,518</b>	<b>587,566</b>	<b>581,007</b>	<b>2,364,978</b>	
<b>Biaya Pengembangan</b>						
2 Server @ %125,000	250,000	0	0	0	0	
Printer	100,000	0	0	0	0	
Lisensi Software	34,825	0	0	0	0	
Software Server	10,945	0	0	0	0	
Pengembangan Tenaga Kerja	1,236,525	0	0	0	0	
<b>Total Biaya Pengembangan</b>	<b>1,632,295</b>	<b>0</b>	<b>0</b>	<b>0</b>	<b>0</b>	
<b>Biaya Operasional</b>						
Hardware	50,000	50,000	50,000	50,000	50,000	
Software	20,000	20,000	20,000	20,000	20,000	
Operasional Tenaga Kerja	115,000	119,600	124,384	129,359		
<b>Total Biaya Operasional</b>	<b>185,000</b>	<b>189,600</b>	<b>194,384</b>	<b>199,359</b>		
<b>Total Biaya</b>	<b>1,632,295</b>	<b>185,000</b>	<b>189,600</b>	<b>194,384</b>	<b>199,359</b>	
<b>Nilai Sekarang Total Biaya</b>	<b>1,632,295</b>	<b>174,528</b>	<b>168,743</b>	<b>163,209</b>	<b>157,911</b>	<b>2,296,686</b>
<b>NPV (Total Manfaat PV - Total Biaya PV)</b>						<b>68,292</b>

1) Nilai layanan Pelanggan didasarkan pada pengurangan biaya penanganan panggilan telepon keluhan pelanggan  
2) Manfaat penting namun tidak berwujud adalah kemampuan untuk menawarkan layanan yang saat ini ditawarkan oleh pesaing

Gambar 2.7  
Analisis Biaya-Manfaat - Metode *Discounted Arus Kas*

### 3. Kelayakan Organisasi

Teknik terakhir yang digunakan untuk analisis kelayakan adalah menilai kelayakan organisasi sistem: seberapa baik sistem pada akhirnya akan diterima oleh

penggunanya dan diadopsi ke dalam operasi organisasi yang sedang berlangsung. Ada banyak faktor organisasi yang dapat berdampak pada proyek. Pengembang berpengalaman umumnya tahu bahwa kelayakan organisasi dapat menjadi dimensi kelayakan yang paling sulit untuk dinilai. Intinya, analisis kelayakan organisasi berusaha menjawab pertanyaan "Jika kita membangunnya, apakah mereka akan hadir?".

Salah satu cara untuk menilai kelayakan organisasi dari proyek adalah dengan memahami seberapa baik tujuan proyek sejalan dengan tujuan bisnis organisasi. Penyelarasan strategis adalah kesesuaian antara proyek dan strategi bisnis. Semakin besar kesesuaiannya, semakin sedikit risiko proyek, dari perspektif kelayakan organisasi. Misalnya, jika departemen pemasaran telah memutuskan untuk lebih fokus pada pelanggan, maka proyek CRM (*Customer Relationship Management*) yang menghasilkan informasi pelanggan secara terintegrasi akan memiliki keselarasan strategis yang kuat dengan tujuan pemasaran. Banyak proyek gagal ketika departemen Teknologi Informasi sendiri yang memprakarsainya, di mana hanya ada sedikit atau bahkan tidak ada sama sekali keselarasan dengan unit bisnis atau strategi organisasi.

Cara kedua untuk menilai kelayakan organisasi adalah dengan melakukan analisis terhadap pemangku kepentingan (*stakeholder analysis*). Seorang pemangku kepentingan adalah orang, kelompok, atau organisasi yang dapat memengaruhi (atau dapat dipengaruhi oleh) sistem baru. Secara umum, pemangku kepentingan yang paling penting dalam pengenalan sistem baru adalah pengagas proyek, pengguna sistem, dan manajemen organisasi (lihat Gambar 2.8), tetapi terkadang sistem juga memengaruhi pemangku kepentingan lainnya. Misalnya, departemen Sistem Informasi dapat menjadi pemangku kepentingan dari suatu sistem karena pekerjaan atau peran Sistem Informasi dapat diubah secara signifikan setelah implementasi sistem.

Pengagas (*champion*) adalah eksekutif tingkat tinggi dan biasanya (tetapi tidak selalu) adalah sponsor proyek yang menciptakan permintaan sistem. Pengagas mendukung proyek dengan menyediakan waktu dan sumber daya (misalnya pendanaan), dukungan politik dalam organisasi dengan mengomunikasikan pentingnya sistem kepada pembuat keputusan organisasi lainnya. Lebih dari satu pengagas lebih disukai karena jika seorang pengagas meninggalkan organisasi, dukungan terhadap proyek juga akan menghilang.

	<b>Wewenang</b>	<b>Peran</b>
Pengagas ( <i>Champion</i> )	<p>Seorang Pengagas:</p> <ol style="list-style-type: none"> <li>1. Memulai Proyek</li> <li>2. Mempromosikan Proyek</li> <li>3. Mengalokasikan waktunya untuk proyek tersebut</li> <li>4. Menyediakan sumber daya</li> </ol>	<ol style="list-style-type: none"> <li>1. Membuat presentasi tentang tujuan proyek dan manfaat yang diusulkan kepada para eksekutif yang akan mendapat manfaat langsung dari sistem</li> <li>2. Membuat prototipe sistem untuk menunjukkan nilainya yang potensial</li> </ol>
Manajemen Organisasi ( <i>Organizational Management</i> )	<p>Manager Organisasi:</p> <ol style="list-style-type: none"> <li>1. Mengetahui tentang Proyek</li> <li>2. Anggaran uang yang cukup untuk proyek tersebut</li> <li>3. Dorong pengguna untuk menerima dan menggunakan sistem</li> </ol>	<ol style="list-style-type: none"> <li>1. Buat presentasi kepada manajemen tentang tujuan proyek dan manfaat yang diusulkan</li> <li>2. Pasarkan manfaat sistem, menggunakan memo dan buletin organisasi</li> <li>3. Dorong juara untuk berbicara tentang proyek dengan rekan-rekannya.</li> </ol>
Pengguna Sistem ( <i>System User</i> )	<p>Pengguna:</p> <ol style="list-style-type: none"> <li>1. Membuat keputusan yang memengaruhi proyek</li> <li>2. Melakukan kegiatan langsung untuk proyek</li> <li>3. Pada akhirnya menentukan apakah proyek berhasil dengan menggunakan atau tidak menggunakan sistem</li> </ol>	<ol style="list-style-type: none"> <li>1. Menetapkan pengguna peran resmi pada tim proyek.</li> <li>2. Menetapkan tugas khusus pengguna untuk dilakukan, dengan tenggat waktu yang jelas.</li> <li>3. Meminta umpan balik dari pengguna secara teratur (misalnya Pada pertemuan mingguan).</li> </ol>

**Gambar 2.8**  
**Stakeholder Penting untuk Kelayakan Organisasi**

Sementara pengagas memberikan dukungan sehari-hari untuk sistem, manajemen organisasi juga perlu mendukung proyek. Dukungan manajemen seperti itu meyakinkan kepada seluruh organisasi bahwa sistem akan memberikan kontribusi yang berharga dan bahwa sumber daya yang diperlukan akan tersedia. Idealnya, manajemen harus mendorong orang-orang dalam organisasi untuk menggunakan sistem dan menerima banyak perubahan yang mungkin akan ditimbulkan oleh sistem.

Sekelompok pemangku kepentingan ketiga adalah pengguna sistem yang pada akhirnya akan menggunakan sistem begitu telah diinstal dalam organisasi. Seringkali tim proyek bertemu dengan pengguna hanya di awal proyek, dan setelah itu tidak lagi sampai setelah sistem dibuat. Dalam situasi seperti ini, jarang produk akhir dapat memenuhi harapan dan kebutuhan pengguna. Partisipasi pengguna harus dipromosikan sepanjang proses pengembangan untuk memastikan bahwa sistem akhir akan diterima dan digunakan, dengan membuat pengguna terlibat secara aktif dalam pengembangan sistem (misalnya: memberikan umpan balik dan membuat keputusan).

Studi kelayakan akhir membantu organisasi melakukan investasi yang lebih bijaksana tentang proyek Sistem Informasi karena memaksa tim proyek untuk mempertimbangkan faktor teknis, ekonomi, dan organisasi yang dapat mempengaruhi proyek-proyek yang dikembangkan. Perlu diingat, studi kelayakan harus direvisi beberapa kali selama proyek pada titik-titik di mana tim proyek membuat keputusan penting tentang sistem (misalnya sebelum desain dimulai). Studi kelayakan akhir dapat digunakan untuk mendukung dan menjelaskan keputusan-keputusan penting dan strategis yang dibuat di sepanjang siklus SDLC.



### Latihan

Untuk memperdalam pemahaman Anda mengenai materi di atas, kerjakanlah latihan berikut!

- 1) Jelaskan tujuan dari analisis kelayakan! Bagaimana kegiatan tersebut dilakukan dalam proses pemilihan proyek?
- 2) Jelaskan tiga dimensi analisis kelayakan!
- 3) Jelaskan perbedaan antara nilai tidak berwujud dan nilai berwujud (nyata), berikan contoh masing-masing!
- 4) Jelaskan beberapa hal yang berpotensi menyebabkan proyek menjadi "berisiko" dalam hal kelayakan teknis!
- 5) Jelaskan langkah-langkah untuk menilai kelayakan ekonomi!

#### Petunjuk Jawaban Latihan

- 1) Tujuan analisis kelayakan adalah untuk menilai apakah suatu proyek layak untuk direalisasikan atau tidak. Analisis kelayakan dilakukan dengan cara melakukan penelitian pendahuluan yang tidak terlalu rinci, yaitu hanya mengumpulkan informasi kelayakan dari proyek yang akan dilaksanakan, dan bukan untuk menyusun kebutuhan (*requirement*) sistem.
- 2) Tiga dimensi analisis kelayakan sistem.
  - a) Kelayakan Teknis: untuk menjawab pertanyaan “dapatkah sistem dibangun?”. Dimensi ini mencakup keakraban pengguna dan analis dengan sistem yang akan dikembangkan, keakraban dengan teknologi yang akan digunakan dalam pengembangan sistem, ukuran proyek, serta kompatibilitas sistem baru dengan teknologi yang sudah ada di organisasi.
  - b) Kelayakan ekonomi: untuk menjawab pertanyaan “haruskah sistem dibangun?”. Dimensi ini mencakup biaya pengembangan, biaya operasi tahunan, manfaat tahunan, serta manfaat dan biaya tak berwujud.

- c) Kelayakan organisasi: untuk menjawab pertanyaan "ketika sistem dibangun, akankah para pemangku kepentingan menyambutnya?". Dimensi ini mencakup keterlibatan pemangku kepentingan dan keselarasan proyek dengan sistem bisnis yang sedang berjalan.
- 3) Nilai berwujud (*tangible benefits*) adalah manfaat nyata yang dapat diterima oleh organisasi, misalnya: pendapatan dari peningkatan penjualan, penghematan/pengurangan biaya (pengurangan staf dapat mengurangi biaya gaji, pengurangan tingkat persediaan dapat mengurangi biaya persediaan). Nilai tidak berwujud (*intangible benefits*) adalah manfaat tidak nyata yang dapat diterima oleh organisasi, misalnya: peningkatan pelayanan pelanggan untuk mengurangi keluhan pelanggan (dapat mengurangi biaya telepon terkait keluhan pelanggan).
- 4) Beberapa hal yang berpotensi menyebabkan proyek menjadi "berisiko" dalam hal kelayakan teknis.
- a) Ketika pengguna dan analis tidak akrab dengan sistem yang akan dikembangkan, misalnya: ketika analis tidak familiar pada area aplikasi bisnis sehingga salah memahami kebutuhan pengguna; ketika pengguna sendiri kurang terbiasa dengan sistem aplikasi sehingga pengembangan sistem baru lebih berisiko daripada jika tetap melanjutkan menggunakan sistem yang ada, karena sistem yang ada cenderung lebih dipahami.
  - b) Ketika analis tidak akrab dengan teknologi yang akan digunakan dalam pengembangan sistem. Ketika suatu sistem yang dikembangkan akan menggunakan teknologi yang belum pernah digunakan sebelumnya dalam organisasi, kemungkinan besar menimbulkan masalah keterlambatan, disebabkan karena kebutuhan untuk mempelajari bagaimana menggunakan teknologi tersebut.
- 5) Langkah-langkah untuk menilai kelayakan ekonomi sebagai berikut.
- a) Mengidentifikasi biaya dan manfaat, dilakukan dengan cara membuat daftar biaya dan manfaat nyata untuk proyek, termasuk biaya satu kali penggunaan dan penggunaan berulang.
  - b) Menentukan biaya dan manfaat (baik yang berwujud maupun yang tidak berwujud), melalui kerjasama dengan pengguna bisnis dan profesional Teknologi Informasi dalam menentukan nilai untuk setiap biaya dan manfaat. Dapat juga dilakukan dengan memperkirakan kisaran nilai untuk biaya atau manfaat, kemudian menetapkan estimasi untuk setiap nilai.
  - c) Menetapkan Arus Kas (baik arus masuk maupun arus keluar). Pengukuran dilakukan dengan memperkirakan selama beberapa periode mendatang, kemudian dievaluasi menggunakan beberapa teknik untuk menilai apakah

manfaat yang diproyeksikan akan sesuai dengan biaya yang akan dikeluarkan. Beberapa teknik evaluasi yang dapat digunakan adalah; *Return on Investment* (ROI) untuk mengukur tingkat pengembalian rata-rata yang diperoleh dari uang yang diinvestasikan dalam proyek, *Break-Even Point* (BEP) untuk mengukur nilai titik impas proyek, dan *Net Present Value* (NPV) untuk menentukan apakah total nilai sekarang dari manfaat lebih besar atau kurang dari total nilai sekarang dari biaya.



## Rangkuman

Mendefinisikan proyek-proyek sistem adalah bagian dari perencanaan sistem setelah suatu proyek sistem prioritas ditentukan dan analis sistem telah ditunjuk. Pada kegiatan ini analis melakukan suatu studi untuk mencari alternatif-alternatif pemecahan terbaik yang paling layak untuk dikembangkan. Kegitan tersebut meliputi kegiatan mengidentifikasi kembali ruang lingkup dan sasaran proyek sistem, melakukan studi kelayakan, dan menilai kelayakan proyek sistem.

Ruang lingkup dan sasaran proyek-proyek sistem perlu didefinisikan kembali oleh analis dengan cara berdiskusi dengan manajemen untuk mengkaji kembali mengenai ruang lingkup dan sasaran sistem. Tim analis dapat memberikan usulan-usulan tambahan bila dipandang perlu. Ruang lingkup dan sasaran dari proyek sistem harus betul-betul dipahami terlebih dahulu oleh tim analis, karena hal ini sangat erat hubungannya terutama dalam perencanaan biaya dan waktu pengembangan sistem yang akan diperkirakan oleh analis.

Tujuan utama perencanaan proyek sistem adalah menyediakan sebuah kerangka kerja yang memungkinkan manajer proyek atau analis membuat estimasi yang dapat dipertanggungjawabkan mengenai sumber daya, biaya, dan jadwal. Estimasi dibuat dengan sebuah kerangka waktu yang terbatas pada awal sebuah proyek sistem informasi dan diperbarui secara teratur selagi proyek sedang berjalan. Meskipun estimasi juga merupakan suatu seni seperti juga pada sains, aktivitas ini tidak dilakukan secara serampangan. Terdapat teknik-teknik tertentu yang dapat digunakan untuk melakukan estimasi.

Analisis kelayakan digunakan untuk memberikan rincian lebih lanjut tentang risiko yang terkait dengan sistem yang diusulkan, mencakup kelayakan teknis, ekonomi, dan organisasi. Kelayakan teknis berfokus pada apakah sistem dapat dibangun, dengan memeriksa risiko yang terkait dengan keakraban pengguna dan analis dengan aplikasi, keakraban dengan teknologi, ukuran proyek, dan kompatibilitas dengan sistem yang ada. Kelayakan ekonomi membahas apakah sistem harus dibangun. Ini mencakup analisis biaya-manfaat dari biaya pengembangan, biaya operasional, manfaat berwujud, dan biaya dan manfaat tidak berwujud. Analisis kelayakan organisasi menilai seberapa baik sistem akan diterima oleh penggunanya dan integrasikan ke dalam operasi organisasi yang sedang berlangsung. Penyelarasan strategis proyek dan analisis pemangku kepentingan dapat digunakan untuk menilai dimensi kelayakan ini.



## Tes Formatif 1

Pilihlah satu jawaban yang paling tepat!

- 1) Tujuan penelitian pendahuluan dalam rangkaian kegiatan Studi Kelayakan adalah, *kecuali* ....
  - A. memahami operasi sistem yang lama
  - B. menganalisis permasalahan-permasalahan yang terjadi pada sistem yang lama
  - C. menentukan kebutuhan-kebutuhan pemakai secara garis besar untuk dapat mencapai sasaran sistem
  - D. menentukan detail kebutuhan fungsional sistem sebagai landasan untuk mengembangkan sistem baru
- 2) Misalkan dalam sebuah proyek pengembangan bisnis rental komputer dan internet, yang termasuk manfaat tidak langsung dalam analisis Biaya-Manfaat berikut ini adalah ....
  - A. memaksimalkan kerja teknisi dalam melakukan perawatan berkala secara rutin terhadap perangkat komputer untuk mengurangi frekuensi kerusakan.
  - B. mendesain sistem saklar lampu otomatis untuk menghemat pemakaian listrik
  - C. menawarkan paket rental yang beragam untuk meningkatkan waktu sewa rata-rata pelanggan
  - D. mengurangi jumlah teknisi dalam batas yang normal di mana operasi bisnis tetap berjalan normal
- 3) Ukuran proyek merupakan salah satu pertimbangan penting dalam penilaian kelayakan teknis. Ukuran sebuah proyek dapat dinilai dari ....
  - A. jumlah orang dalam tim pengembangan
  - B. lamanya waktu yang dibutuhkan untuk penyelesaian proyek
  - C. banyaknya fitur yang akan disediakan dalam sistem yang dikembangkan
  - D. semua jawaban benar
- 4) Berikut ini adalah salah satu cara untuk menilai kelayakan organisasi dalam kegiatan analisis kelayakan sistem, *kecuali* ....
  - A. memahami seberapa baik tujuan proyek sejalan dengan tujuan bisnis organisasi
  - B. menganalisis seberapa jauh pemangku kepentingan terlibat secara aktif dalam pengembangan sistem
  - C. menganalisis kompatibilitas sistem baru dengan teknologi yang sudah ada dalam organisasi
  - D. jawaban A dan B benar.

- 5) Terkait dengan soal nomor 1 (penelitian pendahuluan), yang dapat bertindak sebagai responen pada kegiatan tersebut adalah ....
  - A. manajer departemen, sebab manajer departemen yang paling memahami kondisi di dalam organisasi
  - B. pengguna akhir sistem (*end-user*), sebab pengguna akhir yang paling berkepentingan dalam operasional sistem pada saat sistem telah dibangun
  - C. perekayasa sistem, sebab perekayasa sistem yang akan membangun sistem tersebut
  - D. A, B, dan C, semua dilibatkan
- 6) Studi kelayakan merupakan suatu studi yang dilakukan untuk menentukan kemungkinan apakah pengembangan proyek sistem layak diteruskan atau dihentikan. Studi kelayakan ini hanya diberlakukan pada proyek-proyek skala besar saja.
  - A. Benar
  - B. Salah
- 7) Dalam analisis kelayakan secara teknis, kompatibilitas sistem baru dengan teknologi yang sudah pada organisasi tidak menjadi hal yang perlu diperhitungkan dengan serius, sebab sistem baru sudah barang tentu memiliki teknologi yang lebih baik daripada sistem lama.
  - A. Benar
  - B. Salah
- 8) Studi mengenai seberapa baik sistem pada akhirnya akan diterima oleh para pemangku kepentingan, adalah juga tergolong dalam studi kelayakan secara teknis.
  - A. Benar
  - B. Salah
- 9) Jumlah tahun yang dibutuhkan suatu perusahaan untuk mengembalikan investasi awalnya disebut *Break Even Point* (BEP).
  - A. Benar
  - B. Salah
- 10) Dalam analisis kelayakan secara ekonomi, biaya dan manfaat dapat dikategorikan menjadi: biaya pengembangan, biaya operasional, manfaat nyata, serta manfaat tidak berwujud.
  - A. Benar
  - B. Salah

Cocokkanlah jawaban Anda dengan Kunci Jawaban Tes Formatif 1 yang terdapat di bagian akhir modul ini. Hitunglah jawaban yang benar. Kemudian, gunakan rumus berikut untuk mengetahui tingkat penguasaan Anda terhadap materi Kegiatan Belajar 1.

$$\text{Tingkat Penguasaan} = \frac{\text{Jumlah Jawaban yang Benar}}{\text{Jumlah Soal}} \times 100$$

Arti tingkat penguasaan

<70%

70% - 79%

80% - 89%

90% - 100%

kurang

cukup

baik

baik sekali

Apabila mencapai tingkat penguasaan 80% atau lebih, Anda dapat meneruskan dengan Kegiatan Belajar 2. **Bagus!** Jika masih di bawah 80%, Anda harus mengulangi materi Kegiatan Belajar 1, terutama bagian yang belum dikuasai.

## Membuat Rencana Proyek

Scara historis, Departemen Teknologi Informasi/TI (*Information Technology Department*) cenderung memilih proyek dengan metode *ad hoc*: pertama masuk, pertama keluar atau karena pengaruh politik. Dalam beberapa tahun terakhir, departemen TI telah mengumpulkan informasi proyek dan memetakan kontribusi proyek untuk tujuan bisnis dengan menggunakan perspektif portofolio proyek. Manajemen portofolio proyek, proses memilih proyek, penentuan prioritas, dan pemantauan proyek, telah menjadi faktor penentu keberhasilan bagi departemen TI menghadapi begitu banyak pilihan proyek potensial namun dengan keberadaan sumber daya yang terbatas. Perangkat lunak untuk manajemen portofolio proyek, seperti *Manajemen Portofolio Hewlett Packard*, *Primavera Systems ProSight*, dan *project.net open-source*, telah menjadi alat yang sangat populer dan berharga bagi organisasi TI dalam memilih proyek.

Setelah proyek dipilih dan ditetapkan oleh komite persetujuan proyek, dan hasil studi kelayakan telah dinyatakan layak untuk diteruskan, proyek pengembangan sistem menjalani proses menyeluruh yaitu proses perencanaan proyek dalam kerangka waktu tertentu, dengan biaya minimum untuk mencapai hasil yang diinginkan. Dalam organisasi besar atau pada proyek besar, peran manajer proyek biasanya diisi oleh seorang spesialis profesional dalam manajemen proyek. Dalam organisasi yang lebih kecil atau pada proyek yang lebih kecil, analis sistem dapat mengisi peran ini.

Seorang manajer proyek memiliki tanggung jawab utama untuk mengelola ratusan tugas dan peran yang perlu dikoordinasikan dengan cermat. Banyak produk perangkat lunak tersedia untuk mendukung kegiatan manajemen proyek. Meskipun pengalaman pelatihan dan perangkat lunak tersedia untuk membantu manajer proyek, seringkali tuntutan yang tidak masuk akal yang ditetapkan oleh sponsor proyek dan manajer bisnis dapat membuat manajemen proyek menjadi sangat sulit. Jadwal yang terlalu optimis dianggap sebagai salah satu masalah terbesar yang dihadapi proyek; berharap mendorong penyelesaian proyek lebih cepat, malah menghasilkan penundaan. Dengan demikian, faktor penentu keberhasilan untuk manajemen proyek adalah mulai dengan penilaian realistik dari pekerjaan yang perlu diselesaikan dan kemudian mengelola proyek sesuai dengan rencana. Ini dapat dicapai dengan mengikuti langkah-langkah dasar manajemen proyek sebagaimana akan diuraikan dalam bagian ini. Pertama, manajer proyek memilih metodologi pengembangan sistem yang sesuai

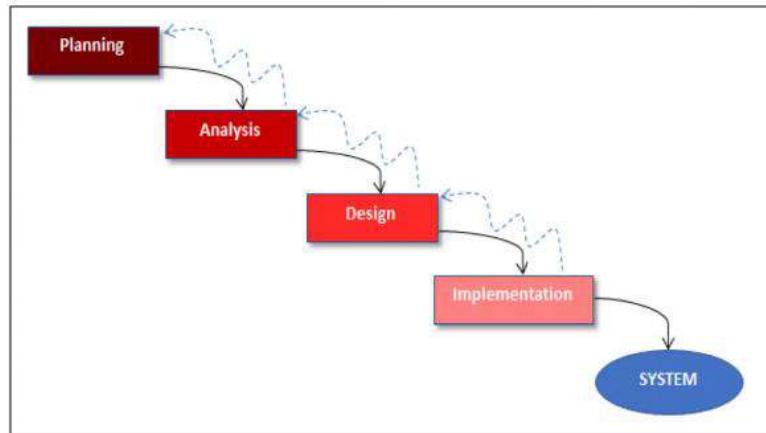
dengan karakteristik proyek. Selanjutnya perkiraan jangka waktu dibuat berdasarkan ukuran sistem yang dikembangkan. Kemudian, daftar tugas yang harus dilakukan dibuat, sebagai dasar dari rencana kerja proyek. Akhirnya, manajer proyek memantau proyek dan merevisi estimasi sebagai hasil pekerjaan.

## A. OPSI-OPSI METODOLOGI PENGEMBANGAN SISTEM

Seperti yang telah dibahas pada Modul 1 Kegiatan Belajar 2, Siklus Hidup Pengembangan Sistem (*System Development Life Cycle/SDLC*) menyediakan landasan bagi metodologi-metodologi yang digunakan untuk mengembangkan sistem informasi. Metodologi-metodologi tersebut adalah pendekatan formal untuk mengimplementasikan SDLC. Ada banyak metodologi pengembangan sistem yang berbeda, dan setiap metodologi berbeda dalam hal mengimplementasikan fase-fase SDLC. Beberapa metodologi merupakan standar formal yang digunakan oleh lembaga pemerintah, sementara yang lain telah dikembangkan oleh perusahaan konsultan untuk dijual kepada klien. Banyak organisasi memiliki metodologi internal mereka sendiri yang telah didefinisikan selama bertahun-tahun, dan mereka menguraikannya dengan lengkap bagaimana setiap fase SDLC harus dilakukan di organisasi itu. Pada kegiatan belajar ini akan dibahas beberapa metodologi utama yang telah berkembang seiring waktu.

### 1. *Waterfall*

Pada metodologi pengembangan berbasis *waterfall* (air terjun), analis dan pengguna bersama-sama merumuskan suatu kegiatan secara berurutan dari satu fase ke fase berikutnya (Gambar 2.9). Hasil utama untuk setiap fase disampaikan kepada komite persetujuan dan sponsor proyek untuk disetujui selama proyek berjalan dari fase ke fase. Setelah pekerjaan yang dihasilkan dalam satu fase disetujui, fase berakhir, dan fase berikutnya dimulai. Seiring kemajuan proyek dari fase ke fase, proyek bergerak maju dengan cara yang sama seperti air terjun. Meskipun dimungkinkan untuk melangkah mundur melalui fase (misalnya dari desain kembali ke analisis), namun ini cukup sulit (bayangkan ketika seekor ikan yang mencoba berenang ke hulu dari air terjun).



Gambar 2.9  
Model *Waterfall*

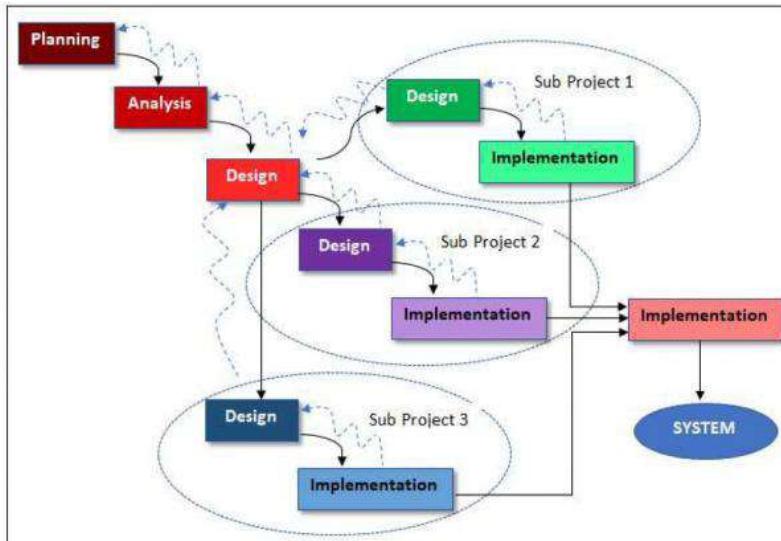
Metodologi pengembangan *Waterfall* memiliki keunggulan dalam mengidentifikasi kebutuhan pengguna jauh sebelum kegiatan pemrograman dimulai dan membatasi perubahan kebutuhan pengguna saat proyek dimulai. Kelemahan utama adalah bahwa rancangan sistem harus benar-benar dipersiapkan secara matang sebelum pemrograman dimulai, sebab kebutuhan perubahan sistem tidak diakomodasi selama fase-fase berjalan. Kelemahan lain adalah terdapat jeda waktu yang lama sejak usulan sistem dalam fase analisis hingga fase pengujian dan pendistribusian sistem dalam fase implementasi, ini merupakan hal yang patut direnungkan. Selain itu, pada fase pendistribusian sistem seringkali terjadi mekanisme komunikasi yang buruk, sehingga persyaratan penting diabaikan dalam dokumentasi sistem. Jika tim proyek melewatkannya persyaratan penting, pemrograman pasca implementasi mungkin menimbulkan biaya yang mahal. Pengguna mungkin telah lupa tujuan utama sistem, karena begitu panjang rentang waktu antara ditetapkannya ide hingga implementasi aktual. Demikian juga dalam lingkungan bisnis yang dinamis saat ini, sebuah sistem yang telah memenuhi kondisi lingkungan yang ada sejak tahap analisis, mungkin saja perlu pengerjaan ulang pada beberapa hal agar sesuai dengan keadaan lingkungan saat diterapkan. Pengerjaan ulang ini perlu kembali mengimplementasikan fase awal dan membuat perubahan yang diperlukan melalui masing-masing fase berikutnya.

Ada dua varian utama dari model *Waterfall*, yaitu model *Parallel Model* dan *V-Model*.

#### a. Model Parallel

Model Parallel dikembangkan untuk mengatasi permasalahan waktu pengembangan yang lama pada model *Waterfall*. Seperti yang ditunjukkan pada Gambar 2.10, pada tahap awal dibuat desain secara umum untuk keseluruhan sistem. Kemudian proyek dibagi menjadi serangkaian sub proyek yang dapat dirancang dan

diimplementasikan secara paralel. Setelah semua sub proyek selesai, dilakukan integrasi akhir dari bagian-bagian yang terpisah, dan selanjutnya sistem didistribusikan.

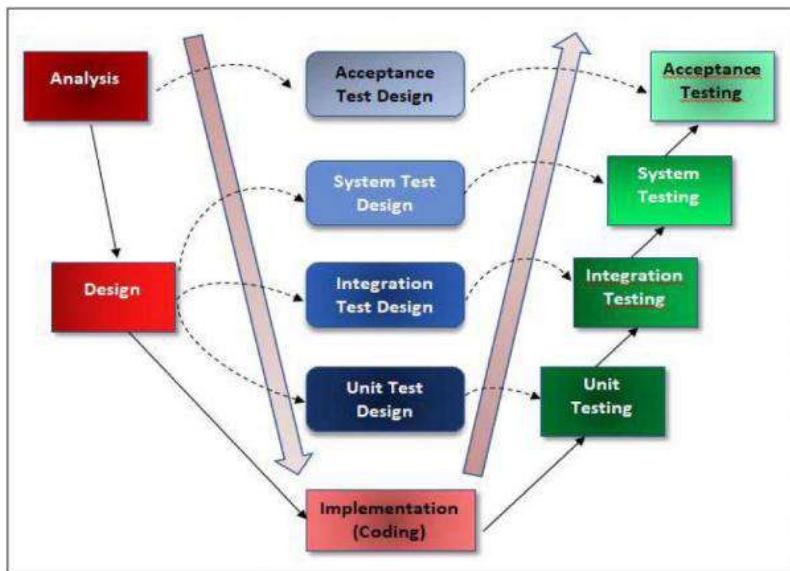


Gambar 2.10  
Model *Parallel*

Dengan konsep pengembangan yang paralel, metodologi pengembangan Parallel dapat mengurangi waktu yang diperlukan untuk menghasilkan suatu sistem. Perubahan dalam lingkungan bisnis tidak menyebabkan perubahan yang besar pada produk yang memerlukan pengerjaan ulang. Pendekatan ini masih mengalami masalah yang disebabkan oleh hasil yang sangat banyak. Masalah lain adalah jika sub proyek tidak didesain sepenuhnya untuk dapat berdiri sendiri, hasil desain dalam satu sub proyek dapat mempengaruhi yang lain, dan pada akhir proyek pengintegrasian sub-sub proyek mungkin cukup rumit.

#### b. V-Model

Model-V adalah variasi lain dari model *Waterfall* yang lebih difokuskan pada pengujian secara eksplisit. Seperti ditunjukkan pada Gambar 2.11, proses pengembangan berlangsung di sisi kiri kemiringan V, yaitu kegiatan penentuan kebutuhan sistem dan perancangan komponen sistem. Di dasar V, dilakukan kegiatan pemrograman. Di kemiringan sisi kanan model, terdapat pengujian komponen, pengujian integrasi, dan berakhir pada pengujian penerimaan sistem. Konsep kunci dari model ini adalah bahwa ketika kebutuhan sistem ditentukan dan komponen dirancang, pengujian untuk elemen-elemen tersebut juga dilakukan. Dengan cara ini, setiap tingkat pengujian akan secara jelas terkait dengan fase analisis atau fase desain, membantu dalam memastikan kualitas dan relevansi pengujian, serta memaksimalkan efektivitas pengujian.



Gambar 2.11  
Model-V

Model-V sederhana dan mudah diimplementasikan, serta dapat meningkatkan kualitas sistem secara keseluruhan melalui penekanan pada pengembangan rencana pengujian. Pengujian dan keahlian lebih difokuskan pada fase-fase awal proyek, daripada diletakkan pada bagian akhir proyek. Pengujinya memperoleh pemahaman tentang proyek lebih awal. Namun, model ini kaku dalam proses *Waterfall*, dan tidak selalu sesuai pada keadaan lingkungan bisnis yang bersifat dinamis (selalu berubah-ubah).

## **2. Rapid Application Development**

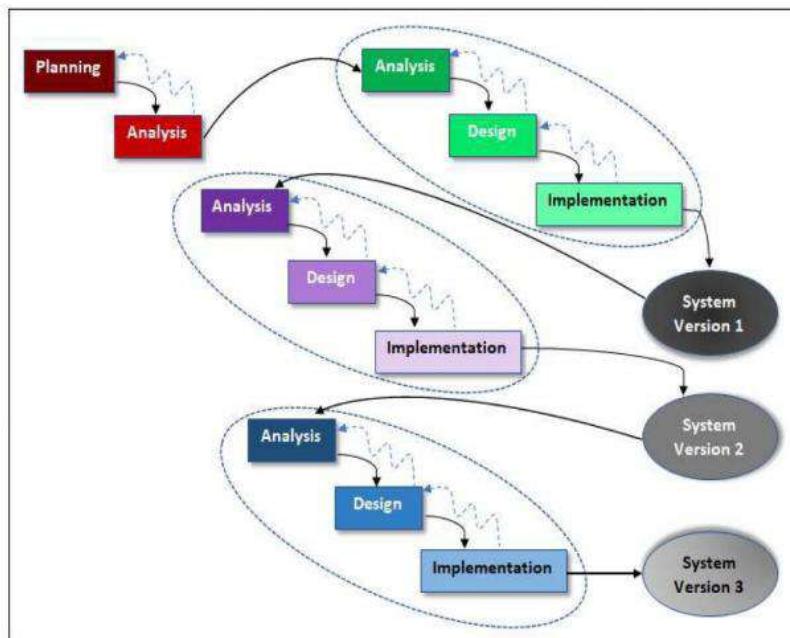
Rapid Application Development (RAD) adalah kumpulan metodologi yang muncul untuk merespon kelemahan pada metode pengembangan *Waterfall* dan variasinya. RAD menggabungkan teknik-teknik khusus dan peralatan komputer guna mempercepat fase analisis, desain, dan implementasi untuk mendapatkan sebagian dari sistem yang dikembangkan dengan cepat ke tangan pengguna untuk dievaluasi dan mendapatkan umpan balik.

Tool pengembangan berupa CASE (*Computer-Aided Software Engineering*), JAD (*Joint Application Development*), bahasa pemrograman visual (misalnya Visual Basic.NET), dan pembangkit kode (*code generator*) semuanya dapat memainkan peran dalam RAD. Meskipun RAD dapat meningkatkan kecepatan dan kualitas pengembangan sistem, RAD juga dapat menimbulkan masalah dalam mengelola keinginan pengguna. Karena sistem dikembangkan lebih cepat dan pengguna mendapatkan pemahaman yang lebih baik tentang teknologi informasi, harapan pengguna dapat meningkat secara dramatis dan kebutuhan sistem dapat berkembang selama fase proyek berlangsung.

RAD dapat dilakukan dengan berbagai cara, seperti *iterative development*, *system prototyping*, dan *throwaway prototyping*.

a. *Iterative Development*

Pengembangan *iterative* memecah satu proyek menjadi beberapa bagian untuk dikembangkan secara berurutan (Fowler, 2005). Kebutuhan paling penting dan mendasar digabungkan ke dalam versi pertama sistem. Versi ini dikembangkan dengan cepat melalui model SDLC mini, dan sekaligus diimplementasikan. Pengguna dapat memberikan umpan balik yang berharga untuk dimasukkan ke dalam versi sistem selanjutnya (lihat Gambar 2.12) Pengembangan secara berulang mendapatkan versi awal sistem kepada pengguna dengan cepat sehingga nilai bisnis dari proyek dapat langsung disediakan. Selama waktu pengguna bekerja dengan versi awal sistem, kebutuhan tambahan penting dapat diidentifikasi dan dimasukkan ke dalam versi berikutnya. Kelemahan utama dari model pengembangan *iterative* adalah bahwa ketika pengguna mulai menggunakan sistem (yang memang sengaja dibuat tidak lengkap), pengguna sering tidak menyadari bahwa pada versi awal hanya kebutuhan paling kritis dari sistem yang akan tersedia, dan mereka terus menuntut untuk menggunakan versi yang sempurna.

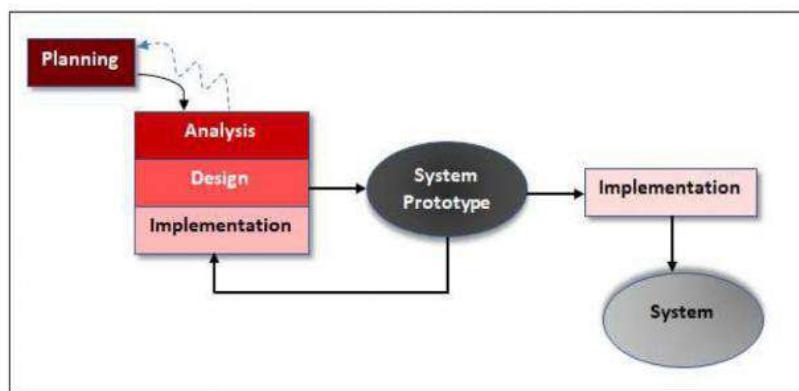


Gambar 2.12  
Model *Iterative*

b. *System Prototyping*

Model *System Prototyping* melakukan fase analisis, desain, dan implementasi secara keseluruhan (tuntas) untuk mengembangkan versi sederhana dari sistem yang diusulkan, dan mendistribusikan kepada pengguna untuk mendapatkan evaluasi dan

umpan balik (lihat Gambar 2.13). *System Prototype* adalah versi "cepat dan kotor (tidak sempurna)" dari sistem, yang menyediakan fitur minimal. Berdasarkan tanggapan dan komentar/masukan dari pengguna, pengembang melakukan analisis ulang, mendesain ulang, dan mengimplementasikan kembali *prototype* kedua yang mengoreksi kekurangan *prototype* pertama dan menambahkan lebih banyak fitur. Siklus ini berlanjut hingga analis, pengguna, dan sponsor proyek setuju bahwa *prototype* telah menyediakan fungsionalitas yang cukup untuk dipasang dan digunakan dalam organisasi. *System prototyping* sangat cepat menyediakan sistem bagi pengguna untuk mengevaluasi dan meyakinkan pengguna bahwa kemajuan sedang dibuat. Pendekatan ini sangat berguna ketika pengguna memiliki kesulitan mengungkapkan seluruh kebutuhan pada fase awal pengembangan sistem. *Prototype* berfungsi sebagai sebuah mekanisme untuk mengidentifikasi kebutuhan pengguna yang sesungguhnya. Namun, kekurangannya adalah kurangnya analisis metodis yang cermat sebelum membuat keputusan melanjutkan ke tahap desain dan implementasi. *System Prototype* mungkin memiliki beberapa keterbatasan pada rancangan sistem yang paling mendasar, yang merupakan akibat langsung dari pemahaman sistem yang tidak memadai terkait dengan kebutuhan sistem yang sebenarnya pada fase awal proyek.



Gambar 2.13  
Model System Prototyping

c. *Throwaway Prototyping*

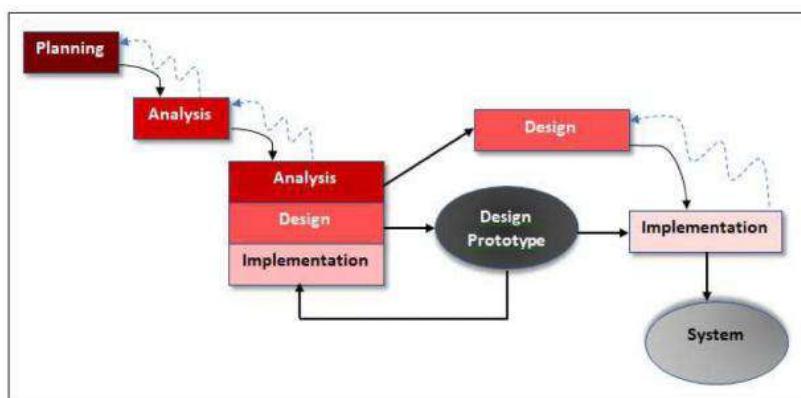
Model *Throwaway Prototyping* tergolong dalam model pengembangan *prototype*, akan tetapi penekanannya pada penggunaan *prototype* terutama untuk mengeksplorasi alternatif desain daripada sebagai sistem yang berfungsi seperti dalam *System Prototyping*. Seperti yang ditunjukkan pada gambar 2.14, *Throwaway Prototyping* memiliki fase analisis yang cukup lengkap untuk mengumpulkan kebutuhan pengguna dan mengembangkan ide-ide untuk konsep sistem. Namun, banyak fitur yang disarankan oleh pengguna mungkin tidak dipahami dengan baik, dan mungkin ada masalah teknis yang sulit untuk dipecahkan. Masing-masing masalah

diperiksa dengan menganalisis, merancang, dan membangun *prototype design*. *Prototype design* tidak dimaksudkan sebagai sistem yang sudah dapat bekerja (beroperasi), namun hanya berisi rincian untuk memungkinkan pengguna memahami masalah yang sedang menjadi fokus perhatian.

#### *Contoh*

Anggaplah pengguna tidak memahami dengan jelas tentang cara kerja sistem entri pesanan. Tim analis dapat membuat serangkaian halaman yang dapat dilihat pada browser Web untuk membantu pengguna memvisualisasikan sistem entri pesanan. Melalui *mock up* desain, pengguna dapat melihat visualisasi calon aplikasi secara nyata sehingga pengguna dapat memberikan masukan kepada desainer jika tampilan *mock up* dirasa belum sesuai dengan permintaan sebelumnya.

Sebagai contoh, anggaplah bahwa tim proyek perlu mengembangkan sebuah program grafis yang canggih. Tim dapat menulis sebagian dari program dengan data buatan untuk memastikan bahwa mereka dapat membuat program yang lengkap dengan sukses.



Gambar 2.14  
Model *Throwaway Prototyping*

Suatu sistem yang dikembangkan menggunakan metodologi *Throwaway Prototyping* ini mungkin memerlukan beberapa *prototype design* selama fase analisis dan desain. Masing-masing *prototype* digunakan untuk meminimalkan risiko yang terkait dengan sistem dengan mengonfirmasi bahwa masalah-masalah penting memang telah dipahami dengan jelas sebelum sistem nyata dibangun. Setelah masalah diselesaikan, proyek dilanjutkan ke fase desain dan implementasi. Pada titik ini, *prototype design* tidak lagi digunakan (dibuang). Di sinilah letak perbedaan mendasar antara pendekatan *Throwaway Prototyping* dan *System Prototyping*, di mana *prototype* berevolusi menjadi sistem akhir.

*Throwaway Prototyping* menyeimbangkan antara manfaat dari fase analisis dan fase desain yang matang dengan manfaat menggunakan *prototype* untuk menyelesaikan segala permasalahan utama sebelum sistem dibangun. Mungkin butuh waktu lebih lama untuk menghasilkan sistem final (akhir) dibandingkan dengan *System Prototyping* (karena *prototype* tidak menjadi sistem final), tetapi pendekatannya biasanya menghasilkan sistem yang lebih stabil dan andal.

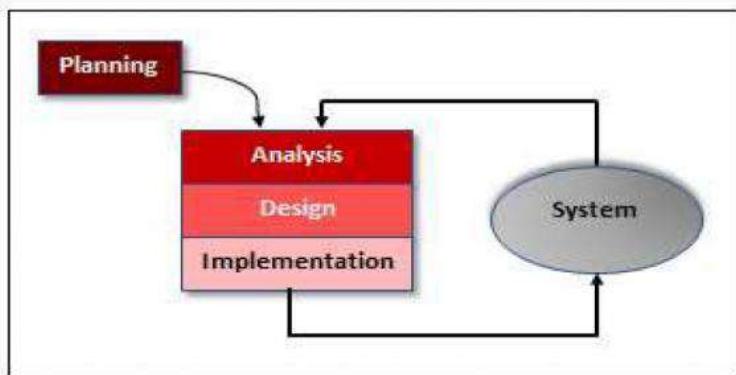
### 3. Agile Development

*Agile Development* adalah sekelompok metodologi pemrograman yang fokus pada penyederhanaan SDLC. Banyak fitur tambahan pada fase pemodelan dan dokumentasi dihilangkan, sebaliknya proses komunikasi melalui interaksi langsung dengan pengguna sistem lebih disukai. Pengembangan proyek menekankan pada hal-hal yang sederhana, pengembangan sistem secara berulang di mana setiap perulangan merupakan serangkaian fase pekerjaan yang lengkap, yaitu: perencanaan, analisis kebutuhan, desain, pengkodean, pengujian, dan dokumentasi (lihat gambar 2.15). Siklus dijaga agar tetap pendek (sekitar satu hingga empat minggu), dan tim pengembangan berfokus pada bagaimana beradaptasi dengan lingkungan bisnis saat ini. Ada beberapa pendekatan populer dalam metodologi pengembangan *Agile*, seperti *Extreme Programming* (XP), Scrum, dan *Dynamic Systems Development Method* (DSDM). Modul ini hanya akan memberikan gambaran secara singkat mengenai *Extreme Programming*.

#### a. Extreme Programming

Metodologi pengembangan *Extreme Programming* (XP) menekankan pada kepuasan pelanggan dan kerja tim (kerja secara kelompok). Komunikasi, kesederhanaan, umpan balik, dan keberanian adalah prinsip utama dalam *Extreme Programming*. Pengembang sistem berkomunikasi dengan pelanggan dan antar sesama programmer. Desain sistem dibuat sederhana dan jelas. Pengujian awal dan pengujian-pengujian lanjutan menyediakan umpan balik, dan pengembang dapat dengan sigap (berani) menanggapi setiap perubahan kebutuhan dan teknologi yang direkomendasikan dari hasil pengujian sistem. Tim proyek juga dijaga agar tetap kecil.

Proyek XP dimulai dengan pengguna menggambarkan apa yang perlu dilakukan sistem. Kemudian, programmer membuat kode program dalam modul kecil dan sederhana, mengujinya untuk memenuhi kebutuhan tersebut. Pengguna harus bersedia untuk memberikan keterangan secara jelas dan tuntas atas segala pertanyaan dan permasalahan yang muncul. Standar sangat penting untuk meminimalkan perbedaan persepsi, sehingga tim XP menggunakan serangkaian nama, deskripsi, dan praktik pengkodean yang umum. Proyek XP memberikan hasil lebih cepat daripada pendekatan RAD, dan tim pengembang jarang mengalami kebuntuan dalam kegiatan pengkajian kebutuhan untuk sistem.



Gambar 2.15  
Model *Extreme Programming*

Untuk proyek-proyek kecil dengan tim pengembang yang tetap, memiliki motivasi tinggi, kompak, dan berpengalaman, XP semestinya dapat bekerja dengan baik. Namun, jika proyek berskala besar dan tim tidak kompak, maka kemungkinan keberhasilan proyek XP akan berkurang. Menggunakan model XP di mana tim pengembang berkolaborasi dengan kontraktor luar tidak menjamin keberhasilan, karena kontraktor luar mungkin tidak pernah "kompak" dengan orang-orang di dalam tim. XP membutuhkan banyak disiplin untuk mencegah proyek menjadi tidak fokus dan kacau. Selain itu, direkomendasikan hanya untuk tim pengembang dalam kelompok kecil (tidak lebih dari 10), dan tidak disarankan digunakan dalam pengembangan aplikasi untuk tujuan atau misi yang sangat penting. Karena hanya sedikit dokumentasi analisis dan desain yang diproduksi dalam XP, maka hanya terdapat dokumentasi kode program. Oleh karena itu, pemeliharaan pada sistem besar yang dikembangkan menggunakan XP mungkin tidak dimungkinkan. Juga, karena sistem informasi bisnis untuk misi yang sangat penting cenderung keberadaannya untuk waktu yang lama, manfaat yang diberikan oleh XP sebagai metodologi pengembangan sistem informasi bisnis diragukan. Akhirnya, metodologi ini memerlukan masukan yang besar dari tempat pengguna, sesuatu yang sering sulit diwujudkan.

#### b. Agile versus Waterfall-Based Methodology

Pendekatan pengembangan berbasis *Agile* telah ada selama lebih dari satu dekade. Praktik pengembangan berbasis *Agile* dilandasi adanya ketidakpuasan terhadap model sekuensial, seperti pendekatan berbasis air terjun (*waterfall*) yang tidak fleksibel. Saat ini, pengembangan berbasis *Agile* telah membuat terobosan ke dalam organisasi pengembangan perangkat lunak, di mana banyak organisasi bereksperimen dengan model *Agile* sambil terus menggunakan pendekatan tradisional *Waterfall*. Para pengembang perangkat lunak secara aktif meneliti untuk mengintegrasikan elemen terbaik dari model *Waterfall* dan model *Agile* dalam praktik pengembangan perangkat

lunak. Proses pengembangan sistem informasi tidak pernah statis. Sebagian besar departemen Sistem Informasi dan manajer proyek mengakui bahwa pilihan metodologi pengembangan "terbaik" tergantung pada karakteristik proyek.

## B. MEMILIH METODOLOGI PENGEMBANGAN YANG TEPAT

Seperti yang ditunjukkan pada pembahasan sebelumnya, terdapat banyak metodologi untuk pengembangan sistem. Tantangan pertama yang dihadapi oleh manajer proyek adalah memilih metodologi yang akan digunakan. Memilih metodologi bukanlah perkara yang sederhana, karena tidak ada satu metodologi yang selalu terbaik. Banyak organisasi memiliki standar dan kebijakan sebagai panduan dalam pemilihan metodologi.

Gambar 2.16 merangkum beberapa kriteria penting dalam pemilihan metodologi, namun satu hal yang tidak dibahas dalam gambar ini adalah tingkat pengalaman tim analis. Banyak metodologi RAD dan *Agile* membutuhkan penggunaan alat dan teknik-teknik baru. Seringkali alat dan teknik ini meningkatkan kompleksitas proyek dan membutuhkan waktu ekstra untuk memahaminya, namun begitu metodologi ini diadopsi dan telah dipahami dapat secara signifikan meningkatkan kecepatan untuk dapat memberikan sistem akhir kepada user.

<b>Kriteria</b>	<b>Metodologi Pengembangan</b>						
	Waterfall	Parallel	V-Model	Iterative	System Prototyping	Throwaway Prototyping	Agile
1	Poor	Poor	Poor	Good	Excellent	Excellent	Excellent
2	Poor	Poor	Poor	Good	Poor	Excellent	Poor
3	Good	Good	Good	Good	Poor	Excellent	Poor
4	Good	Good	Excellent	Good	Poor	Excellent	Good
5	Poor	Good	Poor	Excellent	Excellent	Good	Excellent
6	Poor	Poor	Poor	Excellent	Excellent	Good	Good

Sumber: Dennis (2012)

**Gambar 2.16**  
Kriteria untuk Memilih Sebuah Metodologi

Keterangan tentang Kriteria:

1. Kejelasan kebutuhan pengguna
2. Keakraban dengan teknologi
3. Kompleksitas sistem
4. Keandalan Sistem
5. Kerangka Waktu
6. Visibilitas Jadwal

### 1. Kejelasan Kebutuhan Pengguna

Ketika kebutuhan pengguna mengenai apa yang harus dilakukan sistem tidak jelas, sulit bagi analis untuk memahaminya ketika sedang terlibat dalam perbincangan dengan user, sehingga juga akan sulit menyajikannya dalam laporan tertulis. Pengguna biasanya perlu berinteraksi dengan teknologi untuk benar-benar memahami apa yang dapat dilakukan oleh sistem baru dan cara penerapannya untuk memenuhi kebutuhan mereka. *System Prototyping* dan *Throwaway Prototyping* biasanya lebih tepat ketika kebutuhan pengguna tidak jelas, sebab metode tersebut menyediakan *prototype* kepada pengguna untuk berinteraksi lebih awal pada SDLC. Pengembangan berbasis *Agile* mungkin juga sesuai jika masukan dari pengguna tersedia di lapangan.

### 2. Keakraban dengan Teknologi

Ketika sistem yang dikembangkan akan menggunakan teknologi baru yang tidak dikenal oleh para analis dan programer (misalnya: proyek pengembangan Web pertama menggunakan teknologi Ajax), menerapkan teknologi baru di awal metodologi akan meningkatkan peluang keberhasilan. Jika sistem dirancang tidak akrab dengan teknologi dasar, risiko meningkat karena alat pengembangan mungkin tidak mampu melakukan apa yang dibutuhkan. *Throwaway Prototyping* sangat sesuai untuk situasi di mana ada kekurangakraban dengan teknologi, karena secara eksplisit mendorong pengembang untuk membuat *prototype design* pada area yang memiliki risiko yang tinggi. Model pengembangan *Iterative* juga baik, karena akan tercipta peluang untuk menyelidiki teknologi secara mendalam sebelum desain diselesaikan secara lengkap. Lalu bagaimana dengan *System Prototyping*? Peluang *System Prototyping* hanya sedikit, karena *prototype* awal yang dibangun biasanya hanya mengorek permukaan teknologi baru tersebut. Biasanya, setelah beberapa bulan dan beberapa *prototype* dihasilkan, pengembang baru menemukan kelemahan atau masalah dalam teknologi baru.

### 3. Kompleksitas Sistem

Sistem yang kompleks membutuhkan analisis dan desain yang cermat dan terperinci. *Throwaway Prototyping* sangat cocok untuk analisis dan desain terperinci seperti itu, namun tidak cocok menggunakan *System Prototyping*. Metodologi *Waterfall* dapat menangani sistem yang kompleks, akan tetapi tidak memiliki kemampuan untuk menyediakan sistem atau *prototype* ke tangan pengguna lebih awal, dan beberapa masalah utama mungkin terabaikan. Meskipun metodologi pengembangan *Iterative* memungkinkan pengguna untuk berinteraksi dengan sistem di awal proses, namun tim proyek yang menggunakan metodologi ini cenderung kurang mencurahkan perhatian pada analisis domain masalah yang lengkap, dibandingkan jika mereka menggunakan metodologi yang lain.

#### 4. Keandalan Sistem

Keandalan sistem biasanya merupakan faktor yang sangat penting dalam pengembangan sistem. Namun, keandalan hanyalah salah satu dari beberapa faktor. Untuk beberapa sistem yang dikembangkan, faktor keandalan sangat penting (misalnya aplikasi untuk bidang medis, sistem pertahanan militer), sedangkan untuk sistem yang lain seperti aplikasi *Game* dan *Vidio*, faktor keandalan tidak terlalu penting. *V-Model* berguna ketika keandalan menjadi hal yang penting, karena penekanannya pada pengujian. *Throwaway Prototyping* paling tepat ketika keandalan sistem adalah prioritas tinggi, karena analisis rinci dan fase desain dikombinasikan dengan kemampuan tim proyek untuk menguji berbagai pendekatan melalui *prototype design* sebelum menyelesaikan desain. *System Prototyping* umumnya tidak menjadi pilihan untuk mengembangkan sistem yang mementingkan faktor keandalan, karena kurang hati-hati pada fase analisis dan fase desain yang justru sangat penting untuk sistem yang mementingkan keandalan.

#### 5. Kerangka Waktu

Proyek-proyek yang memiliki jadwal waktu singkat sangat cocok untuk dikembangkan dengan metodologi RAD karena metodologi tersebut dirancang untuk meningkatkan kecepatan pengembangan. Model pengembangan *Iterative* dan *System Prototyping* adalah pilihan yang sangat baik ketika hanya tersedia waktu pendek karena kedua metode ini memungkinkan tim proyek untuk menyesuaikan fungsionalitas dalam sistem berdasarkan waktu tertentu. Jika jadwal proyek mulai melenceng, itu dapat disesuaikan dengan menghilangkan fungsi dari versi *prototype* yang sedang berjalan. Metodologi berbasis *Waterfall* adalah pilihan terburuk ketika waktu menjadi sesuatu yang sangat berharga, karena model *Waterfall* tidak memberi peluang untuk melakukan perubahan dengan mudah.

#### 6. Visibilitas Jadwal

Salah satu tantangan terbesar dalam pengembangan sistem adalah mengetahui apakah suatu proyek akan berjalan sesuai jadwal. Ini terutama berlaku untuk metodologi berbasis *Waterfall*, di mana fase desain dan fase implementasinya terjadi pada akhir proyek. Metodologi RAD memindahkan banyak keputusan desain yang kritis ke fase lebih awal dalam proyek untuk membantu manajer proyek mengenali dan mengatasi faktor risiko, serta menjaga agar ekspektasi tetap terjaga.

### C. MEMPERKIRAKAN KERANGKA WAKTU PROYEK

Seperti yang telah diuraikan sebelumnya, beberapa metodologi pengembangan telah dikembangkan sebagai upaya untuk mempercepat proyek melalui SDLC yang dipercepat, sambil tetap fokus untuk menghasilkan sistem yang berkualitas. Terlepas apakah masalah waktu merupakan suatu masalah yang kritis pada suatu siklus proyek atau tidak, manajer proyek harus mengembangkan perkiraan awal dari jumlah waktu yang akan digunakan dalam pengembangan proyek.

Estimasi dapat dilakukan secara manual atau dengan bantuan paket perangkat lunak estimasi seperti *Construx Estimate*, *Costar*, atau *KnowledgePLAN*, dan banyak lagi yang lainnya. Estimasi yang dibuat pada awal proyek biasanya didasarkan pada kisaran nilai yang memang layak. Angka yang digunakan untuk menghitung estimasi ini dapat berasal dari beberapa sumber, misalnya merujuk pada metodologi yang digunakan, diambil dari proyek yang penugasan dan teknologinya serupa, atau disediakan oleh pengembang yang berpengalaman. Secara umum, angka-angkanya harus konservatif. Praktik yang baik adalah menganalisis penggunaan waktu aktual selama siklus SDLC sehingga angka-angka dapat diketahui sepanjang proses. Dengan demikian, proyek selanjutnya dapat memperoleh manfaat dari penggunaan data yang sebenarnya. Salah satu kekuatan terbesar dari perusahaan konsultan adalah pengalaman masa lalu yang mereka tawarkan kepada suatu proyek. Mereka memiliki perkiraan dan metodologi yang telah dikembangkan, teruji dari waktu ke waktu, dan telah diterapkan pada ratusan proyek.

Ada dua cara utama untuk memperkirakan waktu yang dibutuhkan untuk membangun suatu sistem. Metode paling sederhana adalah menggunakan jumlah waktu yang dihabiskan dalam fase perencanaan untuk memprediksi waktu yang dibutuhkan oleh keseluruhan proyek. Idenya adalah bahwa proyek sederhana akan memerlukan sedikit perencanaan, dan proyek yang kompleks akan membutuhkan lebih banyak perencanaan. Jadi menggunakan jumlah waktu yang dihabiskan dalam fase perencanaan adalah cara yang logis untuk memperkirakan keseluruhan kebutuhan waktu untuk pengembangan proyek. Dengan pendekatan ini, pengembang hanya meluangkan waktu untuk memperkirakan waktu yang digunakan pada fase perencanaan dan menggunakan persentase standar industri (atau persentase dari pengalaman pada organisasi sendiri) untuk menghitung estimasi pada fase SDLC lainnya.

#### *Contoh*

Standar industri menyarankan bahwa sistem aplikasi bisnis "khusus" menghabiskan 15% waktu untuk fase perencanaan, 20% pada fase analisis, 35% pada fase desain, dan 30% pada fase implementasi. Ini menunjukkan bahwa jika suatu proyek membutuhkan waktu empat bulan dalam tahap perencanaan, maka sisa dari proyek tersebut kemungkinan akan memakan total 26,66 per bulan ( $4:0.15=26,66$ ). Persentase industri yang sama digunakan untuk memperkirakan jumlah waktu yang digunakan pada fase yang lainnya, seperti pada gambar 2.17.

	Perencanaan	Analisis	Desain	Implementasi
Standar Industri Khusus Untuk Aplikasi Bisnis	15%	20%	35%	30%
Perkiraan Berdasarkan Angka Aktual Pada Tahap Pertama SDLC	Aktual: 4 Bulan	Perkiraan: 5,33 Bulan	Perkiraan: 9,33 Bulan	Perkiraan: 8 Bulan

Gambar 2.17  
Perkiraan Waktu Proyek Menggunakan Standar Industri

Gambar 2.17 memperlihatkan bahwa jika waktu rata-rata yang digunakan untuk menyelesaikan fase Perencanaan Proyek adalah 4 bulan, maka waktu yang akan digunakan untuk menyelesaikan fase Analisis adalah 5,33 bulan, fase Desain selama 9,33 bulan, dan fase implementasi selama 8 bulan.

Keterbatasan pendekatan ini adalah sulit untuk memperhitungkan secara spesifik pada proyek-proyek umum, yang mungkin lebih sederhana atau lebih sulit daripada proyek "khusus".

Pendekatan yang lebih presisi untuk estimasi waktu penyelesaian proyek adalah *function point approach*. Pendekatan ini lebih kompleks, namun lebih andal untuk memperkirakan waktu penyelesaian suatu proyek.

#### D. MENGEMBANGKAN RENCANA KERJA

Setelah seorang manajer proyek memiliki gagasan umum tentang ukuran dan perkiraan jadwal untuk proyek tersebut, manajer membuat rencana kerja berupa jadwal yang mencantumkan semua tugas yang harus diselesaikan selama proyek. Manajer proyek pertama-tama harus mengumpulkan rincian penting tentang setiap tugas yang harus diselesaikan.

Gambar 2.18 adalah contoh informasi tugas yang diperlukan, termasuk kapan informasi tersebut diselesaikan, orang yang ditugasi untuk melakukan pekerjaan, dan hasil yang peroleh. Tingkat perincian dan jumlah informasi yang dituangkan pada rencana kerja tergantung pada kebutuhan proyek (jumlah perincian biasanya meningkat seiring dengan kemajuan proyek).

Informasi Tugas	Contoh
Nama tugas	Menganalisis kelayakan Ekonomi
Tanggal mulai	1 Januari 2020
Tanggal selesai	18 Januari 2020
Person assigned	Sponsor Proyek: Ir. Agussalim
Hasil akhir ( <i>deliverable</i> )	Analisis Biaya - Manfaat
Status penyelesaian	Lengkap
Prioritas	Tinggi
Sumber daya yang dibutuhkan	Perangkat Lunak <i>Spreadsheet</i>
Perkiraan waktu	16 Jam
Waktu yang sebenarnya	14,5 Jam

Gambar 2.18  
Informasi Tugas

Untuk membuat rencana kerja, manajer proyek mengidentifikasi tugas yang perlu diselesaikan dan menentukan berapa lama masing-masing akan diselesaikan. Selanjutnya tugas disusun dalam struktur rincian kerja.

## 1. Identifikasi Tugas

Perlu diingat bahwa tujuan keseluruhan sistem yang dikembangkan dicatat dalam permintaan sistem, dan tugas manajer proyek adalah mengidentifikasi semua tugas yang akan diperlukan untuk mencapai tujuan tersebut. Metodologi yang dipilih oleh manajer proyek harus menjadi sumber daya yang berharga, berupa metodologi yang paling sesuai untuk proyek dalam menyajikan daftar langkah-langkah kerja dan hasil akhir.

Seorang manajer proyek menggunakan sebuah metodologi, memilih langkah-langkah dan hasil akhir yang diberlakukan untuk proyek saat ini, dan menambahkannya ke rencana kerja. Jika metodologi yang ada tidak tersedia dalam organisasi, metodologi dapat dibeli dari konsultan atau *vendor*, atau berpedoman pada buku-buku teks sebagai panduan. Menggunakan metodologi yang ada adalah cara paling populer untuk membuat rencana kerja, karena sebagian besar organisasi memiliki metodologi yang telah mereka gunakan dalam pengembangan proyek di masa lalu.

Jika seorang manajer proyek lebih suka memulai dari awal, dia dapat menggunakan pendekatan terstruktur, dari atas ke bawah di mana tugas-tugas tingkat tinggi didefinisikan terlebih dahulu, kemudian dipecah menjadi beberapa subtugas. Selanjutnya setiap langkah dipecah dan diberi nomor secara hierarkis. Daftar tugas yang diberi nomor secara hierarkis dengan cara seperti itu disebut struktur rincian pekerjaan, dan merupakan penopang utama rencana kerja proyek.

*Contoh*

ID Tugas	Nama Tugas	Durasi (Hari)	Ketergantungan	Status
1	Fase Desain	82		Open
1.1	Pengembangan Dokumen Desain Basis Data	18		Open
1.1.1	<i>Staging</i> Desain Basis Data	9		Open
1.1.2	<i>Suspense</i> Desain Basis Data	9		Open
1.2	Pengembangan Dokumen Desain <i>rejects-handling</i>	9	1.1.1, 1.1.2	Open
1.2.1	Desain <i>rejects-handling</i> Mesin	9		Open
1.3	Pengembangan Desain Dokumen OLAP	9	1.1.1, 1.1.2	Open
1.3.1	Desain Semesta	9		Open
1.4	Dokumen Desain Aplikasi	37		Open
1.4.1	Desain Antarmuka Pengguna Aplikasi Entri Web	25		Open
1.4.2	Desain Antarmuka Pengguna Aplikasi Entri Web <i>Sign-off</i>	1		Open
1.4.3	Formulir Entri Web dan Validasi Model Basis Data	11		Open
1.5	Dokumen Kebutuhan Fungsional	9		Open
1.5.1	Desain Aplikasi	9		Open
1.5.1.1	Otentifikasi Pengguna	4		Open
1.5.1.2	<i>Call Logging</i>	2		Open
1.5.1.3	Pencarian	3		Open

Gambar 2.19  
Struktur Rincian Pekerjaan

Gambar 2.19 menunjukkan contoh sebagian dari struktur rincian kerja untuk fase desain proyek pengembangan *data warehouse*. Setiap tugas utama berfokus pada salah satu hasil desain yang diperlukan. Di dalam setiap tugas, terdapat subtugas yang merinci aktivitas yang diperlukan untuk menyelesaikan tugas utama.

Jumlah tugas dan detail tingkat tergantung pada kompleksitas dan ukuran proyek. Semakin besar proyek, semakin penting untuk mendefinisikan tugas secara rinci sehingga tidak ada langkah-langkah penting yang terabaikan.

## 2. Rencana Kerja Proyek

Rencana kerja proyek adalah mekanisme yang digunakan untuk mengelola tugas-tugas yang tercantum dalam struktur rincian kerja. Rencana kerja ini merupakan alat utama bagi manajer proyek untuk mengelola proyek. Dengan menggunakan rencana kerja, manajer proyek dapat mengetahui apakah proyek itu lebih cepat atau lebih lambat dari jadwal, seberapa baik perkiraan proyek, dan perubahan apa yang perlu dilakukan untuk memenuhi tenggat waktu proyek.

### Contoh

ID Tugas	Nama Tugas	Perkiraan			Aktual			Ketergantungan	Status
		Durasi (Hari)	Tanggal Mulai	Tanggal Selesai	Tanggal Mulai	Tanggal Selesai	Durasi Varians		
1	Fase Desain	82						Buka	
1.1	Pengembangan Dokumen Desain Basis Data	18							
1.1.1	Staging Desain Basis Data	9							
1.1.2	Suspense Desain Basis Data	9							
1.2	Pengembangan Dokumen Desain rejects-handling	9						1.1.1, 1.1.2	
1.2.1	Desain rejects-handling Mesin	9							
1.3	Pengembangan Desain Dokumen OLAP	9						1.1.1, 1.1.2	
1.3.1	Desain Semesta	9							
1.4	Dokumen Desain Aplikasi	37							
1.4.1	Desain Antarmuka Pengguna Aplikasi Entri Web	25							
1.4.2	Desain Antarmuka Pengguna Aplikasi Entri Web Sign-off	1							
1.4.3	Formulir Entri Web dan Validasi Model Basis Data	11							
1.5	Dokumen Kebutuhan Fungsional	9							
1.5.1	Desain Aplikasi Otentifikasi Pengguna	9							
1.5.1.1	Call Logging	2							
1.5.1.3	Pencarian	3							

Gambar 2.20  
Format Rencana Kerja Proyek

Pada dasarnya, rencana kerja adalah gambar yang mencantumkan semua tugas dalam struktur rincian kerja, bersama dengan informasi tugas penting seperti orang-orang yang ditugaskan untuk melakukan tugas-tugas, waktu aktual yang dipergunakan untuk setiap tugas, dan perbedaan antara perkiraan dan waktu penyelesaian aktual (Gambar 2.20).

Minimal informasi yang tercantum mencakup durasi tugas, status tugas saat ini (misalnya: terbuka atau selesai), dan ketergantungan antar tugas, yaitu ketika satu tugas tidak dapat dilakukan sampai tugas lainnya selesai. Contoh pada Gambar 2.20 menunjukkan bahwa tugas 1.2 dan tugas 1.3 tidak dapat dimulai sebelum tugas 1.1 selesai. Peristiwa penting, atau tanggal penting, juga diidentifikasi pada rencana kerja. Presentasi kepada komite persetujuan, dimulainya pelatihan pengguna akhir, dan tanggal jatuh tempo dari *prototype* sistem adalah contoh jenis peristiwa yang mungkin penting untuk ditelusuri.



### Latihan

Untuk memperdalam pemahaman Anda mengenai materi di atas, kerjakanlah latihan berikut!

- 1) Jelaskan prinsip kerja, keunggulan, dan keterbatasan metodologi pengembangan sistem berikut ini:
  - a) Model Waterfall
  - b) Model Parallel
  - c) Model Iterative
  - d) Model System Prototyping
  - e) Model Extreme Programming
- 2) Jelaskan faktor penentu dalam memilih metodologi pengembangan sistem!
- 3) Jelaskan langkah utama dalam menyusun rencana kerja proyek!

#### *Petunjuk Jawaban Latihan*

- 1) Sistem kerja, keunggulan dan keterbatasan metodologi pengembangan sistem:
  - a) Model *Waterfall*  
Tahapan kegiatan dilakukan secara berurutan dari satu fase ke fase yang lain. Setelah pekerjaan yang dihasilkan dalam satu fase disetujui, fase berakhir, dan fase berikutnya dimulai.

*Waterfall* dapat menangani sistem yang kompleks, namun tidak disarankan untuk digunakan ketika waktu menjadi sesuatu yang sangat berharga, karena tidak memiliki kemampuan untuk menyediakan sistem atau *prototype* ke tangan pengguna lebih awal. *Waterfall* tidak memberi peluang dengan mudah untuk melakukan perubahan-perubahan kebutuhan sepanjang fase SDLC.

b) Model *Parallel*

Pada tahap awal diciptakan desain secara umum untuk keseluruhan sistem. Kemudian proyek dibagi menjadi serangkaian sub proyek yang dapat dirancang dan diimplementasikan secara paralel. Setelah semua sub proyek selesai, dilakukan integrasi akhir dari bagian-bagian yang terpisah, dan selanjutnya sistem didistribusikan.

Dengan sistem kerja yang paralel, model *Parallel* dapat mempercepat proses untuk menghasilkan sistem. Namun demikian akan cukup rumit mengintegrasikan sub-sub proyek jika sub proyek tidak didesain dengan cermat untuk dapat berdiri sendiri sehingga hasil desain dalam satu sub proyek dapat mempengaruhi yang lain

c) Model *Iterative*

Sistem iteratif memecah satu proyek menjadi beberapa bagian untuk dikembangkan secara berurutan. Kebutuhan paling penting dan mendasar digabungkan ke dalam versi pertama sistem. Versi pertama ini dikembangkan dengan cepat melalui SDLC mini, dan sekaligus diimplementasikan. Pengguna dapat memberikan umpan balik untuk dimasukkan ke dalam versi sistem selanjutnya.

Kelebihan sistem pengembangan secara berulang adalah pengguna mendapatkan versi awal sistem dengan cepat sehingga nilai bisnis dari proyek dapat langsung disediakan. Kelemahan utama dari model pengembangan iteratif, pengguna sering tidak menyadari bahwa pada versi awal hanya diperuntukkan memenuhi kebutuhan paling kritis dari sistem yang akan tersedia, sehingga pengguna terus menuntut untuk menggunakan versi yang sempurna.

d) Model *System Prototyping*

Model *System Prototyping* melakukan fase analisis, desain, dan implementasi secara keseluruhan (tuntas) untuk menghasilkan versi sederhana dari sistem yang diusulkan, dan mendistribusikan kepada pengguna untuk mendapatkan umpan balik. Masukan dari pengguna digunakan untuk melakukan analisis ulang, mendesain ulang, dan mengimplementasikan kembali *prototype* kedua yang mengoreksi kekurangan *prototype* pertama dan menambahkan lebih banyak fitur. Siklus ini berlanjut hingga analis, pengguna, dan sponsor proyek menyepakati bahwa *prototype* telah menyediakan fungsionalitas yang cukup untuk dipasang dan digunakan dalam organisasi.

Kelebihan *System Prototyping* adalah dalam situasi di mana pengguna memiliki kesulitan mengungkapkan keseluruhan kebutuhan pada fase awal pengembangan sistem, pengembang masih dapat dengan cepat menyediakan sistem. Namun, kekurangannya adalah kurangnya analisis metodis yang cermat sebelum membuat keputusan melanjutkan ke tahap desain dan implementasi, sehingga sistem yang dihasilkan mungkin memiliki beberapa keterbatasan pada rancangan sistem yang paling mendasar, sehingga sistem belum dapat difungsikan secara maksimal.

e) Model *Extreme Programming*

Proyek *Extreme Programming* (XP) dimulai dengan pengguna menggambarkan apa yang perlu dilakukan sistem. Kemudian, programmer secara berkelompok mengembangkan kode program dalam modul kecil dan sederhana, dan mengujinya untuk memenuhi kebutuhan pengguna.

Kelebihan model XP adalah siklus pengembangan yang sangat pendek sehingga dapat memberikan hasil lebih cepat, bahkan lebih cepat daripada pendekatan RAD. Namun demikian, XP memerlukan keterlibatan pengguna sistem secara aktif untuk memberikan masukan-masukan sepanjang pengembangan sistem. XP juga tidak cocok untuk proyek-proyek yang kompleks.

- 2) Faktor-faktor penentu dalam memilih metodologi pengembangan sistem
  - a) Kejelasan Kebutuhan Pengguna: yaitu mengenai apa yang harus dilakukan sistem. Jika kebutuhan pengguna tidak jelas, sulit bagi analis untuk memahaminya ketika sedang terlibat dalam perbincangan dengan user, sehingga juga akan sulit menyajikannya dalam laporan tertulis.
  - b) Keakraban dengan Teknologi: analis mesti mengenal dengan baik teknologi yang akan digunakan dalam pengembangan sistem. Jika tidak, analis dapat menggunakan model pengembangan tertentu yang memiliki kemampuan untuk menyelidiki teknologi secara mendalam sebelum desain diselesaikan secara lengkap.
  - c) Kompleksitas Sistem: sistem yang kompleks membutuhkan metodologi pengembangan yang cermat dan terperinci.
  - d) Keandalan Sistem: ketika keandalan menjadi hal yang penting, analis perlu memilih model pengembangan sistem yang penekanannya pada fase analisis dan fase desain yang rinci, serta pengujian yang mendalam.
  - e) Kerangka Waktu: proyek-proyek yang memiliki jadwal waktu singkat harus menggunakan metodologi-metodologi pengembangan yang menekankan pada pengembangan secara cepat, walaupun hasil pengembangan belum maksimal, yang terpenting segera digunakan oleh

pengguna. Sebaliknya, proyek-proyek yang memiliki jadwal waktu yang relatif panjang, menggunakan model pengembangan yang menekankan pada hasil yang rinci dan sempurna.

- f) Visibilitas Jadwal: yaitu memperkirakan dan memastikan apakah suatu proyek akan berjalan sesuai jadwal. Untuk menanggulangi ketidakpastian, diperlukan metodologi pengembangan yang dapat menempatkan fase keputusan desain yang kritis ke fase lebih awal dalam proyek, untuk membantu manajer proyek mengenali dan mengatasi faktor risiko, serta menjaga agar ekspektasi tetap terjaga.
- 3) Langkah utama dalam menyusun rencana kerja proyek:
- a) Mengidentifikasi Tugas (pekerjaan)  
Tugas-tugas atau pekerjaan yang perlu diselesaikan untuk mencapai tujuan dalam proyek didefinisikan terlebih dahulu, kemudian dipecah menjadi beberapa sub tugas yang merinci aktivitas yang diperlukan untuk menyelesaikan tugas utama.
  - b) Menyusun Rencana Kerja Proyek  
Rencana kerja disusun dengan cara menambahkan rincian aktivitas (tugas-tugas) bersama dengan informasi penting lainnya (seperti personel yang ditugaskan, waktu aktual yang dipergunakan, dan perbedaan antara perkiraan dan waktu aktual) ke dalam tabel rencana kerja.



## Rangkuman

---

Faktor penentu keberhasilan manajemen proyek adalah mulai dengan menilai secara realistik pekerjaan yang perlu diselesaikan dan kemudian mengelola proyek sesuai dengan rencana. Ini dapat dicapai dengan mengikuti beberapa langkah dasar manajemen proyek, berupa: pemilihan metodologi pengembangan sistem yang sesuai dengan karakteristik proyek, membuat perkiraan jangka waktu berdasarkan ukuran sistem yang dikembangkan, serta membuat daftar penugasan.

Terdapat beberapa metodologi yang merupakan pendekatan formal yang dapat digunakan dalam manajemen pengembangan sistem informasi. Metodologi tersebut merupakan standar formal yang digunakan oleh lembaga pemerintah, sementara yang lain telah dikembangkan oleh perusahaan konsultan untuk dijual kepada klien. Banyak organisasi juga menggunakan metodologi internal mereka sendiri yang telah didefinisikan selama bertahun-tahun. Siklus hidup pengembangan sistem (*System Development Life Cycle/SDLC*) menyediakan landasan bagi metodologi-metodologi tersebut dalam pengembangan sistem informasi.

Beberapa metodologi utama adalah *Waterfall* dan variasinya berupa model *Parallel* dan *V-model*; *Rapid Application Development (RAD)* yang terdiri atas model *Iterative*, *System Prototyping* dan *Throwaway Prototyping*; dan model *Agile* termasuk

*Extreme Programming*. Manajer proyek mengevaluasi karakteristik proyek melalui beberapa faktor, seperti kejelasan kebutuhan pengguna, keakraban dengan teknologi, kompleksitas, keandalan, kerangka waktu, dan visibilitas jadwal, untuk memilih metodologi yang paling tepat digunakan dalam pengembangan proyek.

Manajer proyek kemudian memperkirakan kerangka waktu untuk proyek tersebut. Pengalaman masa lalu, standar industri, dan teknik seperti analisis *function-point*, memberikan bantuan dalam tugas ini. Metodologi proyek menyediakan daftar tugas dan sasaran proyek, yang diubah oleh manajer proyek, tergantung pada kebutuhan proyek tertentu. Untuk membuat rencana kerja, manajer proyek merevisi tugas-tugas menjadi struktur rincian kerja, perkiraan waktu penugasan, serta informasi lainnya dimasukkan ke dalam rencana kerja.



## Tes Formatif 2

---

Pilihlah satu jawaban yang paling tepat!

- 1) Varian model *Waterfall* yang proses pengembangannya menekankan pada aspek efektivitas pengujian sistem pada berbagai tingkatan fase adalah ....
  - A. Model *Parallel*
  - B. V-Model
  - C. *Extreme Programming*
  - D. Jawaban A dan B benar
- 2) Metodologi pengembangan sistem berbasis *Rapid Application Development* (RAD) memiliki beberapa varian, *kecuali* ....
  - A. *System Development Life Cycle*
  - B. *Iterative Development*
  - C. *System Prototyping*
  - D. *Throwaway Prototyping*
- 3) Model proses pengembangan sistem yang menekankan pada penggunaan *prototype* untuk mengeksplorasi alternatif desain adalah ....
  - A. Model Dijkstra
  - B. Model *System Prototyping*
  - C. Model *Throwaway Prototyping*
  - D. semua jawaban A, B, dan C salah
- 4) Perhatikan karakteristik metodologi pengembangan sistem berikut:
  - mengikuti seluruh fase SDLC sebelum beriterasi
  - siklus pengembangan yang sangat pendek, sehingga dapat memberikan hasil yang cepat
  - memerlukan keterlibatan pengguna sistem secara aktif untuk memberikan masukan sepanjang siklus pengembangan
  - tidak cocok untuk proyek-proyek yang kompleks

Karakteristik tersebut dimiliki oleh model pengembangan ....

- A. *Waterfall*
  - B. *Iterative*
  - C. *Extreme Programming*
  - D. Semua jawaban A, B, dan C salah
- 5) Beberapa cara yang dapat digunakan untuk memperkirakan waktu yang dibutuhkan untuk mengembangkan sistem adalah, *kecuali* ....
- A. menggunakan paket perangkat lunak estimasi
  - B. menggunakan pendekatan ROI (*Return on Investment*)
  - C. merujuk pada pengalaman masa lalu
  - D. menggunakan pendekatan *function-point*
- 6) Pada prinsipnya konsep dasar model *Waterfall* serupa dengan model *Extreme Programming*, yaitu iterasi terjadi setelah seluruh fase telah dilalui. Namun demikian, model *Waterfall* melalui setiap fase dengan cara mengembangkan modul kecil dan sederhana (tidak secara detail) untuk selanjutnya dilakukan pengujian, sehingga proses pengembangan menjadi cepat dan sangat fleksibel.
- A. Benar
  - B. Salah
- 7) Model *System Prototyping* tidak menghasilkan *prototype* yang akan berevolusi menjadi sistem akhir, melainkan hanya menghasilkan *prototype* desain yang bersifat sementara (hanya digunakan pada fase analisis dan desain sistem)
- A. Benar
  - B. Salah
- 8) Model *Iterative* memecah satu proyek menjadi beberapa bagian proyek untuk dikembangkan secara berurutan, namun tetap mengikuti siklus SDLC untuk setiap bagian proyek sebelum beriterasi, sedangkan model *Waterfall* mengimplementasikan seluruh bagian proyek secara utuh mengikuti siklus SDLC sebelum beriterasi.
- A. Benar
  - B. Salah
- 9) *Throwaway Prototyping* umumnya tidak menjadi pilihan untuk mengembangkan sistem yang mementingkan faktor keandalan, karena kurang hati-hati pada fase analisis dan fase desain yang justru sangat penting untuk sistem yang mementingkan keandalan.
- A. Benar
  - B. Salah

- 10) Keberhasilan model *Extreme Programming* dapat dicapai hanya jika diterapkan pada proyek skala kecil dengan tim pengembang dalam kelompok kecil.
- A. Benar
  - B. Salah

Cocokkanlah jawaban Anda dengan Kunci Jawaban Tes Formatif 2 yang terdapat di bagian akhir modul ini. Hitunglah jawaban yang benar. Kemudian, gunakan rumus berikut untuk mengetahui tingkat penguasaan Anda terhadap materi Kegiatan Belajar 2.

$$\text{Tingkat Penguasaan} = \frac{\text{Jumlah Jawaban yang Benar}}{\text{Jumlah Soal}} \times 100$$

Arti tingkat penguasaan

<70%	70% - 79%	80% - 89%	90% - 100%
kurang	cukup	baik	baik sekali

Apabila mencapai tingkat penguasaan 80% atau lebih, Anda dapat meneruskan dengan modul selanjutnya. **Bagus!** Jika masih di bawah 80%, Anda harus mengulangi materi Kegiatan Belajar 2, terutama bagian yang belum dikuasai.

## Kunci Jawaban Tes Formatif

### *Tes Formatif 1*

- 1) D
- 2) A
- 3) D
- 4) D
- 5) D
- 6) B
- 7) B
- 8) B
- 9) A
- 10) A

### *Tes Formatif 2*

- 1) B
- 2) A
- 3) C
- 4) C
- 5) B
- 6) B
- 7) A
- 8) A
- 9) B
- 10) A

## Glosarium

Agile development	: Sekelompok metodologi pemrograman yang fokus pada penyederhanaan SDLC
Break-Even Point (BEP)	: Perhitungan untuk menentukan jumlah tahun yang dibutuhkan suatu perusahaan untuk mengembalikan investasi awalnya
Cash Flow	: Arus kas Informasi penerimaan dan pengeluaran kas dalam sebuah perusahaan pada periode waktu tertentu
Champion	: Eksekutif tingkat tinggi yang mensponsori pengembangan proyek sistem
Cost-benefit analysis	: Analisis biaya-manfaat
Costs and Benefits	: Biaya dan manfaat
Customer Relationship Management (CRM)	: Program aplikasi yang berfokus pada aktifitas perusahaan dengan pelanggan
End-user	: Pengguna akhir sistem (orang yang akan mengoperasikan sistem)
Estimasi	: Melihat ke masa depan mengenai suatu ketidakpastian untuk mendefenisikan skenario kasus terbaik dan kasus terburuk
Extreme Programming (XP)	: Metodologi pengembangan sistem secara cepat yang mengedepankan konsep komunikasi, kerja secara kelompok, kesederhanaan, umpan balik, dan keberanian untuk memberikan kepuasan kepada pengguna sistem
Familier	: Keakraban dengan teknologi
Fitur	: Modul-modul fungsional yang terdapat dalam aplikasi sistem informasi

Information Technology (IT)	: Teknologi Informasi (TI)
Intangible benefits	: Manfaat (nilai) yang tidak berwujud
Intangible costs	: Biaya yang tidak berwujud
Iterative Development	: Metodologi pengembangan sistem yang bekerja dengan cara memecah satu proyek menjadi beberapa bagian untuk dikembangkan secara berurutan
Net Present Value (NPV)	: Perhitungan untuk menentukan perbedaan antara total nilai sekarang dari manfaat dengan total nilai sekarang dari biaya
Present Value (PV)	: Perhitungan untuk menentukan nilai uang saat ini didasarkan pada hasil di masa mendatang
Project's Economic Value	: Nilai ekonomis proyek
Prototype	: Versi sederhana yang belum memiliki fitur lengkap dari sistem yang dikembangkan
Rapid Application Development (RAD)	: Kumpulan metodologi pengembangan sistem yang didesain untuk mempercepat fase SDLC guna mendapatkan sebagian dari sistem yang dikembangkan dengan cepat tersebut ke tangan pengguna untuk dievaluasi dan mendapatkan umpan balik.
Requirement	: Kebutuhan
Return on Investment (ROI)	: Perhitungan yang mengukur tingkat pengembalian rata-rata yang diperoleh dari uang yang diinvestasikan dalam proyek
Stakeholder	: Pemangku kepentingan yang terdiri atas pengagas proyek, manajemen organisasi, dan pengguna sistem yang dikembangkan
Studi Kelayakan	: Studi yang dilakukan untuk menentukan apakah suatu proyek layak untuk direalisasikan atau tidak

- System Development Life Cycle (SDLC) : Siklus hidup pengembangan sistem sebuah siklus dalam proses pengembangan sistem informasi, yang terdiri dari tahapan merencanakan, membuat, menguji, dan meluncurkan sistem informasi
- System Prototype : Metodologi pengembangan sistem yang bekerja menghasilkan prototype sebagai cikal bakal sistem akhir
- Tangible Benefits : Manfaat (nilai) yang berwujud/nyata
- Tangible items : Barang-barang yang berwujud/nyata
- Throwaway Prototyping : Metodologi pengembangan sistem yang bekerja hanya untuk menghasilkan Prototype Desain selama fase analisis dan fase desain sistem
- V-Model : Fase-fase metodologi pengembangan sistem yang menyerupai kurva V
- Waterfall : Air terjun  
Fase-fase metodologi pengembangan sistem yang bergerak maju dengan cara yang sama seperti air terjun

## Daftar Pustaka

- Dennis, A., Wixom, B. H., & Roth, R.M. (2012). *Systems analysis & design*. 5th Edition. United States of America: John Wiley & Sons, Inc.
- Fowler, M. (2005). *UML distilled*, Edisi 3. Yogyakarta: ANDI.
- Jogiyanto, H.M. (2008). *Analisis & desain sistem informasi: pendekatan terstruktur teori dan praktik aplikasi bisnis*. Edisi 3. Yogyakarta: ANDI.
- Pressman, R.S., & Maxim, B.R. (2014). *Software engineering: a practitioner's approach*, 8th Edition. New York: McGraw-Hill Education.
- Sommerville, I. (2011). *Software engineering*. 9th Edition. USA: Pearson.
- Tantra, R. (2012). *Manajemen proyek sistem informasi*. Yogyakarta: ANDI.

**MSIM4302**  
**Edisi 1**

**MODUL 03**  
**Analisis Sistem**

Bahar, S.T., M.Kom.

## Daftar Isi

Modul 03	
Analisis Sistem	
<b>Kegiatan Belajar 1</b>	3.4
Penentuan Kebutuhan (Persyaratan)	
<b>Latihan</b>	3.18
<b>Rangkuman</b>	3.19
<b>Tes Formatif 1</b>	3.20
<b>Kegiatan Belajar 2</b>	3.23
Teknik Elisitasi dan Strategi Analisis Kebutuhan	
<b>Latihan</b>	3.47
<b>Rangkuman</b>	3.49
<b>Tes Formatif 2</b>	3.50
<b>Kunci Jawaban Tes Formatif</b>	3.53
<b>Glosarium</b>	3.54
<b>Daftar Pustaka</b>	3.56



## Pendahuluan

Dalam konteks *System Development Life Cycle* (SDLC), output dari fase perencanaan proyek (studi kelayakan dan perencanaan proyek sistem) adalah menentukan ruang lingkup proyek, menilai kelayakan proyek, dan memberikan rencana kerja awal. Hasil tahap perencanaan ini merupakan input utama fase analisis.

Selama fase analisis, analis menentukan kebutuhan fungsional untuk sistem baru. Modul ini dimulai dengan menjelaskan fase analisis dan hasil utamanya. Selanjutnya menjelaskan konsep kebutuhan dan mendefinisikan beberapa kategori kebutuhan, menjelaskan struktur pernyataan definisi kebutuhan. Bagian ini juga membahas teknik untuk mendapatkan kebutuhan seperti wawancara, sesi JAD (*Joint Application Development*), kuesioner, analisis dokumen, dan observasi. Pada bagian akhir menjelaskan beberapa strategi analisis kebutuhan untuk membantu analis mengidentifikasi kebutuhan.

Setelah mempelajari modul ini mahasiswa diharapkan secara lebih rinci mampu menjelaskan:

1. tujuan analisis dan hasil utama yang akan dicapai;
2. konsep dan kategori kebutuhan;
3. proses penentuan kebutuhan;
4. cara menyusun struktur pernyataan definisi kebutuhan;
5. berbagai teknik validasi kebutuhan;
6. tentang elitisasi dan berbagai teknik elitisasi kebutuhan;
7. strategi analisis kebutuhan.

## Penentuan Kebutuhan (Persyaratan)

Penentuan kebutuhan sistem dilakukan dengan cara transformasi kebutuhan bisnis sistem menjadi daftar yang lebih terperinci mengenai apa yang harus dilakukan sistem baru untuk memberikan nilai yang dibutuhkan bagi bisnis. Daftar kebutuhan yang terperinci ini didukung, dikonfirmasi, dan diklarifikasi oleh kegiatan lain dari fase analisis, yaitu kegiatan pengembangan model proses dan pengembangan model data (akan dibahas pada kegiatan belajarnya).

### A. FASE ANALISIS

Fase analisis dapat didefinisikan sebagai suatu kegiatan penguraian suatu sistem informasi yang utuh ke dalam bagian-bagian komponennya untuk mengidentifikasi dan mengevaluasi permasalahan-permasalahan, kesempatan-kesempatan, hambatan-hambatan yang terjadi, dan kebutuhan-kebutuhan yang diharapkan sehingga dapat diusulkan perbaikan-perbaikannya (Jogiyanto, 2008).

Pada intinya tujuan akhir dari fase analisis adalah mengidentifikasi kebutuhan-kebutuhan pengguna secara terperinci pada sistem baru yang akan dikembangkan. Analis bekerja sama dengan pengguna bisnis dari sistem baru untuk memahami kebutuhan mereka. Proses analisis sistem melibatkan tiga langkah utama (Dennis, 2012), yaitu:

1. memahami kerja sistem yang ada (sistem yang sedang berjalan);
2. mengidentifikasi perbaikan atau penyempurnaan yang diharapkan;
3. menentukan kebutuhan (persyaratan) untuk sistem baru (sistem yang akan datang).

Memahami kerja sistem yang ada dapat dilakukan dengan mempelajari secara terinci bagaimana sistem yang ada beroperasi. Hal paling utama yang perlu dipahami adalah mengidentifikasi masalah yang muncul dalam operasi sistem yang ada. Masalah inilah yang menyebabkan sasaran dari sistem tidak dapat dicapai. Untuk mempelajari operasi dari sistem ini diperlukan data yang dapat diperoleh dengan cara melakukan penelitian (mekanisme melakukan penelitian akan dibahas pada materi selanjutnya). Bila pada tahap perencanaan sistem juga pernah dilakukan penelitian untuk memperoleh data, penelitian ini sifatnya hanya sebagai penelitian pendahuluan (*preliminary survey*).

Pada tahap analisis sistem, penelitian yang dilakukan adalah penelitian terinci (*detailed survey*).

Ada kalanya langkah pertama (memahami sistem yang ada) tidak dilakukan atau hanya dilakukan secara singkat. Ini dilakukan dengan beberapa alasan: ketika belum ada sistem yang sedang berjalan saat ini, jika sistem dan proses yang ada saat ini tidak relevan dengan sistem di masa depan, atau jika tim proyek menggunakan metodologi pengembangan *Rapid Application Development* (RAD) atau *agile* yang memang tidak menekankan hal tersebut dilakukan. Metode tradisional seperti *waterfall* dan pengembangan *parallel* biasanya mensyaratkan waktu yang cukup untuk memahami sistem yang ada dan mengidentifikasi perbaikan-perbaikan yang diharapkan sebelum memulai mengidentifikasi kebutuhan untuk sistem yang akan datang. Metodologi RAD dan *agile* yang terbaru seperti pengembangan *iterative*, *system prototyping*, *throwaway*, dan *extreme programming*, berfokus secara eksklusif pada bagaimana melakukan perbaikan dan mengidentifikasi kebutuhan sistem yang akan datang, dan hanya mencurahkan sedikit waktu untuk memahami sistem yang ada saat ini. Pemahaman yang diperoleh dari proses memahami sistem yang ada saat ini bisa sangat berharga bagi tim pengembang proyek.

Keterampilan berpikir kritis adalah kemampuan untuk mengenali kekuatan dan kelemahan, serta menyusun kembali ide dalam bentuk yang lebih baik. Keterampilan ini diperlukan oleh analis agar analis dapat memahami masalah dan mengembangkan proses bisnis baru yang lebih baik, yang didukung oleh teknologi sistem informasi. Keterampilan ini sangat penting dalam menganalisis hasil penemuan kebutuhan sistem dan menerjemahkan kebutuhan tersebut menjadi sebuah konsep untuk sistem yang baru.

#### *Contoh Kasus:*

Misalkan dalam melakukan kajian kebutuhan sistem, pengguna menyatakan bahwa sistem baru harus "meniadakan persediaan cadangan". Meskipun ide ini mungkin merupakan tujuan proyek yang sangat layak dipertimbangkan untuk dilakukan, analis perlu memikirkannya secara kritis untuk merumuskan kebutuhan sistem yang berguna. Pertama-tama analis dapat membuat agar pengguna mencoba memikirkan tentang keadaan yang mengarah ke kehabisan stok (misalnya situasi ketika pesanan dari pemasok tidak tepat waktu sehingga terjadi keterlambatan). Dengan berfokus pada masalah ini, tim pengembang berada dalam posisi yang tepat untuk mengembangkan proses bisnis baru untuk mengatasi masalah ini. Kebutuhan baru akan didasarkan pada masalah yang benar-benar perlu diperbaiki. Dalam hal ini, kebutuhan untuk sistem yang baru mungkin termasuk di dalamnya adalah berikut ini.

1. Sistem akan memperbarui (*update*) informasi persediaan yang ada sebanyak dua kali sehari, dari yang sebelumnya hanya satu kali sehari.
2. Sistem akan menghasilkan pemberitahuan kehabisan stok segera ketika jumlah item yang ada telah mencapai titik pemesanan ulang item.

### 3.6 Analisis Sistem

3. Sistem harus menyertakan pemasok yang direkomendasikan pada setiap pemberitahuan persediaan habis.
4. Sistem harus mengirimkan pesanan pembelian untuk persediaan ke pemasok melalui sistem informasi yang aman.

Dari contoh kasus di atas, analis tidak dapat secara realistik berharap bahwa kebutuhan yang sebenarnya untuk sistem baru dapat dengan mudah dikumpulkan setelah melalui beberapa percakapan dengan para pemangku kepentingan. Analis harus siap memahami segala situasi untuk menemukan kebutuhan sistem yang sebenarnya, dan ini seringkali bukanlah suatu proses yang mudah.

Hasil akhir tahap analisis adalah proposal sistem yang memuat pernyataan kebutuhan sistem secara terperinci, model proses, dan model data bersama dengan analisis kelayakan dan rencana kerja yang telah direvisi. Pada akhir fase analisis, proposal sistem disampaikan kepada komite persetujuan, biasanya dalam bentuk penjelasan secara menyeluruh dari sistem. Tujuannya adalah untuk menjelaskan sistem secara terperinci sehingga pengguna, manajer, dan pengambil keputusan utama dapat memahaminya, dapat mengidentifikasi perubahan yang diperlukan, dan dapat membuat keputusan tentang apakah proyek harus dilanjutkan ke fase selanjutnya. Sebelum berpindah ke fase desain, proyek harus ditinjau untuk memastikan terus memberikan kontribusi nilai bisnis kepada organisasi. Jika disetujui, proposal sistem (berisi definisi kebutuhan, model proses sistem, dan model data) digunakan sebagai acuan (masukan) untuk langkah-langkah dalam fase desain, yang memperjelas dan menentukan secara lebih rinci bagaimana sebuah sistem akan dibangun.

Garis antara fase analisis dan fase desain sangat samar, karena hasil akhir yang dibuat dalam fase analisis benar-benar merupakan langkah pertama dalam desain sistem baru. Banyak keputusan mengenai desain utama untuk sistem baru ditemukan dalam hasil analisis. Bahkan, beberapa ahli memberi nama lain yang lebih tepat untuk fase analisis sebagai "analisis dan desain awal", tetapi karena nama ini agak panjang dan karena sebagian besar organisasi hanya menyebut fase ini sebagai "analisis", maka pada pembahasan ini juga akan menggunakan istilah "analisis". Meskipun demikian, penting untuk diingat bahwa hasil dari fase analisis benar-benar merupakan langkah pertama dalam desain sistem baru.

Dalam banyak hal, menentukan kebutuhan sistem adalah satu-satunya aspek paling kritis dari seluruh rangkaian SDLC. Meskipun banyak faktor yang berkontribusi pada kegagalan proyek pengembangan sistem, kegagalan untuk menentukan kebutuhan sistemlah yang benar-benar menjadi penyebab utamanya (Mensah, 2003). Sebuah studi yang dilakukan oleh Mullaney pada tahun 2008 tentang proyek perangkat lunak perusahaan Fortune 500 menemukan hanya 37% responden survei merasa proyek memenuhi kebutuhan pengguna (Dennis, 2012). Oleh karena itu, analis harus mencurahkan banyak perhatian pada pekerjaan yang dilakukan dalam fase analisis. Di sinilah elemen utama sistem pertama kali mulai muncul. Jika kebutuhan sistem

kemudian ditemukan salah atau tidak lengkap, pengerajan ulang yang signifikan mungkin diperlukan, menambah waktu dan biaya yang sangat substansial untuk proyek.

Semasa fase penentuan kebutuhan sistem, konsep sistem yang akan datang masih memungkinkan dengan mudah dapat diubah karena belum banyak pekerjaan yang dilakukan. Namun ketika sistem telah berjalan melalui fase SDLC berikutnya (desain dan implementasi), akan sulit untuk kembali ke penentuan kebutuhan sistem dan membuat perubahan besar karena semua pengerajan ulang akan dilibatkan. Inilah sebabnya mengapa pendekatan *iterative* dari banyak metodologi RAD dan *agile* begitu efektif digunakan (sejumlah kecil kebutuhan sistem dapat diidentifikasi dan diimplementasikan dalam tahap-tahap tambahan, yang memungkinkan keseluruhan sistem berubah dan berkembang seiring waktu), demikian juga dengan metodologi seperti *V-model* yang menekankan pada pengujian sistem yang harus dilakukan pada saat yang sama ketika kebutuhan sistem sedang ditentukan. Dengan cara itu, pengujian bukan hanya proses yang berlangsung pada saat-saat terakhir sistem akan digunakan, akan tetapi juga langsung didasarkan pada kebutuhan sistem yang sedang didefinisikan.

## B. DEFINISI KEBUTUHAN (PERSYARATAN) DAN JENIS-JENIS KEBUTUHAN

Kebutuhan adalah pernyataan tentang apa yang harus dilakukan sistem atau karakteristik apa yang perlu dimiliki oleh sistem. Semasa projek pengembangan sistem berjalan, kebutuhan akan didefinisikan, yang menjelaskan apa yang sistem bisnis butuhkan (kebutuhan bisnis); apa yang perlu dilakukan pengguna (kebutuhan pengguna); apa yang harus dilakukan perangkat lunak (kebutuhan fungsional); karakteristik yang harus dimiliki sistem (kebutuhan nonfungsional); dan bagaimana sistem harus dibangun (kebutuhan sistem). Meskipun daftar jenis kebutuhan ini pada awalnya mungkin tampak "mengintimidasi", namun kategori kebutuhan tersebut hanya mencerminkan tahapan dalam SDLC di mana kebutuhan-kebutuhan didefinisikan.

Dalam permintaan sistem pada fase perencanaan SDLC, tercantum pernyataan yang menjelaskan mengenai alasan untuk mengajukan projek pengembangan sistem. Pernyataan-pernyataan ini mencerminkan **kebutuhan bisnis** bahwa jika dibangun, sistem ini akan memenuhi kebutuhan. Kebutuhan bisnis ini membantu menentukan tujuan keseluruhan sistem dan membantu memperjelas kontribusi yang akan dihasilkannya bagi keberhasilan organisasi. Contoh kebutuhan bisnis meliputi:

1. meningkatkan pangsa pasar;
2. mempersingkat waktu pemrosesan pesanan;
3. mengurangi biaya layanan pelanggan;
4. mengurangi biaya inventaris;
5. meningkatkan respons terhadap permintaan layanan pelanggan.

Ketika proyek pengembangan sistem selesai, kesuksesan akan diukur dengan mengevaluasi apakah kebutuhan bisnis yang disebutkan telah benar-benar tercapai.

Semasa fase analisis, kebutuhan ditulis dari perspektif bisnis, dan kebutuhan-kebutuhan tersebut berfokus pada apa yang perlu dilakukan sistem untuk memenuhi kebutuhan pengguna bisnis. Tempat permulaan yang baik adalah berkonsentrasi pada apa yang sebenarnya harus diselesaikan pengguna dengan sistem yang ada untuk memenuhi pekerjaan atau tugas yang diperlukan. **Kebutuhan pengguna** ini menjelaskan hal-hal yang dilakukan pengguna sebagai bagian integral dari operasi bisnis. Contoh kebutuhan pengguna meliputi:

1. menjadwalkan pertemuan dengan klien;
2. menempatkan pesanan baru pelanggan;
3. *reorder* inventaris.

Menentukan cara di mana sistem baru dapat mendukung kebutuhan pengguna adalah kegiatan yang mengarah pada pernyataan **kebutuhan fungsional** sistem. Kebutuhan fungsional berkaitan langsung dengan proses yang harus dilakukan sistem sebagai bagian dari mendukung tugas pengguna, atau informasi yang perlu disediakannya saat pengguna melakukan suatu tugas. Institut Internasional Analisis Bisnis (IIBA) mendefinisikan kebutuhan fungsional sebagai "kemampuan suatu produk, atau hal-hal yang harus dilakukan oleh suatu produk untuk mendukung penggunanya" (Dennis, 2012), sedangkan Sommerville (2016) mendefinisikan kebutuhan fungsional sebagai pernyataan layanan yang harus diberikan sistem (produk perangkat lunak), bagaimana sistem harus bereaksi terhadap *input* tertentu, dan bagaimana sistem harus berlaku pada situasi-situasi tertentu. Pada beberapa kasus, kebutuhan fungsional juga dapat menyatakan secara eksplisit mengenai apa yang tidak boleh dilakukan sistem. Berdasarkan definisi-definisi tersebut, dapat disimpulkan bahwa kebutuhan fungsional menentukan bagaimana sistem akan mendukung pengguna dalam menyelesaikan tugas.

*Contoh:*

Misalkan kebutuhan pengguna adalah "menjadwalkan pertemuan dengan klien". Kebutuhan fungsional yang terkait dengan tugas itu adalah:

1. memastikan kesiapan klien;
2. menentukan tema pembicaraan yang sesuai dengan kesiapan klien;
3. memilih tempat pertemuan yang diinginkan;
4. mempersiapkan proses perekaman pembicaraan;
5. mengkonfirmasi janji pertemuan.

Bila diperhatian, kebutuhan fungsional ini menjadi berkembang pada tugas pengguna untuk menggambarkan kemampuan dan fungsi yang perlu dimasukkan ke sistem, agar memungkinkan pengguna menyelesaikan tugas. Pada saat analis bekerja bersama dengan pengguna bisnis sistem untuk mengidentifikasi kebutuhan pengguna

dan kebutuhan fungsional, pengguna dapat mengungkapkan proses yang diperlukan atau informasi yang dibutuhkan. Misalnya, seperti yang ditunjukkan pada Gambar 3.1, pengguna dapat menyatakan "Sistem harus dapat menyimpan riwayat pesanan pelanggan selama tiga tahun" (kebutuhan informasi). Analis harus menyelidiki alasan dibalik pernyataan ini, seperti: "Sistem harus memungkinkan pelanggan yang terdaftar untuk meninjau riwayat pesanan mereka sendiri selama tiga tahun terakhir" (suatu kebutuhan proses). Demikian pula, pengguna dapat menyatakan "Sistem harus memeriksa pesanan pelanggan yang masuk untuk mempersiapkan persediaan" (suatu kebutuhan proses). Analis yang tanggap akan mengenali kebutuhan informasi terkait, "Sistem harus menjaga tingkat persediaan secara *real time* di semua gudang". Semua kebutuhan ini diperlukan untuk dapat sepenuhnya memahami sistem yang sedang dikembangkan.

KEBUTUHAN FUNGSIONAL	DESKRIPSI	CONTOH
BERORIENTASI PADA PROSES	Sebuah proses yang harus dilakukan sistem	<ul style="list-style-type: none"> <li>- Sistem harus memungkinkan pelanggan yang terdaftar untuk meninjau riwayat pesanan mereka sendiri selama tiga tahun terakhir</li> <li>- Sistem harus memeriksa pesanan pelanggan yang masuk untuk mempersiapkan persediaan.</li> <li>- Sistem memungkinkan siswa untuk dapat melihat jadwal kursus saat mendaftar untuk mengikuti suatu kelas tertentu</li> </ul>
BERORIENTASI PADA INFORMASI	Informasi yang harus terkandung dalam sistem	<ul style="list-style-type: none"> <li>- Sistem harus dapat menyimpan riwayat pesanan pelanggan selama tiga tahun</li> <li>- Sistem harus menjaga tingkat persediaan secara <i>real time</i> di semua gudang</li> <li>- Sistem harus mencakup jumlah penjualan dan pengeluaran aktual yang dianggarkan untuk tahun berjalan dan tiga tahun sebelumnya</li> </ul>

Gambar 3.1  
Contoh Kebutuhan Fungsional

Model proses (akan dibahas pada modul mendatang) digunakan untuk menjelaskan hubungan fungsi/proses dengan pengguna sistem, bagaimana suatu proses saling berhubungan, bagaimana data dimasukkan dan diproduksi oleh suatu proses, dan bagaimana fungsi suatu proses membuat dan menggunakan data yang tersimpan. Model proses membantu memperjelas komponen perangkat lunak yang akan digunakan untuk memenuhi kebutuhan fungsional. Selain itu, kebutuhan fungsional mulai mendefinisikan data yang harus dilacak untuk menyelesaikan tugas-tugas pengguna. Komponen data dari sistem didefinisikan dalam model data (akan dibahas mendatang).

Kebutuhan pengguna dan kebutuhan fungsional yang ditentukan dalam fase analisis akan mengalir ke fase desain, di mana kebutuhan-kebutuhan tersebut berkembang menjadi lebih teknis, menggambarkan bagaimana sistem akan diimplementasikan. Kebutuhan dalam fase desain mencerminkan perspektif pengembang, dan biasanya disebut **kebutuhan sistem**. Kebutuhan ini berfokus menjelaskan cara membuat produk perangkat lunak yang akan dihasilkan dari proyek. Spesifikasi kebutuhan sistem mencakup model-model sistem yang berbeda seperti model objek atau model aliran data. Jadi pada prinsipnya, kebutuhan sistem harus menyatakan apa yang harus dilakukan sistem dan bukan bagaimana sistem tersebut harus diimplementasikan.

Mungkin sedikit sulit untuk membedakan setiap kategori kebutuhan ini (Dennis, 2012). Ini sejalan dengan pernyataan Sommerville (2016) bahwa perbedaan antara jenis kebutuhan yang berbeda ini seringkali tidak jelas. Kebutuhan user yang berhubungan dengan keamanan, katakanlah bisa tampak sebagai kebutuhan nonfungisional. Namun demikian, ketika dikembangkan dengan lebih rinci, kebutuhan ini bisa menimbulkan kebutuhan lain berupa kebutuhan fungsional seperti kebutuhan untuk memasukkan fasilitas otorisasi pada sistem. Dengan demikian, walaupun pengklasifikasian kebutuhan dengan cara ini akan memudahkan dalam hal pembahasan, mesti tetap diingat bahwa ini sebenarnya hanya merupakan pembedaan yang artifisial. Yang paling membingungkan, beberapa pihak menggunakan istilah-istilah tersebut secara bergantian. Misalnya: kebutuhan sistem (perangkat lunak) sering diklasifikasikan sebagai kebutuhan fungsional dan nonfungisional. Yang terpenting diingat bahwa kebutuhan adalah pernyataan tentang apa yang harus dilakukan sistem, dan fokus kebutuhan akan berubah seiring waktu ketika proyek bergerak dari fase perencanaan ke fase analisis ke fase desain ke fase implementasi. Kebutuhan berevolusi dari pernyataan yang luas mengenai kebutuhan bisnis secara keseluruhan dari sistem ke pernyataan terperinci tentang kapabilitas bisnis yang harus didukung oleh sistem.

Kategori terakhir adalah **kebutuhan nonfungisional**. IIBA mendefinisikan kelompok kebutuhan ini sebagai “sifat kualitas (misalnya: keandalan, waktu tanggap), desain, kendala implementasi, antarmuka *input/output* (I/O) dan representasi data, serta faktor eksternal (misalnya: aturan keselamatan, undang-undang privasi) yang harus dimiliki oleh suatu produk”. Meskipun istilah “nonfungisional” tidak terlalu deskriptif, kategori kebutuhan ini mencakup sifat perilaku penting yang harus memiliki oleh

sistem, seperti kinerja dan kegunaan. Kemampuan untuk mengakses sistem melalui perangkat seluler adalah salah satu contoh kebutuhan nonfungsional.

KEBUTUHAN NONFUNGSIONAL	DESKRIPSI	CONTOH
OPERASIONAL	Lingkungan fisik dan teknis di mana sistem akan beroperasi	<ul style="list-style-type: none"> <li>- Sistem dapat berjalan pada perangkat genggam</li> <li>- Sistem harus dapat diintegrasikan dengan sistem persediaan yang ada</li> <li>- Sistem harus dapat bekerja pada browser Web apa pun.</li> </ul>
PERFORMA	Kecepatan, kapasitas, dan keandalan sistem	<ul style="list-style-type: none"> <li>- Interkoneksi apa pun antara pengguna dan sistem tidak boleh lebih dari 2 detik</li> <li>- Sistem mengunduh parameter status baru dalam tempo 5 menit setelah perubahan status.</li> <li>- Sistem harus tersedia untuk digunakan 24 jam per hari, 365 hari per tahun</li> <li>- Sistem mendukung 300 pengguna simultan dari 9-11 A.M.; 150 pengguna simultan di semua waktu lainnya</li> </ul>
KEAMANAN	Siapa dan dalam situasi apa dapat memiliki akses resmi ke sistem	<ul style="list-style-type: none"> <li>- Hanya manajer langsung yang dapat melihat catatan staf.</li> <li>- Pelanggan dapat melihat riwayat pesanan mereka hanya selama jam kerja.</li> <li>- Sistem ini mencakup semua perlindungan yang tersedia dari virus, worm, trojan horse, dll</li> </ul>
BUDAYA DAN POLITIK	Faktor budaya dan politik, kebijakan, dan persyaratan hukum yang mempengaruhi sistem	<ul style="list-style-type: none"> <li>- Sistem harus dapat membedakan antara mata uang dollar US dan mata uang dari negara lain</li> <li>- Kebijakan perusahaan adalah membeli komputer hanya dari perusahaan Dell.</li> <li>- Manajer area diizinkan untuk mengotorisasi antarmuka pengguna khusus dalam unit mereka.</li> <li>- Informasi pribadi dilindungi sesuai dengan Undang-Undang Perlindungan Data.</li> </ul>

Gambar 3.2  
Contoh Kebutuhan Nonfungsional

Istilah kebutuhan nonfungsional terutama digunakan dalam fase desain ketika membuat keputusan tentang model antarmuka pengguna, model perangkat keras dan perangkat lunak, dan model arsitektur yang mendasari sistem. Namun, banyak dari kebutuhan ini juga ditemukan semasa percakapan dengan pengguna dalam fase analisis, dan harus dicatat ketika kebutuhan-kebutuhan nonfungsional tersebut ditemukan.

Gambar 3.2 menyajikan berbagai jenis kebutuhan nonfungsional dan contoh masing-masing jenis. Perlu diperhatikan bahwa kebutuhan nonfungsional menggambarkan berbagai karakteristik sistem: operasional, kinerja, keamanan, dan budaya dan politik. Karakteristik ini tidak menggambarkan proses bisnis atau informasi, tetapi sangat penting dalam memahami seperti apa sistem akhir yang seharusnya. Sebagai contoh, tim proyek perlu mengetahui apakah suatu sistem harus sangat aman, memerlukan waktu respons yang sangat cepat (*subsecond*), harus mencapai basis pelanggan yang *multiplatform*. Kebutuhan ini akan memengaruhi keputusan desain yang akan dibuat dalam fase desain, terutama desain arsitektur. Jika metodologi yang digunakan mencakup pengembangan rencana pengujian selama analisis, maka kebutuhan ini akan menjadi penting dalam menetapkan tolok ukur pengujian yang akan dibutuhkan nantinya.

Banyak kebutuhan nonfungsional yang berhubungan dengan sistem sebagai satu kesatuan dan bukan dengan fitur sistem secara individu. Ini berarti bahwa kebutuhan tersebut bisa lebih kritis dari kebutuhan fungsional individu. Jika kegagalan memenuhi kebutuhan fungsional individu dapat mendegradasi sistem, maka kegagalan memenuhi kebutuhan sistem nonfungsional bisa membuat seluruh sistem tidak dapat dipakai. Sebagai contoh: jika sebuah sistem pesawat udara tidak memenuhi kebutuhan keandalannya, sistem tersebut tidak akan disertifikasi aman untuk beroperasi. Contoh lain: jika sebuah sistem kontrol *real-time* gagal memenuhi kebutuhan kinerjanya, fungsi kontrol tidak akan beroperasi dengan benar.

### C. PROSES PENENTUAN KEBUTUHAN

Penentuan kebutuhan selama fase analisis ditinjau dari sudut pandang bisnis maupun dari sudut pandang Teknologi Informasi. Analis sistem mungkin saja tidak memahami kebutuhan bisnis sebenarnya dari pengguna. Sebuah studi oleh *Standish Group* menemukan bahwa kurangnya keterlibatan pengguna dalam fase analisis kebutuhan adalah penyebab utama kegagalan proyek Teknologi Informasi (Dennis, 2012). Di sisi lain, pengguna bisnis mungkin tidak menyadari peluang yang mungkin ditawarkan oleh teknologi baru. Penting bagi tim analis untuk mempertimbangkan proses bisnis yang mendasarinya dan cara terbaik untuk mendukung proses bisnis tersebut dengan Teknologi Sistem Informasi.

**Contoh Kasus:**

Misalkan dianalogikan dengan kasus membangun rumah. Orang yang menempati sebuah rumah sudah barang tentu memiliki pemahaman yang mendalam tentang apa yang mereka inginkan di rumah tersebut. Namun, jika mereka diminta untuk mendesain rumah tersebut dari awal, itu akan menjadi tantangan karena mereka tidak memiliki keterampilan desain yang tepat dan keterampilan teknik untuk itu. Demikian juga, seorang arsitek yang bertindak sendiri tanpa berkomunikasi dengan pemilik rumah mungkin akan kehilangan beberapa kebutuhan unik yang diinginkan oleh pemilik rumah. Dengan demikian, diperlukan keterlibatan kedua belah pihak (pemilik rumah dan arsitek) dalam upaya menghasilkan desain rumah yang sesuai dengan kebutuhan pemilik rumah.

Oleh karena itu, pendekatan yang paling efektif adalah membuat pelaku/ pengguna bisnis dan analis bekerja sama untuk menentukan kebutuhan. Bahkan, fase analisis seyogyanya melibatkan interaksi yang signifikan dengan orang-orang yang memiliki kepentingan pada sistem baru (sering disebut pemangku kepentingan). Salah satu tugas utama analis adalah mengidentifikasi sumber utama kebutuhan, termasuk sponsor proyek dan semua pengguna sistem (baik langsung maupun tidak langsung). Juga penting dipahami bahwa semua perspektif pengguna harus dipertimbangkan dalam kegiatan analisis kebutuhan.

Analisis juga harus mempertimbangkan cara terbaik untuk mendapatkan kebutuhan dari para pemangku kepentingan. Ada berbagai teknik elitisasi yang dapat digunakan untuk memperoleh informasi, termasuk wawancara, kuesioner, observasi, *Joint Application Development* (JAD), dan analisis dokumen (akan dibahas pada bagian selanjutnya). Informasi yang dikumpulkan oleh teknik-teknik ini dianalisis secara kritis dan digunakan untuk membuat pernyataan definisi kebutuhan. Analis bekerja dengan seluruh tim proyek dan pengguna bisnis untuk memverifikasi, mengubah, dan melengkapi daftar kebutuhan dan, jika perlu, memprioritaskan pentingnya kebutuhan yang diidentifikasi. Semasa proses ini, kasus penggunaan, model proses, dan model data dapat digunakan untuk mengklarifikasi dan mendefinisikan ide-ide untuk sistem baru. Proses ini berlanjut sepanjang fase analisis, dan definisi kebutuhan berkembang seiring waktu ketika kebutuhan baru diidentifikasi dan ketika proyek bergerak ke fase selanjutnya dari SDLC.

Perlu diperhatikan bahwa evolusi definisi kebutuhan harus dikelola dengan hati-hati. Menjaga daftar kebutuhan secara ketat dan fokus adalah kunci keberhasilan proyek. Tim proyek tidak dapat terus menambahkan item baru ke definisi kebutuhan atau sistem akan terus tumbuh dan berkembang dan tidak pernah selesai. Alih-alih, tim proyek dengan hati-hati mengidentifikasi persyaratan dan mengevaluasi mana yang sesuai dengan ruang lingkup sistem. Ketika suatu kebutuhan mencerminkan kebutuhan bisnis yang nyata, tetapi tidak berada dalam ruang lingkup sistem saat ini atau rilis saat ini, ia harus dievaluasi dalam hal kepentingan dan dampaknya terhadap waktu dan anggaran. Mungkin kebutuhannya cukup penting untuk ditambahkan ke proyek saat ini,

bersama dengan penyesuaian yang sesuai dengan ruang lingkup proyek, anggaran, dan kerangka waktu. Kita tidak boleh berasumsi bahwa kebutuhan untuk proyek tidak pernah dapat diubah. Namun, mungkin juga kebutuhan tersebut dapat ditambahkan ke daftar kebutuhan di masa depan atau diberikan prioritas rendah. Pengelolaan kebutuhan (dan ruang lingkup sistem) adalah termasuk salah satu bagian tersulit dalam mengelola proyek pengembangan sistem.

#### D. PERNYATAAN DEFINISI KEBUTUHAN

Pernyataan definisi kebutuhan adalah laporan/teks yang hanya mencantumkan kebutuhan fungsional dan nonfungsional dalam format garis besar. Gambar 3.3 menunjukkan contoh definisi kebutuhan untuk Holiday Travel Vehicle, sebuah dealer kendaraaan (Dennis, 2012).

##### Kebutuhan Fungsional

- a. Manajemen kendaraan baru
  - 1.1 Sistem akan memungkinkan manajer kendaraan baru untuk melihat inventaris kendaraan baru saat ini
  - 2.1 Sistem akan memungkinkan manajer kendaraan baru untuk memesan kendaraan baru.
  - 3.1 Sistem akan mencatat penambahan kendaraan baru ke inventaris ketika kendaraan-kendaraan tersebut diterima dari pabrik
- b. Manajemen Penjualan Kendaraan
  - 2.1 Sistem ini akan memungkinkan tenaga penjualan untuk membuat penawaran pelanggan
  - 2.2 Sistem ini akan memungkinkan tenaga penjualan untuk mengetahui apakah suatu penawaran tertunda pada kendaraan tertentu
  - 2.3 Sistem ini akan memungkinkan manajer untuk mencatat persetujuan dari penawaran pelanggan.
  - 2.4 Sistem akan dapat digunakan untuk menyiapkan kontrak penjualan
  - 2.5 Sistem akan dapat menyiapkan pesanan berdasarkan opsi dealer yang diminta pelanggan

2.6 Sistem akan dapat mencatat setoran pelanggan
2.7 Sistem akan dapat mencatat pembayaran pelanggan
2.8 Sistem akan dapat membuat catatan pembelian kendaraan pelanggan.
c. Manajemen Kendaraan Bekas
3.1 Sistem akan dapat mencatat informasi tentang kendaraan yang dipertukarkan pelanggan
3.2 Dan sebagainya
<b>Kebutuhan Nonfungsional</b>
1. Operasional
1.1 Sistem harus dapat berjalan pada PC tablet untuk digunakan oleh tenaga penjualan
1.2 Sistem harus dapat berinteraksi dengan sistem manajemen toko
1.3 Sistem harus dapat terhubung ke printer secara nirkabel
2. Performa
2.1 Sistem harus dapat mendukung staf penjualan yang terdiri dari 15 tenaga penjualan
2.2 Sistem harus dapat diperbarui dengan penawaran yang tertunda pada kendaraan setiap 15 menit
3. Keamanan
3.1 Tidak ada tenaga penjual yang dapat mengakses kontak pelanggan tenaga penjual lainnya
3.2 Hanya pemilik dan manajer penjualan yang dapat menyetujui penawaran pelanggan
3.3 Penggunaan setiap PC tablet harus dibatasi hanya untuk tenaga penjual yang ditugaskan
4. Budaya dan Politik
4.1 Kebijakan perusahaan mengatakan bahwa semua peralatan komputer dibeli dari perusahaan Dell
4.2 Informasi pribadi pelanggan dilindungi sesuai dengan Undang-Undang Perlindungan Data
4.3 Sistem ini akan sesuai dengan hukum di Indonesia

**Gambar 3.3**  
Contoh Definisi Kebutuhan

Seperti yang ditunjukkan pada Gambar 3.3, cara umum untuk memberi nomor kebutuhan dalam format legal atau garis besar, sehingga setiap kebutuhan diidentifikasi dengan jelas. Adalah penting bahwa kebutuhan diidentifikasi dengan angka unik sehingga setiap kebutuhan dapat dengan mudah dilacak dalam seluruh proses pengembangan. Untuk kejelasan, kebutuhan biasanya dikelompokkan ke dalam pengelompokan fungsional dan nonfungsional. Kemudian, di dalam masing-masing kelompok itu, kebutuhan-kebutuhan tersebut diklasifikasikan lebih lanjut berdasarkan jenis kebutuhan atau berdasarkan area bisnis.

Terkadang, kebutuhan mesti disusun berdasarkan urutan prioritas pada pernyataan definisi kebutuhan. Item kebutuhan dapat digolongkan dari yang memiliki

kepentingan "tinggi", "sedang", atau "rendah" dalam sistem baru, atau item kebutuhan dapat diberi label dengan versi sistem yang akan memenuhi kebutuhan (misalnya, rilis 1, rilis 2, rilis 3). Praktik ini sangat penting dengan metodologi RAD yang memberikan kebutuhan dalam *batch* dengan mengembangkan versi tambahan sistem.

Tujuan paling jelas dari definisi kebutuhan adalah untuk memberikan pernyataan yang jelas tentang apa yang harus dilakukan sistem baru untuk mencapai visi sistem yang dijelaskan dalam permintaan sistem. Pemodelan seperti model proses dan model data menyediakan konten penjelas tambahan dalam berbagai format. Namun, tujuan yang sangat penting dari definisi kebutuhan adalah untuk mendefinisikan ruang lingkup sistem. Dokumen tersebut menjelaskan kepada analis apa yang perlu dilakukan sistem final. Selain itu, juga berfungsi untuk menetapkan harapan pengguna terhadap sistem. Jika pada suatu ketika muncul perbedaan pandangan antar pihak, dokumen berfungsi sebagai acuan untuk klarifikasi.

#### E. VALIDASI KEBUTUHAN

Validasi kebutuhan berkenaan dengan pengidentifikasiannya bahwa kebutuhan benar-benar mendefinisikan sistem yang diinginkan pengguna. Kegiatan ini memiliki banyak kesamaan dengan analisis karena berhubungan dengan penemuan masalah dengan kebutuhan. Namun demikian, keduanya merupakan proses yang berbeda karena validasi harus berhubungan dengan draft dokumen kebutuhan yang lengkap, sementara analisis melibatkan pekerjaan dengan kebutuhan yang tidak lengkap.

Validasi kebutuhan penting karena kesalahan pada dokumen kebutuhan dapat menimbulkan biaya penggerjaan ulang yang ekstensif jika ditemukan pada saat pengembangan atau setelah sistem dipakai. Biaya melakukan perubahan sistem yang merupakan akibat dari masalah kebutuhan, lebih besar dari perbaikan desain atau kesalahan pengkodean. Alasan untuk hal ini adalah karena perubahan kebutuhan biasanya mengharuskan perubahan desain sistem dan implementasinya, beserta pengujian ulang sistem.

Pada saat proses validasi kebutuhan, tipe pemeriksaan yang berbeda harus diterapkan pada kebutuhan-kebutuhan di dokumen kebutuhan. Pemeriksaan ini meliputi hal-hal berikut.

##### 1. Pemeriksaan validitas

Seorang user mungkin berpikir bahwa sistem diperlukan untuk melakukan fungsi-fungsi tertentu. Namun demikian, pemikiran dan analisis lebih lanjut dapat mengidentifikasi fungsi tambahan atau fungsi berbeda yang diinginkan. Sistem yang memiliki berbagai user dengan kebutuhan yang berbeda dengan kebutuhan apapun pada akhirnya akan merupakan suatu kompromi dari komunitas user.

2. Pemeriksaan Konsistensi

Kebutuhan pada dokumen seharusnya tidak bertentangan. Artinya, seharusnya ada batasan-batasan yang saling bertentangan atau deskripsi yang berbeda dari fungsi sistem yang sama.

3. Pemeriksaan Kelengkapan

Dokumen kebutuhan harus mencakup kebutuhan yang mendefinisikan semua fungsi dan batasan yang dimaksudkan oleh pihak pengguna.

4. Pemeriksaan Realisme

Dengan menggunakan pengetahuan mengenai teknologi yang ada, kebutuhan harus diperiksa untuk menjamin kebutuhan dapat diimplementasikan. Pemeriksaan ini harus memperhitungkan anggaran dan jadwal pengembangan sistem.

5. Kemampuan dapat Diverifikasi

Untuk mengurangi potensi pertentangan antara pengguna dan pengembang, kebutuhan sistem harus selalu dituliskan sedemikian rupa sehingga dapat diverifikasi. Ini berarti bahwa serangkaian pemeriksaan dapat dirancang untuk mendemonstrasikan bahwa sistem yang diserahkan memenuhi kebutuhan tertentu.

Ada sejumlah teknik validasi kebutuhan yang dapat digunakan secara bersama atau berdiri sendiri sebagai berikut.

1. Peninjauan Kebutuhan

Kebutuhan dianalisis secara sistematis oleh satu tim peninjau. Peninjauan kebutuhan biasanya merupakan proses manual yang melibatkan banyak pembaca, baik dari staf pengguna maupun pengembang yang memeriksa dokumen kebutuhan untuk menemukan kejanggalan dan hal-hal yang terlewatkan. Peninjauan kebutuhan dapat bersifat informal atau formal.

Peninjauan informal hanya melibatkan pengembang yang membahas kebutuhan dengan sebanyak mungkin pemangku kepentingan sistem. Banyak masalah yang bisa terdeteksi hanya dengan membicarakan sistem dengan pemangku kepentingan sebelum membuat komitmen untuk tinjauan formal.

Dalam peninjauan kebutuhan secara formal, tim pengembang harus menuntun pengguna melalui kebutuhan sistem, menjelaskan akibat dari setiap kebutuhan. Tim peninjau harus memeriksa setiap kebutuhan untuk konsistensi dan harus memeriksa kebutuhan sebagai satu kesatuan untuk kelengkapan.

2. Pembuatan Prototipe

Pada pendekatan validasi ini, diberikan sebuah model sistem yang dapat dijalankan (didemonstrasikan) kepada *end-user* dan pengguna. Mereka dapat berekspresi dengan model ini untuk melihat apakah model sistem itu

memenuhi kebutuhan mereka yang sebenarnya. Praktik ini biasanya dilakukan dalam metodologi pengembangan *prototyping*.

### 3. Pembuatan *test-case*

Idealnya, kebutuhan harus dapat diuji. Jika pengujian untuk kebutuhan dibuat sebagai bagian dari proses validasi, masalah-masalah kebutuhan seringkali akan terungkap. Jika pengujian sulit atau tidak mungkin dirancang, ini berarti bahwa kebutuhan akan sulit diimplementasikan dan harus dipertimbangkan lagi. Praktik ini biasanya dilakukan dalam metodologi pengembangan *V-Model*.



### Latihan

Untuk memperdalam pemahaman Anda mengenai materi di atas, kerjakanlah latihan berikut!

- 1) Jelaskan arti kebutuhan!
- 2) Jelaskan arti dari analisis, serta tujuan fase analisis dalam SDLC!
- 3) Sebutkan dan jelaskan tiga langkah dasar dalam proses analisis!
- 4) Jelaskan perbedaan kebutuhan fungsional dan kebutuhan nonfungsional, berikan masing-masing dua contoh!
- 5) Jelaskan arti kebutuhan sistem, berikan dua contoh!

#### *Petunjuk Jawaban Latihan*

- 1) Kebutuhan adalah pernyataan tentang apa yang harus dilakukan sistem atau karakteristik apa yang perlu dimiliki oleh sistem
- 2) Analisis kebutuhan adalah kegiatan menguraikan suatu sistem informasi yang utuh ke dalam bagian-bagian komponen untuk mengidentifikasi dan mengevaluasi permasalahan-permasalahan, kesempatan-kesempatan, hambatan-hambatan yang terjadi, dan kebutuhan-kebutuhan yang diharapkan, sehingga dapat diusulkan perbaikan-perbaikannya. Analisis kebutuhan bertujuan untuk mengidentifikasi kebutuhan-kebutuhan pengguna secara terperinci pada sistem baru yang akan dikembangkan.
- 3) Tiga tahapan pokok dalam fase analisis.
  - a) memahami sistem yang ada (sedang berjalan): dilakukan dengan mempelajari secara terinci (melalui serangkaian kegiatan penelitian) bagaimana sistem yang ada beroperasi;
  - b) mengidentifikasi permasalahan yang muncul dalam operasi sistem yang ada dan mengusulkan perbaikan-perbaikan atau penyempurnaan yang diharapkan;

- c) menentukan kebutuhan untuk sistem yang akan datang (baru), dilakukan dengan cara transformasi kebutuhan bisnis sistem menjadi daftar yang lebih terperinci mengenai apa yang harus dilakukan sistem baru untuk memberikan nilai yang dibutuhkan bagi bisnis.
- 4) Jenis-jenis kebutuhan:
- a) Kebutuhan bisnis: pernyataan yang menjelaskan tujuan keseluruhan sistem serta kontribusi yang akan dihasilkannya bagi keberhasilan organisasi.  
Contoh: (1) meningkatkan pangsa pasar; (2) mengurangi biaya operasional
  - b) Kebutuhan pengguna: pernyataan yang menjelaskan hal-hal yang dilakukan pengguna sebagai bagian integral dari operasi bisnis.  
Contoh: (1) memproses pesanan pelanggan; (2) reorder persediaan
  - c) Kebutuhan fungsional: pernyataan yang menjelaskan mengenai proses yang harus dilakukan sistem, atau informasi yang perlu disediakan sistem sebagai bagian dari mendukung tugas pengguna.  
Contoh: (1) sistem harus memeriksa pesanan pelanggan yang masuk untuk mempersiapkan persediaan; (2) sistem dapat menyimpan dan menyajikan informasi mengenai riwayat pesanan pelanggan.
  - d) Kebutuhan nonfungsional: sifat perilaku penting yang dimiliki oleh suatu produk atau sistem terkait dengan kinerja dan kegunaannya.  
Contoh: (1) sistem dapat diakses melalui perangkat seluler; (2) sistem harus dapat digunakan 1 x 24 jam sehari dan 365 hari per tahun.
- 5) Kebutuhan sistem adalah kebutuhan yang berfokus pada penjelasan mengenai cara membuat produk yang akan dihasilkan dari proyek sistem. Contoh: (1) model-model proses sistem; (2) model aliran data.



## Rangkuman

Analisis berfokus pada kegiatan menjaring kebutuhan (persyaratan) bisnis untuk sistem. Analisis mengidentifikasi "apa" yang perlu ada pada sistem, dan itu akan bermuara pada langkah awal dalam fase desain, yaitu "bagaimana" sistem ditentukan. Hasil yang diperoleh selama fase analisis yaitu definisi kebutuhan, termasuk kasus penggunaan, model proses, dan model data. Pada akhir analisis, semua hasil ini bersama dengan dokumen perencanaan awal yang telah direvisi, digabungkan ke dalam proposal sistem dan diserahkan kepada komite persetujuan untuk keputusan mengenai apakah proyek akan dilanjutkan ke fase desain.

Penentuan kebutuhan adalah bagian dari analisis di mana tim proyek mengubah kebutuhan bisnis secara umum yang dinyatakan dalam permintaan sistem menjadi daftar kebutuhan yang lebih terperinci. Kebutuhan adalah pernyataan tentang apa yang harus dilakukan sistem atau karakteristik apa yang perlu dimiliki. Kebutuhan bisnis menggambarkan tujuan keseluruhan sistem serta kontribusi "apa" yang akan dihasilkan

bagi keberhasilan organisasi, dan kebutuhan sistem menggambarkan "bagaimana" sistem akan dilaksanakan. Kebutuhan pengguna menjelaskan hal-hal yang dilakukan pengguna sebagai bagian integral dari operasi bisnis. Kebutuhan fungsional berhubungan langsung dengan proses yang harus dilakukan sistem atau informasi yang perlu dikandungnya, sedangkan kebutuhan nonfungsional mengacu pada properti perilaku yang harus dimiliki sistem, seperti kinerja dan kegunaan. Semua kebutuhan bisnis fungsional dan nonfungsional yang sesuai dengan ruang lingkup sistem dituliskan dalam definisi kebutuhan.

Validasi kebutuhan berkenaan dengan pengidentifikasiannya bahwa kebutuhan benar-benar mendefinisikan sistem yang diinginkan pengguna. Validitas kebutuhan dilakukan melalui serangkaian pemeriksaan terhadap dokumen kebutuhan yang meliputi pemeriksaan validitas, pemeriksaan konsistensi, pemeriksaan kelengkapan, pemeriksaan realisme, dan kemampuan untuk dapat diverifikasi. Validasi kebutuhan penting karena kesalahan pada dokumen kebutuhan dapat menimbulkan biaya penggerjaan ulang yang ekstensif jika ditemukan pada saat pengembangan atau setelah sistem dipakai.



### Tes Formatif 1

Pilihlah satu jawaban yang paling tepat!

- 1) Hasil akhir dari fase analisis adalah sebuah dokumen yang memuat ....
  - A. pernyataan kelayakan proyek sistem
  - B. pernyataan kebutuhan sistem secara terperinci
  - C. rancangan sistem secara lebih rinci tentang bagaimana sebuah sistem akan dibangun
  - D. jawaban A dan B benar
- 2) Pernyataan berikut ini merupakan pernyataan tentang "Kebutuhan Bisnis", *kecuali* ....
  - A. menjadwalkan pertemuan dengan klien
  - B. meningkatkan pangsa pasar
  - C. mengurangi biaya operasional
  - D. meningkatkan layanan kepada pelanggan
- 3) Di antara pernyataan berikut ini, yang termasuk dalam "Kebutuhan Pengguna" adalah ....
  - A. mengurangi biaya inventaris
  - B. sistem akan memungkinkan pelanggan untuk menelusuri pilihan berdasarkan kategori yang telah ditentukan
  - C. informasi pembayaran akan dienkripsi dan diamankan
  - D. reorder inventaris

- 4) Berikut ini, yang termasuk jenis kebutuhan fungsional adalah ....
  - A. kebutuhan yang berorientasi pada proses
  - B. kebutuhan yang berorientasi pada informasi
  - C. kebutuhan yang berorientasi pada performa
  - D. jawaban A dan B benar
- 5) "Kecepatan unduh sistem akan dipantau dan dijaga pada tingkat yang dapat diterima". Pernyataan ini adalah contoh jenis kebutuhan nonfungsional berdasarkan ....
  - A. operasional
  - B. performa
  - C. keamanan
  - D. kebijakan manajemen
- 6) Berikut adalah beberapa tipe pemeriksaan yang dapat diterapkan dalam proses validasi kebutuhan, *kecuali* ....
  - A. pemeriksaan validitas
  - B. pemeriksaan konsistensi
  - C. pemeriksaan prototipe
  - D. pemeriksaan kelengkapan
- 7) Dalam fase analisis, kegiatan memahami sistem yang sedang berjalan adalah suatu kegiatan yang mutlak dilakukan secara mendalam dan terperinci. Pernyataan tersebut adalah ....
  - A. Benar
  - B. Salah
- 8) "Kebutuhan Bisnis" adalah pernyataan yang menjelaskan tujuan keseluruhan sistem serta kontribusi yang akan dihasilkan bagi keberhasilan organisasi. Pernyataan tersebut adalah ....
  - A. Benar
  - B. Salah
- 9) Pernyataan yang menjelaskan hal-hal yang dilakukan pengguna sebagai bagian integral dari operasi bisnis disebut "kebutuhan sistem". Pernyataan tersebut adalah ....
  - A. Benar
  - B. Salah

- 10) Sebuah model sistem yang dapat didemonstrasikan kepada *end user* dan pengguna, di mana pengguna dapat bereksperimen dengan model ini untuk melihat apakah model sistem itu telah memenuhi kebutuhan mereka yang sebenarnya. Teknik validasi kebutuhan ini dikenal dengan nama *test-case*. Pernyataan tersebut adalah ....
- A. Benar
  - B. Salah

Cocokkanlah jawaban Anda dengan Kunci Jawaban Tes Formatif 1 yang terdapat di bagian akhir modul ini. Hitunglah jawaban yang benar. Kemudian, gunakan rumus berikut untuk mengetahui tingkat penguasaan Anda terhadap materi Kegiatan Belajar 1.

Tingkat Penguasaan =

$$\frac{\text{Jumlah Jawaban yang Benar}}{\text{Jumlah Soal}} \times 100$$

Arti tingkat penguasaan

<70%

70% - 79%

80% - 89%

90% - 100%

kurang

cukup

baik

baik sekali

Apabila mencapai tingkat penguasaan 80% atau lebih, Anda dapat meneruskan dengan Kegiatan Belajar 2. **Bagus!** Jika masih di bawah 80%, Anda harus mengulangi materi Kegiatan Belajar 1, terutama bagian yang belum dikuasai.

# Teknik Elisitasi dan Strategi Analisis Kebutuhan

**S**ebagai ilustrasi, analis sangat mirip detektif, dan pengguna bisnis terkadang seperti tersangka yang sulit ditangkap. Analis tahu bahwa ada masalah yang harus dipecahkan dan karena itu harus mencari petunjuk untuk mengungkap solusi. Sayangnya, petunjuknya tidak selalu jelas (dan sering terlewatkan), sehingga analis perlu memperhatikan dengan seksama, berbicara dengan saksi, dan mengikuti petunjuk-petunjuk yang ada. Analis terbaik akan benar-benar menggali kebutuhan menggunakan dengan berbagai teknik dan memastikan bahwa proses bisnis saat ini dan kebutuhan untuk sistem baru dipahami dengan baik sebelum pindah ke fase desain. Orang-orang tidak ingin mengetahui di kemudian hari bahwa mereka memiliki kebutuhan utama yang salah. Kejadian seperti ini di akhir SDLC dapat menyebabkan munculnya berbagai jenis masalah.

## A. ELISITASI KEBUTUHAN DALAM PRAKTIK

**Elisitasi kebutuhan** adalah sekumpulan aktivitas yang ditujukan untuk menemukan kebutuhan suatu sistem melalui komunikasi dengan pengguna bisnis, pengguna akhir sistem (*end user*), dan pihak lain yang memiliki kepentingan (*stakeholder*) dalam pengembangan sistem (Sommerville, 2016). Sebelum membahas teknik elisitasi secara terperinci, ada beberapa tips yang perlu diperhatikan.

Pertama, analis harus menyadari bahwa efek samping penting dari proses definisi kebutuhan adalah termasuk membangun dukungan politik untuk proyek dan membangun kepercayaan serta hubungan antara tim proyek dan pengguna akhir sistem (*end user*). Setiap kontak dan interaksi antara analis dan pengguna bisnis potensial atau manajer adalah peluang untuk membangkitkan minat, antusiasme, dan komitmen terhadap proyek. Oleh karena itu, analis harus siap untuk memanfaatkan peluang ini saat muncul selama proses definisi kebutuhan.

Kedua, analis harus hati-hati menentukan siapa yang dimasukkan dalam proses definisi kebutuhan. Pilihan untuk memasukkan (atau mengecualikan) seseorang adalah sangat berarti. Melibatkan seseorang dalam proses tersebut menyiratkan bahwa analis memandang orang itu sebagai sumber daya penting dan menghargai pendapatnya. Analis harus menyertakan semua pemangku kepentingan utama (orang-orang yang dapat mempengaruhi sistem atau yang akan dipengaruhi oleh sistem). Ini mungkin

termasuk manajer, karyawan, anggota staf, dan bahkan beberapa pelanggan dan pemasok. Analis juga harus peka terhadap fakta bahwa beberapa orang mungkin memiliki pengaruh yang signifikan dalam organisasi, walau mungkin mereka tidak menempati kedudukan tinggi dalam hierarki organisasi formal. Jika analis tidak melibatkan orang-orang kunci, orang-orang itu mungkin saja memiliki informasi penting, namun tidak dapat diperoleh sehingga menimbulkan masalah selama implementasi.

Akhirnya, analis harus melakukan segala sesuatu secara maksimal untuk menghargai komitmen para pihak yang telah bersedia menyempatkan waktu terlibat dalam proses elisitasi. Cara terbaik untuk melakukan ini adalah sepenuhnya siap memanfaatkan semua jenis teknik elisitasi. Meskipun, seperti yang sering terlihat bahwa wawancara adalah teknik yang paling umum digunakan, metode tidak langsung lainnya dapat membantu analis mengembangkan pemahaman dasar tentang domain bisnis sehingga teknik langsung lebih produktif. Secara umum, strategi yang berguna bagi analis untuk dipakai adalah memulai pengumpulan kebutuhan dengan mewawancarai manajer senior untuk mendapatkan "gambaran besar" tentang proyek. Wawancara awal ini kemudian dapat diikuti dengan analisis dokumen dan, mungkin, pengamatan proses bisnis untuk mempelajari lebih lanjut tentang domain bisnis dan sistem yang ada. Wawancara selanjutnya dapat dilakukan untuk mengumpulkan sisa informasi yang diperlukan untuk memahami sistem yang ada.

Berdasarkan beberapa pengalaman atau hasil penyelidikan lapangan, mengidentifikasi perbaikan atau kebutuhan paling umum dilakukan melalui sesi *Joint Application Development* (JAD) karena sesi ini memungkinkan pengguna dan pemangku kepentingan utama untuk bekerja bersama dan menciptakan pemahaman bersama mengenai rencana sistem yang akan datang. Kadang-kadang sesi JAD ini dilengkapi dengan kuesioner yang dikirim ke kelompok pengguna yang jauh lebih besar atau pengguna potensial untuk mendapatkan berbagai pendapat. Konsep kebutuhan sistem juga sering dikembangkan melalui wawancara dengan manajer senior, diikuti oleh sesi JAD dengan pengguna dari semua tingkatan, untuk memastikan bahwa kebutuhan utama dari sistem baru dipahami dengan baik.

Fokus pembahasan selanjutnya mengenai teknik-teknik elisitasi kebutuhan yang paling umum digunakan: wawancara, sesi JAD, kuesioner, analisis dokumen, dan observasi.

## 1. Wawancara

Wawancara adalah teknik elisitasi kebutuhan yang paling umum digunakan. Secara umum, wawancara dilakukan satu lawan satu (satu pewawancara dan satu orang yang diwawancarai), tetapi kadang-kadang karena keterbatasan waktu, beberapa orang diwawancarai pada saat yang sama. Ada lima langkah dasar untuk proses wawancara: memilih orang yang diwawancarai, merancang pertanyaan wawancara, mempersiapkan wawancara, melakukan wawancara, dan tindak lanjut pasca wawancara.

a. *Memilih Responden (Orang yang Diwawancarai)*

Jadwal wawancara harus memuat daftar siapa yang akan diwawancarai, tujuan wawancara, serta di mana dan kapan itu akan dilaksanakan (lihat Gambar 3.4). Orang-orang yang muncul pada jadwal wawancara dipilih berdasarkan kebutuhan informasi yang diperlukan analis. Sponsor proyek, pengguna bisnis utama, dan anggota lain dari tim proyek dapat membantu analis menentukan siapa di dalam organisasi yang paling dapat memberikan informasi penting tentang kebutuhan. Orang-orang ini didaftarkan pada jadwal wawancara sesuai urutan wawancara.

Orang-orang di berbagai tingkatan organisasi akan memiliki sudut pandang yang berbeda tentang sistem, sehingga penting untuk menyertakan manajer yang mengelola proses dan staf yang benar-benar melakukan proses untuk mendapatkan perspektif secara menyeluruh tentang suatu masalah. Juga, jenis-jenis subjek wawancara yang dibutuhkan dapat berubah seiring waktu. Misalnya, pada awal proyek, analis memiliki pemahaman yang terbatas tentang proses bisnis yang ada. Dalam situasi ini, aktivitas dimulai dengan mewawancara satu atau dua manajer senior untuk mendapatkan pandangan strategis, kemudian pindah ke manajer tingkat menengah yang dapat memberikan informasi yang luas dan menyeluruh tentang proses bisnis dan peran yang diharapkan dari sistem yang sedang dikembangkan. Begitu analis memiliki pemahaman yang baik tentang gambaran besar (secara umum), manajer tingkat rendah dan anggota staf dapat memberikan informasi secara rinci tentang bagaimana proses itu bekerja. Seperti pada kebanyakan kegiatan analisis sistem, proses ini selalu berulang, dimulai dengan manajer senior, pindah ke manajer tingkat menengah, kemudian anggota staf, kembali ke manajer tingkat menengah, dan seterusnya, tergantung pada informasi apa yang dibutuhkan sepanjang proses yang berjalan.

NAMA	POSISI	TUJUAN WAWANCARA	WAKTU PERTEMUAN
M. Rusdi, M.Ak.	Direktur Akuntansi	Visi strategis untuk sistem akuntansi baru	Senin, 2 Maret 2020; 08.00-10.00 WITA
Hesti, S.Ak.	Manajer Piutang	Masalah yang terkait dengan piutang saat ini; rencana ke depan	Senin, 2 Maret 2020; 14.00-15.15 WITA
Hendri J., S.Ak.	Manajer Hutang	Masalah yang terkait dengan hutang dagang saat ini; rencana ke depan	Rabu, 4 Maret 2020; 10.00-12.00 WITA
Harusl D.S., SE.	Supervisor Data	Proses piutang dan hutang	Kamis, 5 Maret 2020; 08.00-10.00 WITA
Hana Julio	Petugas Entri Data	Proses piutang dan hutang	Kamis, 5 Maret 2020; 13.00-14.30 WITA

Gambar 3.4  
Contoh Jadwal Wawancara

Sudah umum terjadi daftar orang yang diwawancara akan bertambah sekitar 50% samapai 75%. Ketika sedang mewawancara orang-orang, analis kemungkinan akan mengidentifikasi lebih banyak informasi yang diperlukan sehingga orang-orang yang akan memberikan informasi menjadi bertambah.

*b. Merancang Pertanyaan Wawancara*

Ada tiga jenis pertanyaan wawancara: pertanyaan tertutup, pertanyaan terbuka, dan pertanyaan menyelidik (menggali). **Pertanyaan tertutup** membutuhkan jawaban spesifik. Responden dapat menganggapnya mirip dengan pilihan ganda atau pertanyaan aritmatika pada ujian (lihat Gambar 3.5). Pertanyaan-pertanyaan tertutup digunakan ketika analis mencari informasi spesifik dan tepat, misalnya: "Berapa jumlah permintaan kartu kredit yang diterima per hari?". Secara umum, pertanyaan yang tepat adalah yang terbaik. Misalnya, alih-alih bertanya: "Apakah anda menangani banyak permintaan?", lebih baik langsung bertanya: "Berapa banyak permintaan yang anda proses per hari?". Pertanyaan tertutup memungkinkan analis untuk mengendalikan wawancara dan mendapatkan informasi yang mereka butuhkan. Namun, pertanyaan-pertanyaan semacam ini tidak mengungkap alasan mengapa jawabannya seperti itu, juga tidak mengungkap informasi yang pewawancara tidak pikirkan/duga sebelumnya.

JENIS PERTANYAAN	CONTOH
<b>Pertanyaan Tertutup</b>	<ol style="list-style-type: none"> <li>1. Berapa banyak pesanan melalui telepon yang diterima per hari?</li> <li>2. Bagaimana cara pelanggan memesan?</li> <li>3. Informasi apa yang hilang (tidak diperoleh) dari laporan penjualan bulanan?</li> </ol>
<b>Pertanyaan Terbuka</b>	<ol style="list-style-type: none"> <li>1. Apa pendapat Anda tentang cara memproses faktur saat ini?</li> <li>2. Apa saja masalah yang Anda hadapi setiap hari?</li> <li>3. Perbaikan apa saja yang anda inginkan mengenai cara pemrosesan faktur?</li> </ol>
<b>Pertanyaan Menyelidik (Menggali)</b>	<ol style="list-style-type: none"> <li>1. Mengapa?</li> <li>2. Dapatkan Anda memberi saya contoh?</li> <li>3. Dapatkah Anda menjelaskan sedikit lebih detail?</li> </ol>

Gambar 3.5  
Contoh Jenis Pertanyaan

**Pertanyaan terbuka** adalah pertanyaan-pertanyaan yang memberikan ruang untuk dijabarkan oleh pihak yang diwawancara. Dalam banyak hal pertanyaan ini serupa dengan pertanyaan esai yang biasa ditemukan pada ujian (lihat Gambar 3.5

sebagai contoh). Pertanyaan terbuka dirancang untuk mengumpulkan informasi yang kaya dan memberi orang yang diwawancara lebih banyak kendali atas informasi yang diungkapkan selama wawancara. Kadang-kadang responden yang diwawancara memilih untuk mengungkap informasi lain yang sama pentingnya dengan jawaban yang diharapkan oleh pewawancara (misalnya: jika orang yang diwawancara memberikan keterangan tentang departemen lain ketika diminta mengungkapkan masalah, itu mungkin menunjukkan bahwa ia enggan untuk membeberkan masalah pada departemennya sendiri).

Jenis pertanyaan ketiga adalah **pertanyaan menyelidik (menggali)**. Pertanyaan menyelidik adalah pertanyaan tindaklanjut mengenai hal yang telah dibahas agar pewawancara dapat mempelajari lebih banyak, dan itu sering digunakan ketika pewawancara merasa orang diwawancara memberikan informasi yang tidak jelas. Mereka mendorong orang yang diwawancara untuk memperluas atau untuk mengkonfirmasi informasi dari tanggapan sebelumnya. Banyak analis pemula enggan menggunakan pertanyaan menyelidik karena merasa takut bahwa orang yang diwawancara mungkin tersinggung karena merasa ditantang, atau karena pewawancara merasa bahwa itu menunjukkan bahwa pewawancara tidak mengerti apa yang dikatakan oleh orang yang diwawancara. Ketika dilakukan dengan sopan, pertanyaan menyelidik bisa menjadi alat yang ampuh dalam penemuan kebutuhan yang lebih jelas dan lengkap.

Secara umum, pewawancara harus menghindari pertanyaan mengenai suatu hal yang diperkirakan jawabannya dapat berasal dari banyak sumber. Misalnya, daripada menanyakan informasi apa yang digunakan untuk membuat sebuah tugas (misal: laporan), lebih baik menunjukkan kepada orang yang diwawancara contoh laporan yang dimaksudkan dan menanyakan informasi apa yang digunakan untuk menghasilkan laporan tersebut. Ini membantu memfokuskan orang yang diwawancara memberikan jawaban yang tepat dan menghemat waktu, karena responden tidak perlu menjelaskan informasi secara rinci.

Pertanyaan wawancara harus mengantisipasi jenis informasi yang mungkin tidak diketahui oleh orang yang diwawancara. Manajer sering agak kurang memahami detail proses bisnis sehari-hari dan karenanya mungkin tidak dapat menjawab pertanyaan tentang hal itu, sedangkan anggota staf tingkat bawah dapat dengan mudah memberikan jawaban. Sebaliknya, karyawan tingkat bawah mungkin tidak dapat menjawab pertanyaan yang luas dan berorientasi kebijakan, sementara manajer bisa. Karena tidak ada seseorang yang ingin terlihat bodoh, maka hindari membingungkan orang yang diwawancara dengan menanyakan pertanyaan di luar bidang pengetahuan mereka.

Tidak ada jenis pertanyaan yang lebih baik dari pertanyaan yang lainnya, sebaiknya menggunakan kombinasi pertanyaan selama wawancara. Pada tahap awal proyek pengembangan Sistem Informasi proses yang ada bisa tidak jelas, sehingga proses wawancara dimulai dengan wawancara tidak terstruktur untuk mencari serangkaian informasi yang luas dan masih kasar untuk didefinisikan. Seiring kemajuan proyek, analis memahami proses bisnis dengan lebih baik, dan ia membutuhkan

informasi yang sangat spesifik tentang bagaimana proses bisnis dilakukan (misal: secara pasti bagaimana permintaan kredit pelanggan disetujui). Pada saat ini, analis melakukan wawancara terstruktur di mana set pertanyaan spesifik dikembangkan sebelum wawancara. Biasanya ada lebih banyak pertanyaan tertutup dalam wawancara terstruktur daripada dalam pendekatan tidak terstruktur.

Tidak peduli jenis wawancara apa yang sedang dilakukan, pertanyaan wawancara harus disusun dalam urutan logis sehingga wawancara mengalir dengan baik. Misalnya, ketika mencoba mengumpulkan informasi tentang proses bisnis saat ini, analis akan merasa lebih baik untuk memulai dari masalah yang paling penting ke yang paling tidak penting.

Ada dua pendekatan mendasar untuk mengatur pertanyaan wawancara, yaitu pendekatan *top-down* atau *bottom-up* (lihat Gambar 3.6).



Gambar 3.6  
Strategi Pertanyaan *Top-Down* dan *Bottom-Up*

Pada model *top-down*, pewawancara memulai dengan masalah umum yang luas dan secara bertahap menuju ke arah yang lebih spesifik. Pada model *bottom-up*, pewawancara memulai dengan pertanyaan yang sangat spesifik dan beralih ke pertanyaan yang luas. Dalam praktiknya, analis biasa mencampur dua pendekatan, memulai dengan masalah umum yang luas, beralih ke pertanyaan spesifik, dan kemudian kembali lagi ke masalah umum.

Pendekatan *top-down* adalah pendekatan yang paling umum digunakan oleh sebagian besar pewawancara. Pendekatan *top-down* memungkinkan orang yang diwawancara menjadi akrab dengan topik tersebut sebelum ia perlu memberikan informasi yang spesifik. Ini juga memungkinkan pewawancara untuk memahami sebuah masalah sebelum pindah ke yang lebih detail, karena pewawancara mungkin tidak

memiliki informasi yang cukup pada awal wawancara untuk mengajukan pertanyaan yang sangat spesifik. Yang paling penting, pendekatan *top-down* memungkinkan orang yang diwawancarai "terperangkap" untuk mengemukakan serangkaian masalah dalam bentuk gambaran besar sebelum mengemukakan secara terperinci, sehingga pewawancara cenderung tidak akan melewatkannya suatu masalah yang sangat penting.

Suatu situasi di mana strategi *bottom-up* mungkin lebih disukai adalah ketika analis telah mengumpulkan banyak informasi tentang masalah dan hanya perlu beberapa informasi pelengkap. Atau, *bottom-up* mungkin tepat jika anggota staf tingkat bawah merasa khawatir atau tidak mampu menjawab pertanyaan tingkat tinggi. Misalnya, "Bagaimana kita dapat meningkatkan layanan pelanggan?" mungkin merupakan pertanyaan yang terlalu luas untuk petugas layanan pelanggan, sedangkan pertanyaan spesifik mudah dijawab, misalnya, "Bagaimana dapat mempercepat pengembalian pelanggan?". Dalam hal apapun, semua wawancara harus dimulai dengan pertanyaan yang nonkontroversial dan kemudian secara bertahap beralih ke masalah yang lebih kontroversial setelah pewawancara merasa akrab dengan orang yang diwawancarai.

#### c. Mempersiapkan Wawancara

Mempersiapkan wawancara sama pentingnya seperti mempersiapkan presentasi. Pewawancara harus memiliki rencana wawancara umum yang berisi daftar pertanyaan yang akan ditanyakan dalam urutan yang logis. Mengkonfirmasi bagian-bagian di mana orang yang diwawancarai berada dan memastikan mereka memiliki pengetahuan yang relevan sehingga pewawancara tidak mengajukan pertanyaan yang tidak dapat ia jawab. Meninjau area topik, pertanyaan, dan menentukan mana yang memiliki prioritas terbesar jika pewawancara merasa akan kehabisan waktu.

Beberapa analis pemula lebih suka wawancara yang tidak terstruktur, dengan alasan dapat mengabaikan persiapan yang matang. Ini sangat berbahaya dan sering kali kontraproduktif, karena informasi apapun yang tidak dikumpulkan dalam wawancara pertama harus diperoleh pada upaya wawancara tindak lanjut, dan kebanyakan orang tidak suka diwawancarai berulang kali tentang masalah yang sama.

Ketika pewawancara menjadwalkan wawancara, pewawancara harus menginformasikan sebelumnya kepada orang yang diwawancarai tentang maksud dari kegiatan wawancara dan bidang-bidang yang akan didiskusikan secara mendalam. Dengan demikian orang yang diwawancarai memiliki waktu yang cukup untuk memikirkan masalah dan mengatur pemikirannya. Ini sangat penting ketika pewawancara adalah orang dari luar di organisasi yang akan mewawancarai karyawan tingkat bawah yang sering tidak dipertimbangkan pendapatnya dan mungkin tidak yakin mengapa mereka yang menjadi responden wawancara.

*d. Melakukan Wawancara*

Hal pertama yang dilakukan ketika memulai wawancara adalah membangun hubungan dengan orang yang diwawancarai sehingga tercipta sikap saling memercayai. Dengan begitu orang yang diwawancarai bersedia mengungkap seluruh kebenaran, bukan hanya memberikan jawaban yang pewawancara inginkan. Pewawancara harus tampak profesional, independen, dan tidak memihak. Wawancara harus dimulai dengan menjelaskan tujuan wawancara, alasan mengapa memilih mewawancarai orang tersebut, baru selanjutnya beralih ke pertanyaan wawancara yang direncanakan.

Sangat penting untuk berhati-hati mencatat semua informasi yang diberikan oleh orang yang diwawancarai, walaupun misalnya informasi tersebut tidak secara langsung memiliki relevansi dengan pertanyaan yang diajukan. Jangan takut untuk meminta orang yang diwawancarai memperlambat atau berhenti saat kita sedang menulis, karena ini mengindikasikan bahwa informasi orang yang diwawancarai sangat penting bagi kita.

Salah satu masalah yang berpotensi kontroversial adalah merekam atau tidak proses wawancara yang sedang berjalan. Rekaman memastikan bahwa pewawancara tidak kehilangan poin-poin penting, akan tetapi itu bisa menakutkan bagi orang yang diwawancarai. Sebagian besar organisasi memiliki kebijakan atau praktik yang dapat membenarkan rekaman wawancara, namun sebagian lagi tidak. Jadi harus mencari tahu hal tersebut sebelum memulai wawancara. Jika pewawancara khawatir kehilangan informasi ketika tidak bisa merekam wawancara, pewawancara dapat mengikutkan orang kedua untuk mencatat secara terperinci.

Saat wawancara berlangsung, penting bagi pewawancara untuk memahami masalah yang dibahas. Jika pewawancara tidak mengerti sesuatu, pastikan untuk bertanya, walau pertanyaan tersebut terkesan seperti pertanyaan yang "bodoh". Jika pewawancara tidak memahami sesuatu selama wawancara, pewawancara tentu tidak akan memahami apa yang telah dicatat sebelumnya. Salah satu strategi yang baik untuk meningkatkan pemahaman selama wawancara adalah dengan merangkum secara berkala poin-poin penting yang dikomunikasikan oleh orang yang diwawancarai. Ini menghindari kesalahanpahaman dan juga menunjukkan bahwa pewawancara mendengarkan dengan seksama.

Akhirnya, harus dipastikan untuk memisahkan fakta dari pendapat. Misalnya orang yang diwawancarai mengatakan: "Kami memproses terlalu banyak permintaan kartu kredit". Pernyataan tersebut adalah pendapat, dan akan berguna jika ditindaklanjuti dengan pertanyaan menyelidik/menggali, yaitu meminta untuk memperjelas pernyataan tersebut, misalnya: "Oh, berapa banyak yang Anda proses dalam sehari?". Sangat penting untuk memeriksa fakta, karena perbedaan antara fakta dan pendapat orang yang diwawancarai dapat menunjukkan bagian utama yang memerlukan perbaikan. Misalkan orang yang diwawancarai mengeluhkan "jumlah kesalahan yang terus meningkat", akan tetapi catatan menunjukkan bahwa kesalahan telah berkurang. Ini menunjukkan bahwa kesalahan dipandang sebagai masalah yang

sangat penting yang harus diatasi oleh sistem baru, bahkan ketika kesalahan-kesalahan tersebut telah berkurang.

Ketika wawancara hampir selesai, pastikan untuk memberikan waktu kepada orang yang diwawancarai untuk mengajukan pertanyaan atau memberikan informasi yang menurutnya penting tetapi bukan bagian dari rencana wawancara. Dalam beberapa kasus, ini akan memberikan informasi yang tidak terduga namun sangat penting. Demikian juga, akan bermanfaat untuk bertanya kepada orang yang diwawancarai apakah ada orang lain yang harus diwawancarai. Pastikan wawancara berakhir tepat waktu. Jika perlu, hilangkan beberapa topik jika waktu sudah tidak mencukupi, atau rencanakan untuk menjadwalkannya pada wawancara lain.

e. *Tindak Lanjut Pasca Wawancara*

Setelah wawancara selesai, analis perlu menyiapkan laporan wawancara yang menggambarkan informasi dari wawancara (Gambar 3.7).

Catatan Wawancara Disetujui oleh: Linda Hasanah	
<b>Orang yang Diwawancarai:</b>	Linda Hasanah Direktur Sumber Daya Manusia (SDM)
<b>Pewawancara :</b>	Hendry Artawan
<b>Tujuan Wawancara :</b>	<ul style="list-style-type: none"> <li>- Memahami laporan yang dihasilkan oleh sistem saat ini pada bagian Sumber Daya Manusia (SDM)</li> <li>- Menentukan kebutuhan informasi untuk sistem masa depan.</li> </ul>
<b>Ringkasan Wawancara :</b>	<ul style="list-style-type: none"> <li>- Contoh laporan dari semua laporan SDM saat ini dilampirkan pada laporan ini. Informasi yang tidak digunakan dan informasi yang hilang dicatat pada laporan.</li> <li>- Dua masalah terbesar pada sistem yang berjalan saat ini adalah:           <ol style="list-style-type: none"> <li>1. Pemrosesan data terlalu lama (Bagian SDM membutuhkan informasi dalam 2 hari sebelum akhir bulan, namun saat ini informasi diberikan kepada mereka tertunda 3 minggu.)</li> <li>2. Data berkualitas buruk. (Seringkali, laporan harus direkonsiliasi dengan database departemen SDM)</li> </ol> </li> <li>- Kesalahan data yang paling umum ditemukan dalam sistem saat ini adalah informasi mengenai tingkat jabatan yang salah dan informasi gaji yang hilang (kurang).</li> </ul>
<b>Item terbuka :</b>	<ul style="list-style-type: none"> <li>- Dapatkan laporan daftar karyawan saat ini dari Ary Sukamdana (ekstensi 4355).</li> <li>- Verifikasi perhitungan yang digunakan untuk menetapkan waktu liburan dengan Ary Sukamdana.</li> <li>- Jadwalkan wawancara lanjutan dengan Ahmad Irawan (ekstensi 2337) mengenai masalah kualitas data.</li> </ul>
<b>Catatan Rinci :</b>	Lihat transkrip terlampir.

Gambar 3.7  
Contoh Laporan Wawancara

Laporan tersebut berisi catatan wawancara, informasi yang dikumpulkan selama wawancara, dan dirangkum dalam format yang bermanfaat. Secara umum, laporan wawancara harus ditulis dalam waktu 48 jam wawancara, karena semakin lama didiamkan semakin besar kemungkinan lupa informasi.

Seringkali, laporan wawancara dikirim ke orang yang diwawancara, meminta untuk membacanya dan memberi tahu analis tentang klarifikasi atau pembaruan jika ada. Jika terdapat perubahan dan perubahan tersebut cukup signifikan, itu menunjukkan bahwa wawancara kedua diperlukan. Jangan pernah menyebarkan informasi seseorang tanpa persetujuan sebelumnya.

## 2. *Joint Application Development*

*Joint Application Development* (JAD) adalah teknik pengumpulan informasi yang memungkinkan tim proyek, pengguna, dan manajemen untuk bekerja sama dalam mengidentifikasi kebutuhan untuk sistem. IBM mengembangkan teknik JAD pada akhir tahun 1970-an, dan menjadi metode yang paling populer untuk mengumpulkan informasi dari pengguna (Wood, 1995). JAD adalah proses terstruktur di mana 10 hingga 20 pengguna bertemu di bawah arahan seorang fasilitator yang terampil dalam teknik JAD. Fasilitator adalah orang yang menetapkan agenda pertemuan dan memandu diskusi, namun tidak bergabung sebagai peserta dalam diskusi. Fasilitator tidak memberikan ide atau pendapat tentang topik yang dibahas dan tetap netral selama sesi diskusi. Fasilitator harus menjadi ahli dalam teknik mengelola kelompok dan teknik analisis dan desain sistem. Satu atau dua juru tulis membantu fasilitator dengan mencatat, membuat salinan, dan sebagainya. Seringkali, juru tulis akan menggunakan komputer dan *case tools* untuk mencatat informasi saat sesi JAD berlangsung.

Kelompok JAD bertemu selama beberapa jam, beberapa hari, atau beberapa minggu sampai semua masalah telah dibahas dan informasi yang diperlukan telah dikumpulkan. Sebagian besar sesi JAD berlangsung di ruang pertemuan yang disiapkan khusus, jauh dari kantor para peserta, sehingga mereka tidak terganggu. Ruang pertemuan biasanya diatur dalam bentuk "U" sehingga semua peserta dapat dengan mudah melihat satu sama lain. Di bagian depan ruangan (bagian terbuka dari "U"), ada papan tulis, bagan kertas dan/atau proyektor untuk digunakan oleh fasilitator memimpin diskusi.

Salah satu masalah klasik pada JAD sebagai media pertemuan kelompok adalah: kadang-kadang orang enggan untuk menantang pendapat orang lain (terutama bos mereka), beberapa orang sering mendominasi diskusi, dan tidak semua orang berpartisipasi. Dalam kelompok yang beranggotakan 15 orang misalnya, jika setiap orang berpartisipasi secara setara, maka setiap orang dapat berbicara hanya 4 menit setiap jam dan harus menjadi pendengar selama 56 menit, ini bukanlah cara yang efisien untuk mengumpulkan informasi.

*Electronic JAD* (e-JAD) berupaya mengatasi masalah ini dengan menggunakan *groupware*. Di ruang pertemuan e-JAD, setiap peserta menggunakan perangkat lunak

khusus terkoneksi pada komputer jaringan untuk mengirimkan ide secara anonim, melihat semua ide yang dihasilkan oleh grup, dan menilai dan memberi peringkat ide melalui voting. Fasilitator menggunakan alat elektronik dari sistem e-JAD untuk memandu proses dalam kelompok, mempertahankan anonimitas dan memastikan kelompok untuk fokus pada manfaat masing-masing ide dan bukan kekuatan atau kedudukan orang yang menyumbangkan ide. Dengan cara ini, semua peserta dapat berkontribusi pada saat yang sama, tanpa takut akan balas dendam dari orang-orang yang berbeda pendapat. Beberapa penelitian pada masa awal diperkenalkannya e-JAD menunjukkan bahwa e-JAD dapat mengurangi waktu yang dibutuhkan untuk menjalankan sesi JAD sebesar 50% - 80% (Dennis, 1999).

a. *Memilih Peserta*

Memilih peserta JAD pada dasarnya dilakukan sama dengan cara memilih peserta wawancara. Peserta dipilih berdasarkan kontribusi informasi yang mereka dapat berikan untuk sistem baru. Idealnya, peserta yang dibebaskan dari tugas rutin untuk menghadiri sesi JAD haruslah orang-orang terbaik di unit masing-masing untuk memberikan dukungan manajemen yang kuat.

Fasilitator haruslah seseorang yang ahli dalam teknik JAD atau e-JAD, dan idealnya adalah seseorang yang memiliki pengalaman pada bisnis yang sedang dibahas. Dalam banyak kasus, fasilitator JAD adalah konsultan di luar organisasi karena organisasi mungkin tidak memiliki keahlian JAD atau e-JAD untuk kebutuhan sehari-hari. Mengembangkan dan mempertahankan keahlian JAD atau e-JAD dalam organisasi membutuhkan biaya yang mahal.

b. *Merancang Sesi JAD*

Sesi JAD dapat berlangsung dari hanya setengah hari hingga mungkin beberapa minggu, tergantung pada ukuran dan ruang lingkup proyek. Dalam praktiknya, sebagian besar sesi JAD cenderung berlangsung 5 hingga 10 hari, yang dibagi dalam 3 periode (minggu) pelaksanaan. Setiap periode berlangsung 1 hingga 4 hari. Sesi JAD dan e-JAD biasanya bergerak mulai dari pengumpulan informasi hingga menghasilkan hasil analisis. Misalnya, pengguna dan analis secara bersama-sama dapat merancang kasus penggunaan fungsi (*use cases*), model proses, dan mendefinisikan kebutuhan.

Seperti halnya wawancara, kesuksesan JAD tergantung pada rencana yang cermat. Sesi JAD biasanya dirancang dan disusun berdasarkan prinsip yang sama seperti wawancara. Sebagian besar sesi JAD dirancang untuk mengumpulkan informasi spesifik dari pengguna, dan ini membutuhkan pengembangan serangkaian pertanyaan sebelum pertemuan. Perbedaan antara JAD dan wawancara adalah bahwa semua sesi JAD disusun melalui perencanaan yang sangat hati-hati. Secara umum, pertanyaan tertutup jarang digunakan, karena hal itu tidak memicu diskusi terbuka dan jujur yang khas dari JAD. Dalam praktik sehari-hari, akan lebih baik untuk menggunakan strategi *top-down* pada sesi JAD ketika mengumpulkan informasi. Biasanya pembahasan setiap

item diagendakan dalam waktu 30 menit, selanjutnya istirahat sebelum berpindah ke agenda lain untuk menghindari para peserta kelelahan.

c. *Mempersiapkan Sesi JAD*

Seperti halnya wawancara, penting untuk mempersiapkan analis dan peserta untuk sesi JAD dengan sebaik-baiknya. Karena sesi JAD diharapkan dapat melebihi wawancara mendalam dan biasanya dilakukan di luar lokasi kantor, peserta harus lebih peduli dalam mempersiapkannya. Sangat penting bagi para peserta untuk memahami apa yang diharapkan dari sesi JAD tersebut. Misalnya, jika tujuan sesi JAD adalah untuk mengembangkan pemahaman tentang sistem saat ini, maka peserta dapat membawa manual prosedur dan dokumen. Jika tujuannya adalah untuk mengidentifikasi perbaikan suatu sistem, maka mereka sudah harus memikirkan bagaimana akan meningkatkan sistem sebelum sesi JAD.

d. *Melaksanakan Sesi JAD*

Sebagian besar sesi JAD mengikuti agenda formal, dan sebagian besar memiliki aturan dasar formal yang mendefinisikan perilaku yang harus ditaati. Aturan dasar bersama mencakup mengikuti jadwal, menghormati pendapat orang lain, menerima perbedaan pandangan, dan memastikan bahwa hanya satu orang yang berbicara pada satu waktu.

Peran fasilitator JAD bisa jadi sangat menantang. Banyak peserta datang ke sesi JAD dengan pandangan yang kuat tentang sistem yang sedang dibahas. Para peserta menyampaikan pandangan-pandangan mereka sehingga sesi JAD bergerak maju ke arah yang positif, membuat para peserta menerima (namun tidak harus menyetujui) pendapat dan situasi yang berbeda dari pandangan mereka sendiri mengenai analisis dan desain sistem, JAD, dan keterampilan interpersonal. Beberapa analis sistem berusaha untuk memfasilitasi sesi JAD tanpa dilatih dalam teknik JAD, dan sebagian lainnya berguru pada fasilitator JAD yang terampil sebelum mereka mencoba untuk memimpin sesi pertama mereka.

Fasilitator JAD melakukan tiga fungsi utama. **Pertama**, memastikan bahwa kelompok yang difasilitasi tetap berpegang pada agenda. Satu-satunya alasan yang diperbolehkan untuk menyimpang dari agenda adalah ketika telah jelas bagi fasilitator, pemimpin proyek, dan sponsor proyek, bahwa sesi JAD telah menghasilkan beberapa informasi baru yang tidak terduga dan mengharuskan sesi JAD untuk berpindah ke sesi yang lainnya. Ketika peserta berusaha mengalihkan diskusi dari agenda, fasilitator harus tegas (namun tetap sopan) dalam mengarahkan diskusi kembali ke agenda dan mengembalikan kelompok ke jalur yang benar.

**Kedua**, fasilitator harus membantu kelompok memahami istilah teknis dan jargon (istilah khusus) yang melingkupi proses pengembangan sistem, dan membantu peserta memahami teknik analisis khusus yang digunakan. Peserta adalah pakar di bidangnya masing-masing, tetapi mereka mungkin bukan ahli dalam analisis sistem.

Oleh karena itu fasilitator harus menyederhanakan pembelajaran yang diperlukan dan mengajari peserta bagaimana cara memberikan informasi yang benar secara efektif.

**Ketiga**, fasilitator mencatat masukan atau usulan kelompok pada area terbuka, yang dapat berupa papan tulis, bagan flip, atau tampilan komputer. Dia menyusun informasi yang disediakan kelompok dan membantu kelompok mengenali masalah-masalah utama dan solusi-solusi penting. Dalam keadaan apapun fasilitator tidak boleh memasukkan pendapatnya ke dalam diskusi. Fasilitator harus tetap netral setiap saat dan hanya membantu kelompok dalam berproses. Pada saat fasilitator mencoba memberikan pendapat tentang suatu masalah, kelompok tidak akan lagi memandangnya sebagai pihak yang netral, tetapi lebih sebagai seseorang yang mencoba menggiring kelompok tersebut ke dalam suatu solusi yang telah ditentukan. Namun demikian, ini tidak berarti bahwa fasilitator tidak boleh mencoba membantu kelompok menyelesaikan masalah.

*Contoh:*

Jika ada dua item tampaknya sama dalam pandangan fasilitator, fasilitator tidak boleh mengatakan, "Saya pikir ini mungkin serupa." Sebaliknya, fasilitator harus bertanya, "Apakah ini serupa?". Jika kelompok memutuskan itu hal yang serupa, fasilitator dapat menggabungkannya dan proses dapat berlanjut. Namun, jika kelompok memutuskan bahwa kedua hal tersebut tidak serupa (terlepas dari apa yang diyakini fasilitator), fasilitator harus menerima keputusan tersebut dan melanjutkan. Kelompok selalu benar, dan fasilitator tidak memiliki hak untuk berpendapat.

Sudah merupakan hal yang umum bagi para peserta JAD untuk menggunakan sejumlah alat selama sesi JAD untuk mendefinisikan sepenuhnya sistem baru. **Kasus penggunaan** (*use cases*) dapat dibuat untuk menggambarkan bagaimana pengguna akan berinteraksi dengan sistem baru. **Prototipe** dapat dibuat untuk lebih memahami antarmuka pengguna atau navigasi melalui sistem. **Model proses** dapat dibangun untuk memahami perangkat lunak yang akan dikembangkan, sementara **model data** dapat digunakan untuk menggambarkan data yang akan ditangkap dan dipelihara atau diproses. Fasilitator dan analis di tim proyek harus menggunakan setiap alat yang mereka miliki untuk membantu para peserta mengklarifikasi dan mendefinisikan kebutuhan mereka pada sistem baru.

e. *Tindak Lanjut Pasca-JAD*

Seperti pada wawancara, laporan pasca sesi JAD disiapkan dan diedarkan di antara peserta sesi. Laporan pasca-sesi pada dasarnya sama dengan laporan wawancara pada Gambar 3.7. Hanya saja sesi JAD lebih panjang dan memberikan lebih banyak informasi, sehingga biasanya dibutuhkan satu atau dua minggu setelah sesi JAD untuk mempersiapkan laporan.

### 3. Kuesioner

Kuesioner adalah serangkaian pertanyaan tertulis untuk mendapatkan informasi dari individu. Kuesioner sering digunakan ketika ada sejumlah besar orang yang informasi dan pendapatnya dibutuhkan. Dalam banyak situasi, kuesioner biasanya digunakan untuk orang (responden) di luar organisasi (misalnya: untuk pelanggan atau vendor) atau untuk sistem dengan pengguna bisnis yang tersebar di banyak lokasi geografis. Ketika berbicara tentang kuesioner, pemikiran orang secara otomatis akan mengarah ke kertas, namun demikian saat ini lebih banyak kuesioner yang didistribusikan dalam bentuk elektronik, baik melalui email atau di Web. Distribusi elektronik dapat menghemat biaya secara signifikan, dibandingkan dengan mendistribusikan kuesioner kertas.

#### a. Memilih Peserta

Seperti halnya wawancara dan sesi JAD, langkah pertama adalah memilih individu yang akan dikirimi kuesioner. Namun demikian, tidak lazim untuk memilih setiap orang untuk dapat memberikan informasi yang bermanfaat. Pendekatan standar adalah memilih sampel dari orang-orang yang mewakili seluruh kelompok. Pedoman pengambilan sampel dibahas dalam sebagian besar buku statistik, jadi tidak akan dibahas di sini. Poin penting dalam memilih sampel adalah menyadari bahwa tidak semua orang yang menerima kuesioner akan benar-benar menyelesaikannya. Rata-rata hanya 30% - 50% dari kuesioner kertas dan kuesioner e-mail dikembalikan. Tingkat respons untuk kuesioner berbasis web cenderung jauh lebih rendah (seringkali hanya 5% - 30%).

#### b. Merancang Kuesioner

Mengembangkan pertanyaan yang baik sangat penting untuk kuesioner karena informasi pada kuesioner tidak dapat segera diklarifikasi bagi responden yang bingung. Pertanyaan tentang kuesioner harus ditulis dengan sangat jelas dan harus meminimalkan kesalahpahaman. Oleh karena itu, pertanyaan tertutup cenderung paling umum digunakan. Pertanyaan harus memungkinkan analis untuk memisahkan fakta dari pendapat. Pertanyaan pendapat sering menanyakan responden sejauh mana mereka setuju atau tidak setuju, sementara pertanyaan faktual mencari nilai yang lebih tepat.

*Contoh:*

- 1) Pertanyaan pendapat: "Apakah masalah pada sistem jaringan lumrah (umum) terjadi?".
- 2) Pertanyaan faktual: "Seberapa sering masalah pada jaringan terjadi: satu jam sekali, satu kali sehari, atau seminggu sekali?".

Panduan tentang desain kuesioner dapat dilihat pada Gambar 3.8.

- Memulai dengan pertanyaan yang menarik dan tidak mengintimidasi atau mengancam
- Mengelompokkan item menjadi bagian-bagian yang logis secara koheren
- Tidak meletakkan hal-hal yang penting di akhir kuesioner
- Tidak memadati halaman dengan terlalu banyak item
- Menghindari menulis singkatan yang tidak umum
- Menghindari item atau ketentuan yang memiliki makna bias
- Menyertakan nomor pertanyaan untuk menghindari kebingungan
- Menyediakan anonimitas bagi responden

**Gambar 3.8**  
Desain Kuesioner yang Bagus

Gaya penyajian pertanyaan harus relatif konsisten sehingga responden tidak harus membaca instruksi berkali-kali untuk setiap pertanyaan sebelum menjawabnya. Biasanya merupakan praktik yang baik untuk mengelompokkan pertanyaan yang saling terkait (koheren) agar lebih mudah dijawab. Beberapa ahli menyarankan bahwa kuesioner harus dimulai dengan pertanyaan yang penting bagi responden, sehingga kuesioner segera menarik minat mereka dan mendorong mereka untuk menjawabnya. Mungkin langkah yang juga penting sebelum mengirim kuesioner kepada responden adalah meminta beberapa kolega mengulang kuesioner dan kemudian melakukan *pretest* kepada beberapa orang yang diambil dari kelompok yang akan dikirim kuesioner, sebab seringkali pertanyaan yang sederhana pun dapat disalahpahami.

#### c. Mengelola Kuesioner

Masalah utama dalam mengelola kuesioner adalah membuat peserta mengisi kuesioner dan mengirimkannya kembali. Puluhan buku riset pemasaran telah ditulis tentang cara-cara untuk meningkatkan tingkat respons. Teknik yang umum digunakan di antaranya menjelaskan mengapa kuesioner dilakukan dan mengapa responden dipilih; menyatakan tanggal kapan kuesioner akan dikembalikan; menyediakan fasilitas untuk melengkapi kuesioner (misalnya: menyertakan pena gratis); dan menawarkan untuk mengirimkan ringkasan tanggapan kuesioner. Analis sistem mungkin juga memiliki teknik tambahan untuk meningkatkan tingkat respons di dalam organisasi, seperti membagikan kuesioner secara pribadi dan secara pribadi menghubungi mereka yang belum mengembalikannya setelah satu atau dua minggu, serta meminta pengawas responden untuk mengelola kuesioner dalam pertemuan kelompok.

d. *Tindak Lanjut Kuesioner*

Sangat penting untuk memproses kuesioner yang telah dikembalikan oleh responden, dan membuat laporan kuesioner segera setelah batas waktu kuesioner. Hal ini untuk memastikan proses analisis berlangsung secara tepat waktu dan responden yang meminta salinan hasil dapat menerimanya segera.

#### 4. Analisis Dokumen

Tim proyek sering menggunakan analisis dokumen untuk memahami sistem yang ada. Secara ideal ketika tim proyek mengembangkan sistem yang ada, akan menghasilkan dokumentasi yang kemudian diperbarui lagi pada pengembangan proyek berikutnya. Dalam hal ini, tim proyek dapat memulai dengan meninjau dokumentasi dan memeriksa sistem yang ada (sistem sebelumnya).

Sayangnya, sebagian besar sistem tidak terdokumentasi dengan baik, karena tim proyek gagal mendokumentasikan proyek mereka di sepanjang jalan, sehingga ketika proyek selesai, tidak ada lagi waktu untuk kembali mendokumentasikan. Oleh karena itu, mungkin tidak tersedia banyak dokumentasi teknis tentang sistem yang ada saat ini, atau mungkin tidak mengandung informasi terbaru tentang perubahan sistem terbaru. Namun, ada banyak dokumen bermanfaat yang ada di organisasi, seperti: laporan manual (kertas), memorandum, manual kebijakan/prosedur, manual pelatihan pengguna, bagan organisasi, dan formulir. Catatan-catatan atau laporan permasalahan sistem yang dimiliki oleh pengguna sistem juga dapat menjadi sumber informasi lain mengenai masalah pada sistem yang ada.

Akan tetapi dokumen-dokumen ini (formulir, laporan, manual prosedur/kebijakan, bagan organisasi) hanya menyajikan sebagian dari cerita yang ada di organisasi. Dokumen-dokumen tersebut mewakili sistem formal yang digunakan organisasi. Sering terjadi, di mana sistem "nyata" atau informal berbeda dari yang formal, dan perbedaan-perbedaan ini (terutama yang besar) memberikan indikasi kuat tentang apa yang perlu diubah. Misalnya, formulir atau laporan yang tidak pernah digunakan kemungkinan harus dihilangkan. Demikian juga, pertanyaan pada formulir yang tidak pernah diisi (atau digunakan untuk tujuan/maksud lain) harus dipertimbangkan kembali. Contoh bagaimana sebuah dokumen dapat diinterpretasikan berbeda dapat dilihat pada Gambar 3.9.

Pelanggan melakukan kesalahan. Ini harus diberi label "Nama Pemilik" untuk mencegah kebingungan.

Staf harus menambahkan informasi tambahan tentang jenis hewan dan tanggal lahir serta jenis kelamin hewan tersebut. Informasi ini harus ditambahkan ke formulir baru di sistem yang baru.

PUSAT KLINIK HEWAN			
Kartu Informasi Pasien			
Nama	:	Buffy	Pat Smith
Nama Binatang Peliharaan	:	Buffy	Jakarta, 3/3/2017
Alamat	:	Jl. Warung Buncit No. 16 Jakarta 021-	
Nomor Telepon	:	3333-3333	
Dan sebagainya			

Pelanggan tidak memasukkan kode area dalam nomor telepon. Ini harus dibuat lebih jelas.

**Gambar 3.9**  
Contoh Melakukan Analisis Dokumen

Indikasi paling kuat bahwa sistem perlu diubah adalah ketika pengguna membuat formulir sendiri atau menambahkan informasi tambahan ke yang sudah ada. Perubahan tersebut jelas menunjukkan perlunya perbaikan sistem yang ada. Dengan demikian, penting untuk meninjau kembali formulir yang kosong dan/atau yang sudah diisi untuk mengidentifikasi penyimpangan ini. Demikian juga, ketika pengguna mengakses beberapa laporan untuk memenuhi kebutuhan informasi mereka, itu adalah tanda yang jelas bahwa informasi baru atau format informasi baru diperlukan.

## 5. Pengamatan

Pengamatan adalah tindakan menyaksikan proses yang sedang dilakukan. Pengamatan merupakan alat yang ampuh untuk mendapatkan wawasan tentang sistem yang ada. Pengamatan memungkinkan analis untuk melihat realitas suatu situasi, daripada mendengarkan orang lain menggambarkannya dalam wawancara atau sesi JAD. Beberapa studi penelitian menunjukkan bahwa banyak manajer benar-benar tidak ingat bagaimana mereka bekerja dan seberapa lama mereka mengalokasikan waktu mereka. Pengamatan adalah cara yang baik untuk memeriksa validitas informasi yang dikumpulkan dari sumber lain seperti wawancara dan kuesioner.

Dalam banyak hal, analis menjadi seorang antropolog ketika ia berjalan mengelilingi organisasi dan mengamati sistem bisnis sebagaimana fungsinya. Tujuannya adalah untuk menjaga agar tidak mengganggu mereka yang bekerja, dan untuk tidak memengaruhi yang diamati. Meskipun demikian, penting untuk dipahami bahwa apa yang diamati oleh para analis mungkin bukan rutinitas sehari-hari yang

normal karena orang-orang cenderung sangat berhati-hati dalam perilaku mereka ketika mereka merasa sedang diawasi (Franke, 1978). Jadi, apa yang kelihatan pada saat itu mungkin bukanlah sesuatu yang sebenarnya diinginkan.

Pengamatan sering digunakan untuk melengkapi informasi wawancara. Lokasi kantor seseorang dan perabotannya memberikan petunjuk tentang kekuatan dan pengaruh mereka dalam organisasi, dan petunjuk semacam itu dapat digunakan untuk mendukung atau membantah informasi yang diberikan dalam wawancara. Contoh, seorang analis mungkin menjadi skeptis terhadap seseorang yang mengklaim menggunakan sistem komputer yang ada secara luas jika komputer tidak pernah dihidupkan saat analis berkunjung. Dalam kebanyakan kasus, pengamatan akan mendukung informasi yang diberikan pengguna dalam wawancara. Ketika tidak, itu adalah sinyal penting bahwa perhatian ekstra harus diambil dalam menganalisis sistem bisnis.

## 6. Memilih Teknik yang Tepat

Setiap teknik elitisasi yang baru saja dibahas memiliki kekuatan dan kelemahan. Tidak ada satu teknik yang selalu lebih baik dari yang lain, dan dalam praktiknya sebagian besar proyek diuntungkan ketika menggunakan kombinasi teknik-teknik yang ada. Dengan demikian, penting untuk memahami kekuatan dan kelemahan masing-masing teknik dan kapan menggunakannya (lihat Gambar 3.10). Salah satu masalah yang tidak dibahas adalah pengalaman analis. Secara umum, analisis dokumen dan observasi membutuhkan pengalaman kerja paling sedikit, sedangkan sesi JAD adalah yang memerlukan pengalaman yang cukup.

	Wawancara	JAD	Kuesioner	Analisis Dokumen	Pengamatan
Jenis Informasi	Apa yang ada saat ini, peningkatan, Kebutuhan mendatang Tinggi	Apa yang ada saat ini, peningkatan, Kebutuhan mendatang Tinggi	Apa yang ada saat ini, peningkatan	Apa yang ada saat ini	Apa yang ada saat ini
Kedalaman Informasi					
Luasnya Informasi	Rendah	Sedang	Tinggi	Tinggi	Rendah
Integrasi Informasi	Rendah	Tinggi	Rendah	Rendah	Rendah
Keterlibatan Pengguna	Sedang	Tinggi	Rendah	Rendah	Rendah
Biaya	Sedang	Rendah-Sedang	Rendah	Rendah	Rendah-Sedang

Gambar 3.10  
Perbandingan Teknik Elitisasi Kebutuhan

a. Jenis Informasi

Karakteristik pertama adalah jenis informasi yang akan diperoleh. Beberapa teknik lebih cocok untuk digunakan pada berbagai tahap proses analisis, apakah memahami sistem yang ada, mengidentifikasi perbaikan, atau mengembangkan sistem yang akan datang. Wawancara dan JAD biasanya digunakan dalam ketiga tahap. Sebaliknya, analisis dan pengamatan dokumen biasanya paling membantu untuk memahami sistem yang ada, meskipun kadang-kadang memberikan informasi tentang perbaikan. Kuesioner sering digunakan untuk mengumpulkan informasi tentang sistem yang ada, serta informasi umum tentang kebutuhan untuk peningkatan sistem.

b. Kedalaman Informasi

Kedalaman informasi mengacu pada seberapa kaya dan terperinci informasi tersebut dihasilkan dari teknik yang ada dan sejauh mana teknik ini berguna untuk mendapatkan tidak hanya fakta dan pendapat, tetapi juga pemahaman tentang mengapa fakta dan pendapat itu ada. Wawancara dan sesi JAD sangat berguna dalam memberikan informasi yang kaya, terperinci, mendalam, dan membantu analis untuk memahami alasan di baliknya. Pada situasi yang lain, analisis dan pengamatan dokumen berguna untuk mendapatkan fakta. Kuisisioner dapat memberikan kedalaman informasi, mengumpulkan fakta dan pendapat, namun hanya menyediakan sedikit pemahaman tentang alasannya.

c. Luasnya Informasi

Luasnya informasi mengacu pada berbagai informasi dan sumber informasi yang dapat dengan mudah dikumpulkan oleh teknik yang ada. Kuisisioner dan analisis dokumen keduanya mampu dengan mudah mendapatkan berbagai informasi dari sejumlah besar sumber informasi. Sebaliknya, wawancara dan observasi mengharuskan analis untuk mengunjungi setiap sumber informasi secara individu sehingga membutuhkan lebih banyak waktu. Sesi JAD ada di tengah karena banyak sumber informasi disatukan pada saat yang sama.

d. Integrasi Informasi

Salah satu aspek yang paling menantang dari pengumpulan kebutuhan adalah mengintegrasikan informasi dari berbagai sumber. Sederhananya, orang yang berbeda dapat memberikan informasi yang bertentangan. Mengintegrasikan informasi dan berusaha untuk menyelesaikan perbedaan pendapat atau fakta biasanya sangat memakan waktu karena itu berarti harus menghubungi setiap sumber informasi secara bergantian, menjelaskan perbedaan, dan berusaha untuk memperbaiki informasi tersebut. Dalam banyak kasus, individu (responden) secara keliru menganggap bahwa analis meragukan informasinya, padahal

sebenarnya sumber konflik adalah pengguna lain dalam organisasi. Ini dapat membuat pengguna defensif dan menyulitkan untuk menyelesaikan perbedaan. Semua teknik mengalami masalah terkait integrasi sampai pada tingkat tertentu, akan tetapi sesi JAD dirancang untuk meningkatkan integrasi karena semua informasi terintegrasi ketika dikumpulkan, bukan terintegrasi sesudahnya. Jika dua pengguna memberikan informasi yang saling bertentangan, pertentangan tersebut segera menjadi jelas, misalnya sumber pertentangannya. Integrasi langsung informasi adalah karakteristik utama yang dimiliki JAD yang membedakannya dari teknik lain, dan inilah mengapa sebagian besar organisasi menggunakan JAD untuk proyek-proyek yang sangat penting.

e. Keterlibatan Pengguna

Keterlibatan pengguna diukur dari jumlah waktu dan energi yang harus dicurahkan oleh pengguna sistem baru dalam proses analisis. Secara umum disepakati bahwa, ketika pengguna menjadi lebih banyak terlibat dalam proses analisis, peluang keberhasilan meningkat. Namun, keterlibatan pengguna dapat menimbulkan biaya yang signifikan, dan tidak semua pengguna bersedia menyediakan waktu dan energi yang berharga itu. Kuesioner, analisis dokumen, dan observasi menempatkan beban paling sedikit pada pengguna, sedangkan sesi JAD membutuhkan upaya keterlibatan pengguna yang besar.

f. Biaya

Biaya selalu menjadi pertimbangan penting. Secara umum, kuesioner, analisis dokumen, dan observasi adalah teknik berbiaya rendah (meskipun observasi bisa memakan waktu lama). Biaya rendah tidak menyiratkan bahwa sebuah teknik tertentu yang digunakan lebih efektif daripada teknik lainnya. Pada umumnya wawancara dan sesi JAD dianggap memiliki biaya sedang. Sesi JAD pada awalnya jauh lebih mahal, karena JAD mengharuskan banyak pengguna meninggalkan pekerjaan mereka untuk periode yang cukup signifikan, dan mereka sering melibatkan konsultan yang dibayar tinggi. Namun, sesi JAD secara signifikan mengurangi waktu yang dihabiskan dalam integrasi informasi, dengan demikian lebih murah dalam hitungan jangka panjang.

## B. STRATEGI ANALISIS KEBUTUHAN

Subbab sebelumnya membahas lima teknik penting yang digunakan analis untuk berinteraksi dengan pemangku kepentingan dalam proyek pengembangan sistem untuk memperoleh dan menetapkan kebutuhan. Analis sering kali harus mendorong para pemangku kepentingan untuk berpikir kritis mengenai kebutuhan sistem agar dapat menemukan kebutuhan mendasar yang sebenarnya. Pada subbab ini, akan disajikan

beberapa strategi yang dapat digunakan analis dengan para pemangku kepentingan untuk mencapai tujuan tersebut.

### 1. Analisis masalah

Strategi analisis kebutuhan yang paling mudah (dan mungkin paling sering digunakan) adalah analisis masalah. Analisis masalah berarti meminta pengguna dan manajer untuk mengidentifikasi masalah pada sistem yang ada dan menggambarkan bagaimana menyelesaiannya dalam sistem yang akan datang. Sebagian besar pengguna memiliki gagasan yang sangat baik tentang perubahan yang ingin mereka lihat, dan sebagian besar dari mereka akan cukup vokal untuk menyarankan perubahan tersebut. Sebagian besar perubahan cenderung untuk memecahkan masalah daripada memanfaatkan peluang, akan tetapi memanfaatkan peluang juga mungkin dapat dilakukan. Perbaikan dari analisis masalah cenderung kecil dan bertahap (misalnya: menambahkan atribut untuk menyimpan nomor ponsel pelanggan; menyediakan laporan baru yang saat ini tidak ada).

Jenis perbaikan ini sering sangat efektif untuk meningkatkan efisiensi atau kemudahan penggunaan sistem. Namun, seringkali hanya memberikan dampak yang kecil dalam nilai bisnis. Sistem baru memang lebih baik daripada yang lama, tetapi mungkin sulit untuk mengidentifikasi manfaat finansial yang signifikan dari sistem baru.

### 2. Analisis Akar Penyebab

Analisis akar penyebab masalah yang dihasilkan oleh analisis masalah cenderung menjadi solusi untuk masalah. Setiap solusi membuat asumsi tentang sifat masalah, asumsi yang mungkin valid atau mungkin tidak valid. Dalam praktiknya, pengguna cenderung melompat cepat ke solusi tanpa sepenuhnya mempertimbangkan karakteristik masalah. Kadang-kadang solusinya tepat, tetapi sering kali solusi tersebut hanya mengatasi gejala dari masalah, dan bukan menyelesaikan masalah sebenarnya atau akar penyebabnya sendiri.

*Contoh:*

Anggaplah pengguna melaporkan bahwa "sering terjadi kehabisan persediaan". Sistem persediaan yang kehabisan persediaan tentu saja tidak baik, dan satu cara yang jelas untuk mengurangi terjadinya hal tersebut adalah dengan meningkatkan jumlah barang yang disimpan dalam persediaan. Namun, tindakan ini akan menimbulkan biaya, jadi ada baiknya untuk menyelidiki penyebab mendasar dari seringnya kehabisan stok alih-alih melompat ke solusi dengan cepat. Dalam kasus ini solusi yang diajukan pengguna (atau yang dipertimbangkan analis) dapat mengatasi gejala atau penyebab, akan tetapi tanpa analisis yang cermat, sulit untuk menentukan penyebab yang sesungguhnya. Menyadari di kemudian hari bahwa organisasi baru saja menghabiskan

jutaan rupiah namun ternyata belum menyelesaikan masalah mendasar yang sebenarnya adalah perasaan yang buruk!

Analisis akar penyebab berfokus pada masalah lebih dulu daripada solusi. Analis memulai dengan meminta pengguna membuat daftar masalah yang terdapat pada sistem saat ini, lalu memprioritaskan masalah dalam urutan kepentingan. Dimulai dengan yang paling penting, pengguna dan/atau analis menghasilkan semua akar penyebab yang mungkin untuk masalah tersebut.



Gambar 3.11  
Analisis Akar Masalah untuk Persediaan Habis

Seperti yang ditunjukkan pada Gambar 3.11, masalah “persediaan sering habis” memiliki beberapa akar penyebab yang potensial (penghitungan langsung yang tidak akurat; titik pemesanan ulang yang salah; keterlambatan dalam menempatkan pesanan pemasok). Setiap akar penyebab yang mungkin diselidiki dan akar penyebab tambahan diidentifikasi. Seperti yang ditunjukkan Gambar 3.11 terkadang berguna untuk menampilkan akar penyebab yang potensial dalam hierarki mirip pohon. Proses investigasi mengungkapkan akar penyebab masalah yang sebenarnya, memungkinkan tim untuk merancang sistem guna memperbaiki masalah dengan solusi yang tepat. Poin kunci dalam analisis akar permasalahan adalah untuk selalu menantang yang sudah jelas dengan menggali masalah lebih dalam sehingga penyebab mendasar yang sebenarnya terungkap.

### 3. Analisis Durasi

Analisis durasi memerlukan pemeriksaan secara terperinci dari jumlah waktu yang diperlukan untuk melakukan setiap proses dalam sistem yang berjalan saat ini. Para analis memulai dengan menentukan jumlah total waktu dan jumlah rata-rata waktu yang dibutuhkan untuk melakukan serangkaian proses bisnis untuk input tertentu.

Analisis kemudian menentukan waktu masing-masing langkah pada suatu proses (atau subproses) dalam proses bisnis. Waktu untuk menyelesaikan langkah-langkah dasar kemudian dijumlahkan dan dibandingkan dengan total untuk keseluruhan proses. Perbedaan yang signifikan antara keduanya (sering kali bisa mencapai 10 atau bahkan 100 kali lebih lama daripada jumlah bagian-bagiannya), menunjukkan bahwa bagian proses ini sangat membutuhkan perbaikan besar-besaran.

*Contoh:*

Anggaplah para analis yang bekerja pada sistem hipotek rumah menemukan bahwa, rata-rata dibutuhkan 30 hari bagi bank untuk menyetujui hipotek. Para analis kemudian melihat setiap langkah dasar dalam proses (misalnya: entri data, pemeriksaan kredit, pencarian judul, penilaian, dan lain-lain), dan menemukan bahwa total jumlah waktu yang dihabiskan untuk masing-masing hipotek adalah sekitar 8 jam. Ini mengindikasikan bahwa keseluruhan proses telah rusak parah, karena butuh 30 hari untuk melakukan pekerjaan yang seharusnya dapat diselesaikan hanya dalam 1 hari.

Masalah-masalah ini kemungkinan terjadi karena prosesnya terfragmentasi (terbagi-bagi). Orang-orang yang berbeda harus melakukan kegiatan yang berbeda (pemeriksaan kredit, penilaian dan pemeriksaan judul) sebelum keseluruhan proses dinyatakan selesai. Dalam contoh hipotek, formulir pengajuan mungkin berada (mengendap) di banyak meja orang yang berbeda dalam waktu yang lama sebelum diproses. Proses di mana banyak orang yang berbeda bekerja hanya pada sebagian kecil dari proses input, adalah pertanda utama untuk integrasi atau paralelisasi proses. Integrasi proses berarti mengubah proses dasar sehingga lebih sedikit orang yang bekerja pada bagian input, yang seringkali membutuhkan perubahan proses dan melatih kembali staf untuk melakukan berbagai tugas yang lebih luas. Proses paralelisasi berarti mengubah proses sehingga semua langkah individu dilakukan pada saat yang sama. Misalnya dalam contoh aplikasi hipotek, mungkin tidak ada alasan bahwa pemeriksaan kredit tidak dapat dilakukan secara bersamaan dengan penilaian dan pemeriksaan judul.

#### 4. **Informal Benchmarking**

*Benchmarking* mengacu pada mempelajari bagaimana organisasi lain melakukan proses bisnis, untuk menjadi rujukan bagaimana suatu organisasi dapat melakukan sesuatu yang lebih baik. *Benchmarking* membantu organisasi dengan memperkenalkan ide-ide yang mungkin tidak pernah dipertimbangkan oleh karyawan, namun berpotensi untuk menambah nilai.

*Informal Benchmarking* cukup umum untuk proses bisnis “menghadapi pelanggan” (yaitu proses yang berinteraksi dengan pelanggan). Dengan *informal benchmarking*, para manajer dan analis memikirkan organisasi lain, atau mengunjungi mereka sebagai pelanggan untuk menyaksikan bagaimana proses bisnis dilakukan. Dalam banyak kasus, organisasi bisnis yang diteliti dapat berupa organisasi pionir yang dikenal dalam industri atau hanya perusahaan yang terkait.

*Contoh:*

Anggaplah bahwa tim sedang mengembangkan situs Web untuk dealer mobil. Sponsor proyek, manajer utama, dan anggota tim utama kemungkinan akan mengunjungi situs Web para pesaing, situs Web organisasi lain di industri mobil (misalnya: pabrikan, pemasok aksesoris), dan situs Web industri lain yang telah memenangkan penghargaan sebagai situs Web terbaik.

### 5. Analisis Teknologi

Banyak perubahan besar dalam bisnis selama dekade terakhir telah dimungkinkan oleh teknologi baru. Oleh karena itu analisis teknologi dimulai dengan meminta analis dan manajer mengembangkan daftar teknologi penting, kemudian secara sistematis mengidentifikasi bagaimana setiap teknologi dapat diterapkan pada proses bisnis, dan mengidentifikasi bagaimana bisnis akan mendapat manfaat.

Misalnya Internet, suatu teknologi yang mungkin bermanfaat. Pabrikan dapat mengembangkan aplikasi intranet untuk pemasoknya. Daripada melakukan pemesanan pembelian suku cadang untuk produk-produknya, produsen lebih baik membuat jadwal produksinya tersedia secara elektronik kepada pemasoknya. Dengan begitu, pemasok dapat mengirimkan suku cadang yang dibutuhkan sehingga mereka tiba di pabrik tepat pada waktunya. Sistem seperti ini menghemat biaya yang signifikan karena menghilangkan aktivitas untuk memantau jadwal produksi dan mengeluarkan pesanan pembelian.

### 6. Penghapusan Aktivitas

Analis dan manajer bekerja bersama untuk mengidentifikasi bagaimana organisasi dapat menghilangkan beberapa aktivitas dalam proses bisnis, bagaimana fungsi dapat beroperasi tanpanya, dan dampak apa yang mungkin terjadi. Meskipun demikian, mereka terlebih dahulu harus membahas setiap kegiatan yang akan dihilangkan dalam proses bisnis.

Misalnya, dalam proses persetujuan hipotek rumah yang dibahas sebelumnya, para manajer dan analis akan mulai dengan menghilangkan aktivitas pertama, "memasukkan data ke dalam komputer perusahaan hipotek". Ini mengarah ke kegiatan menghilangkan penggunaan sistem komputer, dan membuat orang lain melakukan entri datanya sendiri (misalnya: pelanggan, melalui Web). Mereka kemudian melakukan pemeriksaan untuk menghilangkan kegiatan selanjutnya "evaluasi kredit". Kelihatannya tindakan ini agak "sembrono", sebab memastikan pemohon memiliki kredit yang sehat sangat penting sebelum mengeluarkan pinjaman. Namun tidaklah selalu demikian adanya. Jawaban sebenarnya tergantung pada berapa kali pemeriksaan kredit mengidentifikasi aplikasi permohonan yang buruk. Jika semua atau hampir semua pemohon memiliki kredit yang baik dan jarang ditolak oleh cek kredit, maka biaya pemeriksaan kredit mungkin tidak sebanding dengan manfaat dari hanya beberapa

pinjaman macet yang dicegahnya. Menghilangkan itu sebenarnya dapat mengakibatkan biaya yang lebih rendah, bahkan dengan biaya kredit macet, kecuali jika jumlah pemohon dengan kredit buruk sangat meningkat.

## 7. Membandingkan Strategi Analisis

Setiap strategi analisis kebutuhan yang dibahas di sini memiliki tujuannya masing-masing. Tidak ada satu teknik pun yang secara inheren lebih baik daripada yang lain. Perlu diingat bahwa suatu organisasi kemungkinan akan memiliki berbagai proyek dalam portofolionya; strategi analisis kebutuhan harus dipilih agar sesuai dengan sifat proyek. Analisis masalah dan analisis akar penyebab cenderung paling berguna dalam situasi fokus proyek sempit, di mana efisiensi menjadi tujuan utamanya. Analisis durasi dan penghapusan aktivitas membantu tim menemukan proses bisnis yang dianggap tidak efektif sehingga proses tersebut dapat dirancang ulang dan ditingkatkan. Analisis teknologi, sangat berguna ketika tim berusaha menciptakan cara yang sama sekali baru untuk menyelesaikan proses bisnis.



### Latihan

Untuk memperdalam pemahaman Anda mengenai materi di atas, kerjakanlah latihan berikut!

- 1) Jelaskan apa yang dimaksud dengan elisitasi kebutuhan! Hal-hal apa yang perlu diperhatikan oleh analis dalam proses elisitasi kebutuhan?
- 2) Jelaskan lima langkah utama melakukan wawancara untuk menjaring kebutuhan sistem!
- 3) Jelaskan pihak-pihak yang terlibat dalam sesi JAD! Apa peran masing-masing pihak?
- 4) Jelaskan perbedaan mendasar antara wawancara dan kuesioner sebagai teknik elisitasi kebutuhan! Pada situasi bagaimana masing-masing teknik tepat digunakan? Mengapa desain pertanyaan untuk kuesioner sangat sulit?
- 5) Jelaskan kekuatan dan kelemahan menjadikan observasi sebagai teknik elisitasi kebutuhan!

#### Petunjuk Jawaban Latihan

- 1) Elitisasi kebutuhan adalah sekumpulan aktivitas yang bertujuan menjaring/menemukan kebutuhan untuk sistem melalui komunikasi dengan pengguna bisnis, pengguna akhir sistem (*end user*), dan pihak lain yang memiliki kepentingan (*stakeholder*) dalam pengembangan sistem.

Hal-hal yang perlu diperhatikan dalam proses elisitasi kebutuhan:

- a) Elitisasi kebutuhan bukan sekedar kegiatan menjaring kebutuhan sistem, namun sebagai wadah membangun kepercayaan bagi *stakeholder* untuk memberikan komitmen yang kuat terhadap proyek. Dengan demikian, analis harus berupaya membangkitkan antusiasme mereka.
  - b) Analis harus berhati-hati menentukan pihak-pihak yang dilibatkan dalam penentuan kebutuhan, agar tidak ada informasi penting yang terlewatkan dalam menjaring kebutuhan sistem.
  - c) Analis harus siap menggunakan berbagai jenis teknik elitisasi untuk memaksimalkan proses elitisasi kebutuhan.
- 2) Langkah-langkah dasar melakukan wawancara untuk menjaring kebutuhan untuk sistem:
    - a) memilih responden: responden dipilih berdasarkan kebutuhan informasi yang diperlukan;
    - b) merancang pertanyaan wawancara: bentuk pertanyaan wawancara (pertanyaan terbuka, tertutup, menyelidik) harus disesuaikan dengan jenis informasi yang akan dikumpulkan (informasi spesifik atau informasi luas); pertanyaan wawancara juga harus disusun dalam urutan logis sehingga wawancara mengalir dengan baik;
    - c) mempersiapkan wawancara: dilakukan dengan cara mempersiapkan daftar pertanyaan yang akan ditanyakan dalam urutan yang logis, mengonfirmasi kesiapan responden, serta menetapkan skala prioritas pada area topik wawancara;
    - d) melakukan wawancara: dengan terlebih dahulu menjelaskan tujuan wawancara dan alasan memilih responden;
    - e) menindaklanjuti hasil wawancara: dengan cara menyiapkan laporan wawancara berisi informasi hasil wawancara dan meminta klarifikasi serta persetujuan responden atas informasi yang tertera dalam laporan wawancara.
  - 3) Pihak-pihak yang terlibat dalam sesi JAD untuk menjaring kebutuhan untuk sistem adalah: orang-orang pada berbagai tingkatan organisasi dan fasilitator.
    - a) Manajer senior: berperan memberikan pandangan strategis bisnis organisasi
    - b) Manajer tingkat menengah: berperan memberikan informasi yang luas dan menyeluruh tentang proses bisnis dan peran yang diharapkan dari sistem yang dikembangkan
    - c) Manajer tingkat bawah dan anggota staf: berperan memberikan informasi secara rinci tentang bagaimana sebuah proses bekerja
    - d) Fasilitator: berperan menetapkan agenda dan memandu diskusi, namun tidak bergabung sebagai peserta, dan senantiasa bertindak netral selama sesi diskusi JAD

- e) Pembantu fasilitator: berperan mencatat informasi selama sesi JAD berlangsung
  - f) Analis: berperan menjaring kebutuhan untuk sistem
- 4) Perbedaan mendasar antara wawancara dan kuesioner sebagai teknik elisitasi kebutuhan adalah sebagai berikut.
- a) Wawancara dilakukan dengan mengajukan pertanyaan secara langsung dan mencatat/merekam informasi yang diberikan oleh individu (responden). Kuesioner dilakukan dengan cara mengajukan pertanyaan secara tertulis untuk mendapatkan informasi dari individu.
  - b) Wawancara tepat digunakan untuk individu di dalam organisasi, dengan jumlah individu yang cenderung sedikit. Kuesioner tepat digunakan ketika ada sejumlah besar individu yang informasi atau pendapatnya dibutuhkan. Kuesioner juga tepat digunakan untuk individu di luar organisasi (pelanggan atau vendor), atau untuk sistem dengan pengguna bisnis yang tersebar di banyak lokasi geografis.
  - c) Desain pertanyaan untuk kuesioner sangat sulit, karena harus dipastikan tidak terjadi kesalahpahaman atas informasi yang tertera pada kuesioner. Jika terjadi kesalahpahaman, informasi pada kuesioner tidak dapat segera diklarifikasi bagi responden yang bingung.
- 5) Kekuatan dan kelemahan menjadikan observasi sebagai teknik elisitasi kebutuhan:
- Kekuatannya: memungkinkan analis untuk melihat realitas suatu situasi, mengingat responden seringkali tidak dapat mengingat secara penuh suatu kegiatan yang mereka lakukan atau alami. Observasi juga dapat menjadi alat konfirmasi informasi yang diperoleh pada sesi JAD atau wawancara, sehingga validitas informasi menjadi lebih baik.
- Kelemahan: subjek pengamatan seringkali menampilkan situasi yang tidak sebenarnya ketika mereka mengetahui bahwa mereka sedang diawasi, sehingga informasi yang diperoleh boleh jadi tidak mencerminkan situasi yang sebenarnya.



## Rangkuman

Lima teknik dapat digunakan untuk memperoleh kebutuhan bisnis untuk sistem yang diusulkan: wawancara, pengembangan aplikasi bersama, kuesioner, analisis dokumen, dan observasi. Wawancara melibatkan pertemuan dengan satu atau lebih orang dan mengajukan pertanyaan kepada mereka. Ada lima langkah dasar untuk proses wawancara: memilih orang yang diwawancarai, merancang pertanyaan wawancara, mempersiapkan wawancara, melakukan wawancara, dan tindak lanjut pasca wawancara. Pengembangan aplikasi bersama (*Joint Application Development/JAD*)

memungkinkan tim proyek, pengguna, dan manajemen untuk bekerja sama mengidentifikasi kebutuhan untuk sistem. *Electronic JAD* berupaya mengatasi masalah umum yang terkait dengan grup dengan menggunakan *groupware*. Kuesioner adalah serangkaian pertanyaan tertulis yang dikembangkan untuk mendapatkan informasi dari individu. Kuesioner sering digunakan ketika ada sejumlah besar orang yang menjadi sumber informasi atau pendapat. Analisis dokumen mencakup peninjauan dokumentasi yang ada dan memeriksa sistem itu sendiri. Ini dapat memberikan wawasan ke dalam sistem formal dan informal. Pengamatan, tindakan mengamati proses yang dilakukan, adalah alat yang kuat untuk mengumpulkan informasi tentang sistem saat ini karena memungkinkan analis untuk melihat realitas situasi secara langsung.

Analis sering harus membantu pengguna bisnis berpikir kritis tentang kebutuhan sistem baru mereka. Beberapa strategi sangat membantu. Analisis masalah dan analisis akar masalah adalah dua strategi yang dapat membantu pengguna bisnis dalam memahami masalah dan masalah sistem saat ini yang memerlukan perbaikan. Analisis durasi dan penghilangan aktivitas adalah strategi analisis populer yang membantu tim menemukan proses yang paling membutuhkan desain ulang. Analisis hasil, analisis teknologi, dan *benchmarking* adalah tiga strategi yang dapat digunakan untuk “memaksa” pengguna bisnis untuk memikirkan proses penyempurnaan bisnis dengan cara-cara baru, atau mungkin menemukan cara yang sepenuhnya baru untuk menyelesaikan proses bisnis.



### Tes Formatif 2

Pilihlah satu jawaban yang paling tepat!

- 1) Ketika analis bermaksud mencari informasi yang spesifik dan tepat, analis harus merancang pertanyaan-pertanyaan wawancara yang bersifat ....
  - A. tertutup
  - B. terbuka
  - C. menyelidik
  - D. semua jawaban A, B dan C benar
- 2) Pertanyaan berikut ini, yang termasuk pertanyaan menyelidik adalah ....
  - A. Sebagai petugas administrasi perpustakaan, apa saja masalah yang Anda hadapi selama ini dalam mengelola koleksi pustaka menggunakan cara manual?
  - B. Bagaimana tata cara anggota perpustakaan meminjam koleksi pustaka?
  - C. Dapatkah Anda menjelaskan lebih rinci mengenai masalah apa saja yang Anda hadapi dalam mengelola koleksi pustaka menggunakan cara manual?
  - D. Berapa banyak koleksi pustaka berjenis jurnal yang tersedia pada perpustakaan ini?

- 3) Urutan kegiatan yang logis dalam sesi JAD adalah ....
- Merancang Sesi JAD – Memilih Peserta – Mempersiapkan Sesi JAD – Melaksanakan Sesi JAD – Tindak Lanjut Pasca JAD
  - Memilih Peserta – Merancang Sesi JAD – Mempersiapkan Sesi JAD – Melaksanakan Sesi JAD – Tindak Lanjut Pasca JAD
  - Memilih Peserta – Mempersiapkan Sesi JAD – Merancang Sesi JAD – Melaksanakan Sesi JAD – Tindak Lanjut Pasca JAD
  - Merancang Sesi JAD – Mempersiapkan Sesi JAD – Memilih Peserta – Melaksanakan Sesi JAD – Tindak Lanjut Pasca JAD
- 4) Tiga fungsi utama fasilitator JAD, *kecuali* ....
- memastikan kelompok yang difasilitasi tetap berpegang pada agenda
  - membantu kelompok memahami istilah teknis dan istilah khusus yang melingkupi proses pengembangan sistem
  - mencatat masukan atau usulan kelompok pada papan tulis, bagan flip, atau tampilan komputer
  - bergabung sebagai peserta diskusi, agar dapat memberikan ide atau pendapat ketika rapat mengalami kebuntuan (*deadlock*)
- 5) Desain kuesioner yang baik adalah yang memenuhi unsur ....
- mulai pertanyaan dengan pertanyaan yang menarik
  - tidak meletakkan hal-hal yang penting di akhir kuesioner
  - menyertakan nomor pertanyaan untuk menghindari kebingungan
  - semua jawaban A, B dan C benar
- 6) Beberapa hal penting perlu diperhatikan dalam memilih teknik elisitasi kebutuhan, *kecuali* ....
- keterlibatan fasilitator
  - jenis informasi yang akan diperoleh
  - kedalaman, keluasan, dan integrasi informasi
  - keterlibatan pengguna
- 7) Jenis teknik elisitasi kebutuhan yang tergolong berbiaya rendah adalah ....
- kuesioner
  - observasi
  - JAD
  - jawaban A dan B benar
- 8) Strategi analisis kebutuhan yang paling mudah digunakan adalah "analisis masalah". Namun demikian strategi "analisis masalah" memiliki keterbatasan, yaitu ....
- perbaikan sistem cenderung hanya kecil dan bertahap
  - sulit untuk mengidentifikasi manfaat finansial yang signifikan dari sistem baru

- C. solusi penyelesaian masalah hanya mengatasi gejala dari masalah, dan bukan menyelesaikan masalah sebenarnya atau akar penyebab masalah.
- D. semua pilihan jawaban benar
- 9) Analisis masalah dan analisis akar penyebab cenderung paling berguna dalam situasi fokus proyek yang luas, di mana efisiensi tidak menjadi tujuan utamanya. Pernyataan tersebut adalah ....
- A. Benar
- B. Salah
- 10) Analisis durasi dan penghapusan aktivitas membantu tim menemukan proses bisnis yang dianggap tidak efektif sehingga proses tersebut dapat dirancang ulang dan ditingkatkan. Pernyataan tersebut adalah ....
- A. Benar
- B. Salah

Cocokkanlah jawaban Anda dengan Kunci Jawaban Tes Formatif 2 yang terdapat di bagian akhir modul ini. Hitunglah jawaban yang benar. Kemudian, gunakan rumus berikut untuk mengetahui tingkat penguasaan Anda terhadap materi Kegiatan Belajar 2.

$$\text{Tingkat Penguasaan} = \frac{\text{Jumlah Jawaban yang Benar}}{\text{Jumlah Soal}} \times 100$$

Arti tingkat penguasaan

<70%	70% - 79%	80% - 89%	90% - 100%
kurang	cukup	baik	baik sekali

Apabila mencapai tingkat penguasaan 80% atau lebih, Anda dapat meneruskan dengan modul selanjutnya. **Bagus!** Jika masih di bawah 80%, Anda harus mengulangi materi Kegiatan Belajar 2, terutama bagian yang belum dikuasai.

## Kunci Jawaban Tes Formatif

### *Tes Formatif 1*

- 1) B
- 2) A
- 3) D
- 4) D
- 5) B
- 6) C
- 7) B
- 8) A
- 9) B
- 10) B

### *Tes Formatif 2*

- 1) A
- 2) C
- 3) B
- 4) D
- 5) D
- 6) A
- 7) C
- 8) D
- 9) B
- 10) A

## Glosarium

Benchmarking	: Kegiatan mempelajari bagaimana organisasi lain melakukan proses bisnis, untuk menjadi rujukan bagi organisasi untuk dapat melakukan sesuatu yang lebih baik
Elisitasi kebutuhan	: aktivitas yang ditujukan untuk menemukan kebutuhan suatu sistem melalui komunikasi dengan pelanggan, pengguna sistem dan pihak lain yang memiliki kepentingan dalam pengembangan system
Fasilitator	: Orang yang memandu diskusi, namun tidak bergabung sebagai peserta, dan bertindak netral selama sesi diskusi JAD ( <i>Joint Application Development</i> )
Kebutuhan (persyaratan)	: Pernyataan mengenai apa yang harus dilakukan atau apa yang perlu dimiliki oleh sistem yang sedang dikembangkan
Pengguna Bisnis	: Pihak manajemen organisasi sebagai pemilik sistem yang dikembangkan
Pernyataan Kebutuhan	: Naskah yang mencantumkan kebutuhan fungsional dan nonfungsional dalam format garis besar
Pertanyaan menyelidik	: Pertanyaan lanjutan (tindaklanjut) mengenai hal yang telah dibahas melalui salah satu teknik elisitasi kebutuhan
Pertanyaan terbuka	: Pertanyaan yang memberikan ruang untuk dijabarkan oleh responden pada proses wawancara atau angket dalam kegiatan elisitasi kebutuhan
Pertanyaan tertutup	: Pertanyaan yang tidak memberikan ruang untuk dijabarkan oleh responden pada proses wawancara atau angket dalam kegiatan elisitasi kebutuhan
Responden	: Orang yang memberikan tanggapan dan informasi (baik secara langsung maupun tidak langsung) terkait data yang dibutuhkan oleh analis melalui teknik elisitasi kebutuhan

- Sistem yang ada (existing system) : Sistem yang sedang beroperasi saat ini, baik berupa sistem manual maupun sistem yang berbasis teknologi informasi
- Stakeholder : Pihak-pihak yang memiliki kepentingan dalam proyek pengembangan sistem informasi
- User/End user : Staf atau kelompok staf yang akan berinteraksi langsung dengan sistem yang dikembangkan
- Validasi kebutuhan : Proses pembuktian bahwa kebutuhan benar-benar mendefinisikan sistem yang diinginkan pengguna
- Wawancara bottom-up : Menggali masalah dari yang detail ke yang lebih umum dalam proses elisitasi kebutuhan
- Wawancara top-down : Menggali masalah dari yang umum ke yang lebih detail dalam proses elisitasi kebutuhan

## Daftar Pustaka

- Dennis, A. R., Hayes, G. S., & Daniels Jr, R. M. (1999). Business process modeling with group support systems. *Journal of Management Information Systems*, 15(4), 115-142.
- Dennis, A., Wixom, B. H., & Roth, R.M. (2012). *Systems analysis & design*. 5th Edition. United States of America: John Wiley & Sons, Inc.
- Franke, R. H., & Kaul, J. D. (1978). The Hawthorne experiments: First statistical interpretation. *American sociological review*, 623-643.
- Jogiyanto, H.M. (2008). *Analisis & desain sistem informasi: Pendekatan terstruktur teori dan praktik aplikasi bisnis*, Edisi 3. Yogyakarta: ANDI.
- Mensah, K. E. (2003). *Software development failures: Anatomy of failed projects*. MIT Press.
- Sommerville, I. (2016). *Software engineering*. 10th Edition. USA: Pearson.
- Wood, J., & Silver, D. (1995). *Joint application development*. 2nd Edition. New York: John Wiley & Sons.

**MSIM4302**  
**Edisi 1**

**MODUL 04**

# **Pemodelan Analisis dengan Pendekatan Terstruktur**

Bahar, S.T., M.Kom.

## Daftar Isi

<b>Modul 04</b>	
<b>4.1</b>	
Pemodelan Analisis dengan Pendekatan Terstruktur	
<b>Kegiatan Belajar 1</b>	4.5
Data Flow Diagram	
Latihan	4.32
Rangkuman	4.36
Tes Formatif 1	4.37
<b>Kegiatan Belajar 2</b>	4.40
Bagan Alir	
Latihan	4.49
Rangkuman	4.52
Tes Formatif 2	4.53
<b>Kunci Jawaban Tes Formatif</b>	4.56
<b>Glosarium</b>	4.57
<b>Daftar Pustaka</b>	4.59



## Pendahuluan

Modul 3 telah membahas beberapa persyaratan kegiatan elisitasi, seperti wawancara, JAD, dan lain-lain. Pada Modul 4 ini akan dibahas bagaimana definisi kebutuhan dapat diklarifikasi lebih lanjut melalui model proses sistem.

Kebutuhan user harus dituliskan dalam bahasa alami (natural) karena harus dapat dimengerti oleh orang-orang non teknik. Meskipun demikian, kebutuhan sistem yang lebih rinci memungkinkan untuk dinyatakan dengan cara yang lebih teknis. Salah satu teknik yang banyak digunakan adalah pendokumentasian kebutuhan sistem sebagai satu set model proses sistem. Model-model ini merupakan representasi grafis yang mendeskripsikan sistem yang sedang berjalan (sistem yang akan digantikan atau diperbaiki) dan menspesifikasi sistem yang dibutuhkan (dikembangkan) (Sommerville, 2016). Model dengan menggunakan representasi grafis, seringkali lebih dipahami daripada mendeskripsikannya dengan bahasa alami yang rinci tentang kebutuhan sistem. Representasi ini menjadi jembatan antara proses analisis dan perancangan sistem.

Pendekatan terstruktur adalah suatu proses yang mengimplementasikan urutan langkah dalam menyelesaikan suatu masalah. Teknik terstruktur juga merupakan pendekatan formal untuk memecahkan masalah-masalah dalam aktivitas bisnis menjadi bagian-bagian kecil yang dapat diatur dan saling berhubungan untuk selanjutnya dapat disatukan kembali menjadi satu kesatuan yang dapat dipergunakan untuk memecahkan masalah. Pendekatan terstruktur berorientasi pada fungsi. Dekomposisi permasalahan dilakukan berdasarkan fungsi atau proses secara hierarki, mulai dari konteks sampai proses-proses yang paling kecil.

Kelebihan utama pendekatan terstruktur adalah: (1) merupakan pendekatan visual dengan menggunakan analisis grafis, sehingga mudah dimengerti oleh pengguna sistem atau programmer; (2) merupakan pendekatan yang diketahui secara umum pada berbagai industri, karena sudah diterapkan begitu lama sehingga pendekatan ini sudah matang dan layak untuk digunakan. Adapun kekurangannya adalah: (1) berorientasi pada proses, sehingga mengabaikan kebutuhan non-fungsional; (2) tidak dapat memenuhi kebutuhan pemrograman berorientasi objek, karena model ini memang didesain khusus untuk mendukung konsep pemrograman terstruktur.

Model proses telah menjadi bagian dari teknik analisis sistem terstruktur selama bertahun-tahun (Dennis, 2012). Saat ini model proses masih digunakan karena kemampuannya untuk mengklarifikasi kebutuhan pengguna dengan cara yang mudah dimengerti. Namun demikian, saat ini mungkin terdapat organisasi yang kurang menekankan pada pemodelan proses daripada di masa lalu. Sebuah organisasi yang berusaha untuk membuat model proses, mungkin tidak menemukan bahwa model proses menambah banyak pemahaman mereka tentang sistem yang sedang dikembangkan. Organisasi lainnya menemukan pemodelan proses menjadi bagian yang

#### 4.4 Pemodelan Analisis dengan Pendekatan Terstruktur

---

menguntungkan dari hasil tahap analisis yang mereka lakukan. Perlu diingat bahwa tujuan pemodelan analisis adalah untuk dapat menggunakan serangkaian alat dan teknik yang akan membantu analis memahami dan mengklarifikasi apa yang harus dilakukan sistem baru sebelum sistem baru benar-benar dibangun.

Ada banyak teknik pemodelan proses yang digunakan saat ini, seperti: *Data Flow Diagram/DFD*, *Entity Relationship Diagram/ERD*, *State Transition Diagram/STD*, *Structured Chart*, *Diagram Warnier/Orr*, *Flowchart*, dan lain-lain (Jogiyanto, 2008). Pembahasan dalam modul ini akan difokuskan pada teknik yang paling umum digunakan: Diagram Arus Data (*Data Flow Diagram/DFD*) dan Bagan Alir (*Flowchart*). Meskipun Diagram Arus Data (DAD) kelihatannya berfokus pada data, namun bukan ini yang dimaksud. Fokus DAD terutama pada proses atau kegiatan yang dilakukan dan bukan pada Pemodelan Data. Pemodelan Data akan dibahas pada modul mendatang, yang menyajikan bagaimana data yang dibuat diatur dan digunakan oleh proses.

Setelah mempelajari modul ini mahasiswa diharapkan secara lebih rinci mampu:

1. menjelaskan konsep pendekatan terstruktur dalam pemodelan analisis;
2. menjelaskan kegunaan *Data Flow Diagram* sebagai tools pemodelan terstruktur;
3. menjelaskan komponen-komponen *Data Flow Diagram*;
4. menggunakan *Data Flow Diagram* untuk menjelaskan Proses Bisnis Sistem (Pemodelan Analisis);
5. membuat *Data Flow Diagram* untuk memodelkan proses sistem;
6. menjelaskan Bagan Alir (*Flowchart*) sebagai tools pendukung *Data Flow Diagram* dalam menjelaskan logika pemrosesan;
7. menjelaskan jenis-jenis Bagan Alir dan fungsinya masing-masing;
8. menjelaskan simbol-simbol dalam Bagan Alir;
9. membuat Bagan Alir untuk memodelkan prosedur logika proses dan atau prosedur logika program.

Kegiatan  
Belajar

1

## Data Flow Diagram

**D**iagram Arus Data (DAD) atau *Data Flow Diagram* (DFD) paling sering digunakan untuk menggambarkan suatu sistem yang sedang berjalan atau sistem baru yang akan dikembangkan secara logika tanpa mempertimbangkan lingkungan fisik di mana data tersebut mengalir (misalnya, lewat telepon, surat, dan sebagainya), apakah suatu proses terkomputerisasi atau manual, apakah informasi dikumpulkan melalui formulir kertas atau melalui Web, atau lingkungan fisik di mana data tersebut akan disimpan (misalnya, magnetik tape, harddisk, compact disk, dan lain sebagainya). Namun demikian, DFD juga dapat digunakan untuk menggambarkan sistem secara fisik. Rincian fisik ini didefinisikan pada fase desain ketika model logis ini telah disempurnakan menjadi model fisik, yang memberikan informasi yang diperlukan untuk membangun sistem. Dengan berfokus pada proses logis terlebih dahulu, analis dapat fokus pada bagaimana bisnis harus dijalankan, tanpa terganggu oleh detail implementasinya. DFD merupakan alat yang cukup populer, karena dapat menggambarkan arus data di dalam sistem dengan terstruktur dan jelas. DFD juga merupakan dokumentasi dari sistem yang baik.

Jogiyanto (2008) dan Dennis (2012) mengklasifikasikan dua bentuk DFD, yaitu *Logical Data Flow Diagram* (LDFD) dan *Physical Data Flow Diagram* (PDDF). DFD logika lebih menekankan proses-proses apa yang terdapat di sistem, sedangkan DFD fisik lebih menekankan pada bagaimana proses dari sistem diterapkan.

Selama fase analisis, analis sistem mengidentifikasi proses dan aliran data yang diperlukan untuk mendukung kebutuhan fungsional sistem baru. Proses dan aliran data ini terdapat pada DFD logis untuk sistem yang akan datang. Analis menghindari membuat keputusan implementasi selama analisis, dengan fokus pertama pada kebutuhan bisnis sistem. DFD logis tidak mengandung indikasi bagaimana sistem akan benar-benar diterapkan ketika sistem informasi dibangun, melainkan hanya menyatakan apa yang akan dilakukan sistem baru. Dengan cara ini, pengembang tidak terganggu oleh detail teknis dan tidak bias oleh keterbatasan teknis pada tahap awal pengembangan sistem. Pengguna bisnis lebih memahami diagram yang menunjukkan tampilan bisnis dari sistem.

Pada fase desain, model proses secara fisik dibuat untuk menunjukkan detail implementasi dan menjelaskan bagaimana sistem akhir bekerja. Rincian ini mencakup referensi ke teknologi aktual (tertentu), format informasi yang bergerak melalui proses,

dan interaksi manusia yang terlibat. DFD fisik (PDFD) dapat digunakan untuk mendefinisikan model-model yang menggambarkan karakteristik fisik sistem yang akan dibuat.

Kegiatan Belajar ini menjelaskan simbol-simbol DFD dan menjelaskan aturan dasarnya, kemudian menjelaskan proses membangun DFD yang mengambil informasi dari kasus penggunaan/*use cases* (bukan *use case diagram*), atau mengambil informasi dari prosedur bisnis dan dari informasi kebutuhan tambahan yang dikumpulkan dari pengguna selama fase elisitasi.

### A. KOMPONEN DFD

Ada empat simbol dalam DFD (proses, aliran data, penyimpanan data, dan entitas luar), yang masing-masing diwakili oleh simbol grafik yang berbeda. Ada dua gaya simbol yang umum digunakan (Dennis, 2012), yaitu yang dikembangkan oleh Chris Gane dan Trish Sarson dan lainnya oleh Tom DeMarco dan Ed Yourdon (Gambar 4.1).

Elemen DFD	Simbol Gane Dan Sarson	Simbol Demarco Dan Yourdon
<b>ENTITAS LUAR</b> <i>(External Entity)</i>	[Nama]	[Nama]
<b>PROSES</b> <i>(Process)</i>	[1] Nama	{Nama}
<b>ARUS DATA</b> <i>(Data Flow)</i>	Nama →	Nama →
<b>PENYIMPANAN DATA</b> <i>(Data Store)</i>	D1   Nama	D1 Nama

Gambar 4.1  
Elemen Data Flow Diagram (DFD)

Penggambaran DFD dapat menggunakan salah satu dari model penggambaran yang ada, namun demikian harus menggunakan simbol-simbol secara konsisten (kelompok simbol Sarson atau kelompok simbol Yourdon). Penulisan dalam modul ini akan menggunakan gaya Demarco dan Yourdon.

#### 1. Entitas Luar

Entitas luar (*external entity*) adalah orang atau sekelompok orang, suatu organisasi atau unit organisasi di luar organisasi, atau sistem yang berada di luar sistem yang dikembangkan tetapi berinteraksi langsung dengan sistem (misalnya: pelanggan,

pemasok, organisasi pemerintah, sistem akuntansi). Entitas luar biasanya dapat disamakan dengan aktor utama yang diidentifikasi dalam *use case*. Entitas luar menyediakan data ke sistem atau menerima data dari sistem, dan berfungsi untuk menetapkan batas-batas sistem. Setiap entitas luar memiliki nama dan deskripsi. Poin kunci yang perlu diingat tentang entitas luar adalah bahwa entitas luar untuk sistem, namun mungkin tidak menjadi bagian dari organisasi. Orang yang menggunakan informasi dari sistem untuk melakukan proses lain atau yang memutuskan informasi apa yang masuk ke dalam sistem juga didokumentasikan sebagai entitas luar (misalnya: manajer dan staf).

## 2. Proses

Proses adalah suatu kegiatan atau fungsi yang dilakukan dalam proses bisnis tertentu. Proses dapat dilakukan secara manual oleh orang atau mesin, dan dapat juga dilakukan oleh komputer. Setiap proses harus dinamai, dimulai dengan kata kerja (misalnya: menghitung, membuat, membandingkan, memverifikasi, mempersiapkan, merekam, dan lain sebagainya) dan diakhiri dengan kata benda (misalnya: menentukan jumlah permintaan). Nama proses harus pendek, namun mengandung informasi yang cukup, sehingga pembaca dapat dengan mudah memahami apa yang dilakukan oleh proses tersebut. Secara umum, setiap proses hanya melakukan satu aktivitas, sehingga sebagian besar analis sistem menghindari menggunakan kata *dan* dalam nama proses, karena itu menunjukkan bahwa proses melakukan beberapa aktivitas. Selain itu, setiap proses harus memiliki setidaknya satu aliran data *input* dan setidaknya satu aliran data *output*.

Contoh penggambaran proses yang benar:



Gambar 4.2  
Contoh Penggambaran Proses yang Benar

Setiap proses memiliki nomor identifikasi unik, nama, dan deskripsi, yang kesemuanya dicatat dalam repositori *case* (penjelasan atau deskripsi yang menyertainya). Deskripsi dengan jelas dan tepat menggambarkan langkah-langkah dan detail proses. Deskripsi digunakan untuk memandu para programmer yang mengkomputerisasi prosesnya, atau digunakan oleh para penulis manual kebijakan untuk jenis proses yang tidak terkomputerisasi. Deskripsi proses menjadi lebih terperinci

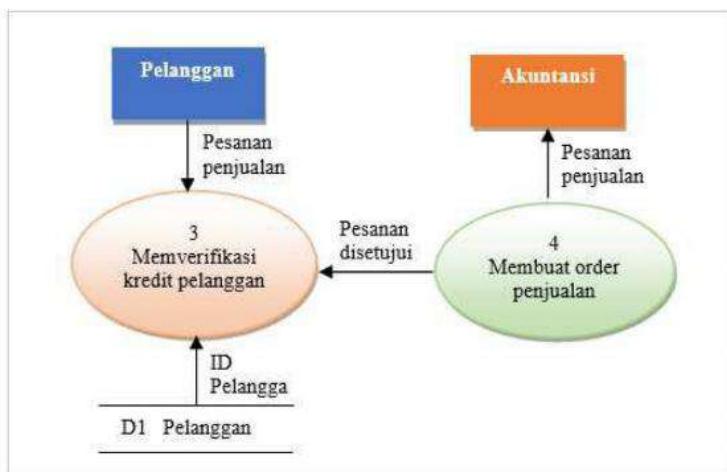
#### 4.8 Pemodelan Analisis dengan Pendekatan Terstruktur

karena informasi tentang proses dipelajari melalui fase analisis. Banyak deskripsi proses ditulis sebagai pernyataan teks sederhana tentang apa yang terjadi. Proses yang lebih kompleks menggunakan teknik yang lebih formal seperti *flowchart*, *structured english*, *decision tables*, atau *decision trees*.

Pada umumnya kesalahan penggambaran proses di DFD dapat berupa berikut.

- Proses mempunyai *input* tetapi tidak menghasilkan *output*. Kesalahan ini disebut dengan *black hole* (data terjebak), karena data masuk ke dalam proses dan lenyap tidak berbekas. Kesalahan tersebut tampak pada proses nomor 3 (Gambar 4.3).
- Proses menghasilkan *output* tetapi tidak pernah menerima *input* dan kesalahan ini disebut dengan *miracle* (mustahil), karena secara ajaib dihasilkan suatu *output* tanpa pernah menerima *input*. Kesalahan tersebut tampak pada proses nomor 4 (Gambar 4.3).

Contoh penggambaran proses yang salah:



Gambar 4.3  
Contoh Penggambaran Proses yang Salah

### 3. Arus Data

Arus data (*data flow*) adalah satu bagian data, kadang-kadang disebut elemen data (misalnya: jumlah yang tersedia), atau kumpulan beberapa informasi (misalnya: permintaan bahan kimia baru). Setiap arus data harus diberi nama yang jelas (mempunyai arti) dengan kata benda. Deskripsi arus data (biasanya menggunakan kamus data) mencantumkan dengan tepat elemen-elemen data yang mengalir. Misalnya, arus data pemberitahuan pengambilan dapat mencantumkan nama barang dan jumlah yang diminta sebagai elemen datanya.

Arus data adalah penghubung yang menyatukan proses. Salah satu ujung dari setiap arus data akan selalu datang dari atau menuju suatu proses, dengan panah yang menunjukkan arah masuk atau keluar dari proses. Arus data menunjukkan *input* apa

yang masuk ke setiap proses dan *output* apa yang dihasilkan setiap proses. Setiap proses harus membuat setidaknya satu arus data *output*, karena jika tidak ada *output*, proses tidak melakukan apa-apa. Demikian juga, setiap proses memiliki setidaknya satu aliran data *input*, karena tidak mungkin untuk menghasilkan *output* tanpa *input*.

Arus data dapat berbentuk sebagai berikut (Jogiyanto, 2008).

- a. Formulir atau dokumen yang digunakan di perusahaan.
- b. Laporan tercetak yang dihasilkan oleh sistem.
- c. Tampilan atau output di layar komputer yang dihasilkan oleh sistem.
- d. Masukan untuk komputer.
- e. Komunikasi ucapan.
- f. Surat-surat atau memo.
- g. Data yang dibaca atau direkamkan ke suatu file.
- h. Suatu isian yang dicatat pada buku agenda.
- i. Transmisi data dari suatu komputer ke komputer yang lain.
- j. Nama dari arus data dituliskan di samping garis panahnya.

*Contoh:*



Gambar 4.4  
Contoh Arus Data

Beberapa aturan dasar penggambaran arus data sebagai berikut.

- a. *Konsep Paket Data (Packet of Data)*

Menurut Jogiyanto (2008), bila dua atau lebih data mengalir dari suatu sumber yang sama ke tujuan yang sama, maka harus dianggap sebagai suatu arus data yang tunggal, karena dua atau lebih data tersebut mengalir bersama-sama sebagai suatu paket. Data yang mengalir bersama-sama harus ditunjukkan sebagai satu arus data, walaupun misalnya terdiri dari beberapa dokumen.

#### 4.10 Pemodelan Analisis dengan Pendekatan Terstruktur

*Contoh:*



Gambar 4.5  
Arus Data Tunggal dengan Dua Data yang Mengalir

Pendapat lain dikemukakan oleh Dennis (2012), yang menggambarkan dua buah data mengalir menggunakan dua arus data (Gambar 4.6).



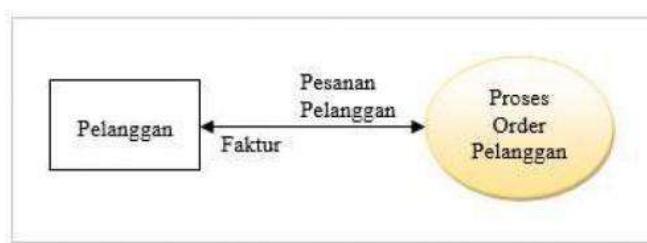
Gambar 4.6  
Dua Arus Data untuk Dua Data yang Mengalir

Jika terdapat arus data berlawanan, maka penggambaran dilakukan seperti pada Gambar 4.7.



Gambar 4.7  
Aliran Data Menggunakan Dua Arus Data Berlawanan

Arus data yang berlawanan dapat juga digambarkan seperti pada Gambar 4.8 (McLeod, 2004).

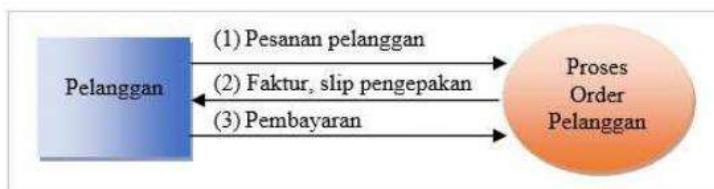


Gambar 4.8  
Aliran Data Berlawanan Menggunakan Satu Arus Data

Berdasarkan hal di atas, maka:

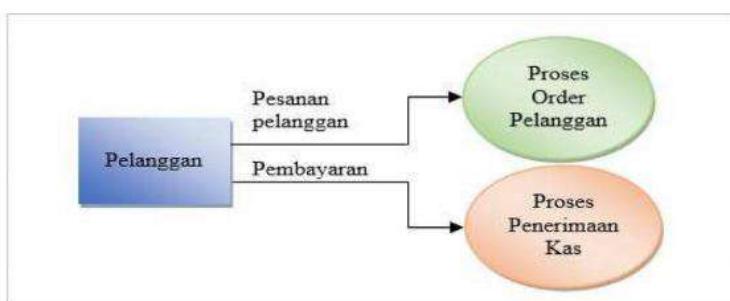
- 1) bila terdapat dua atau lebih data mengalir dari suatu sumber yang sama ke tujuan yang sama dalam satu waktu kejadian yang sama (misalnya: pengiriman *dokumen faktur* bersamaan dengan *dokumen slip pengepakan barang* dalam satu amplop ke pelanggan), maka harus dianggap sebagai suatu arus data yang tunggal (penggambarannya seperti pada Gambar 4.5). Namun jika dua atau lebih data mengalir dari suatu sumber yang sama ke tujuan yang sama dalam waktu kejadian yang berbeda (misalnya: pengiriman *dokumen pesanan pelanggan* dan *pembayaran pelanggan* dalam amplop yang berbeda sebagai akibat dari waktu pengiriman yang berbeda oleh pelanggan), maka harus digambarkan dengan arus data yang berbeda (seperti pada Gambar 4.6).
- 2) penggambaran lebih dari satu aliran data (baik yang menggunakan satu arus data maupun yang menggunakan lebih dari satu arus data) dapat juga disertai nomor proses (Gambar 4.9), sehingga aliran data dapat ditelusuri waktu kejadiannya (urutan prosesnya).

*Contoh:*



Gambar 4.9  
Arus Data yang Menggunakan Nomor Proses

Bila dua buah data dari sumber yang sama akan ditangani oleh dua proses yang berlainan (mempunyai tujuan yang berbeda), maka dapat digambarkan seperti pada Gambar 4.10.



Gambar 4.10  
Aliran Data dari Satu Sumber ke Tujuan yang Berbeda

#### 4.12 Pemodelan Analisis dengan Pendekatan Terstruktur

##### b. Konsep Arus Data Menyebar (Diverging Data Flow)

Arus data yang menyebar merupakan tembusan dari arus data yang sama dari sumber yang sama ke tujuan yang berbeda.

Contoh:



Gambar 4.11  
Arus Data Menyebar

Pada Gambar 4.11 terlihat bahwa arus data *pesananan penjualan* mempunyai 3 tembusan, yaitu *tembusan untuk jurnal* yang mengalir ke proses pembuatan faktur, *tembusan permintaan barang* yang mengalir ke entitas luar gudang, dan *tembusan kredit* yang mengalir ke proses verifikasi kredit. Konsep arus data yang menyebar ini menunjukkan bahwa arus data *tembusan jurnal*, *tembusan permintaan barang* dan *tembusan kredit* merupakan arus data yang mempunyai struktur elemen yang sama, karena merupakan hasil dari tembusan arus data pesanan penjualan.

##### c. Konsep Arus Data Mengumpul (Converging Data Flow)

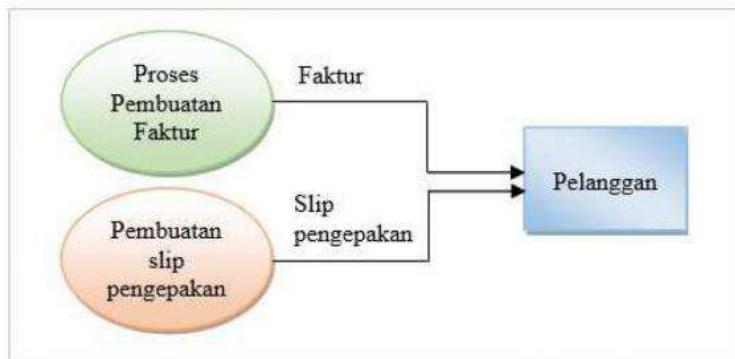
Arus data yang mengumpul adalah gabungan beberapa arus data dari sumber yang berbeda, secara bersama-sama menuju ke tujuan yang sama. Arus data *pengiriman barang* merupakan berkas *faktur* yang digabungkan dengan *slip pengepakan barang*.

Contoh:



Gambar 4.12  
Arus Data Mengumpul

Arus data mengumpul ini jarang dibuat di DFD dan sebagai penggantinya dapat digambarkan seperti pada Gambar 4.13.



Gambar 4.13  
Arus Data Mengumpul

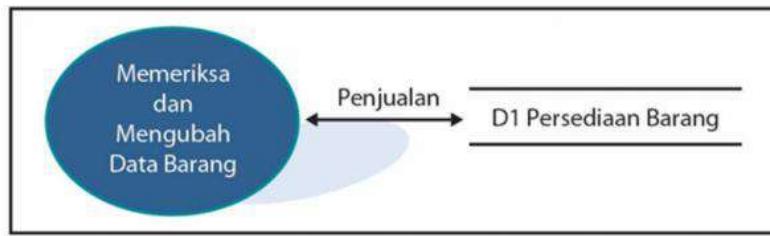
#### 4. Simpanan Data (*Data Store*)

Penyimpanan data adalah kumpulan data yang disimpan dalam beberapa cara (yang ditentukan kemudian saat membuat model fisik). Setiap penyimpanan data dinamai dengan kata benda dan diberi nomor identifikasi (kode D). Penyimpanan data sebagai bentuk titik awal untuk model data (dibahas dalam modul berikutnya) dan merupakan penghubung utama antara model proses dan model data. Simpanan data dapat berupa:

- suatu file atau database di sistem komputer;
- suatu arsip atau catatan manual;
- suatu kotak tempat data di meja seseorang ;
- suatu tabel acuan manual;
- suatu agenda atau buku.

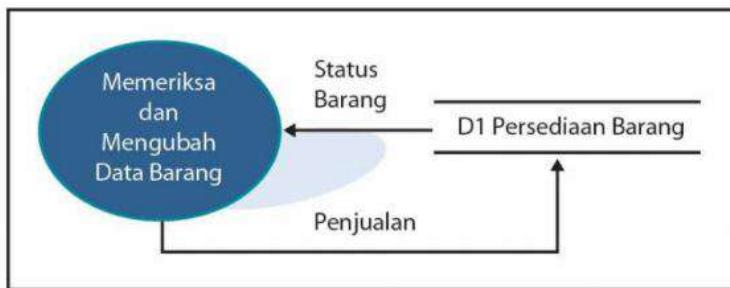
Aliran data yang keluar dari penyimpanan data menunjukkan bahwa informasi diambil dari penyimpanan data, sedangkan aliran data yang masuk ke penyimpanan data menunjukkan bahwa informasi ditambahkan ke penyimpanan data. Aliran data yang masuk dan keluar dari penyimpanan data menunjukkan bahwa informasi pada penyimpanan data diubah (misalnya: dengan mengambil data dari penyimpanan data, mengubahnya, dan menyimpannya kembali).

Semua penyimpanan data harus memiliki setidaknya satu aliran data input, demikian juga memiliki setidaknya satu aliran data keluaran. Secara logis, untuk apa menyimpan data jika tidak pernah menggunakannya? Sebaliknya, suatu hal yang mustahil jika mengambil data dari sebuah penyimpanan data tanpa pernah mengisi penyimpanan data tersebut. Dalam kasus di mana proses yang sama menyimpan data dan mengambil data dari penyimpanan data, seringkali digambarkan dengan satu data mengalir dengan panah di kedua ujungnya (Gambar 4.14).



Gambar 4.14  
Update Data dengan Panah di Kedua Ujung Aliran Data

Namun, menurut Dennis (2012) praktik ini tidak benar. Aliran data yang menyimpan data dan aliran data yang mengambil data harus selalu ditampilkan sebagai dua aliran data terpisah (Gambar 4.15). Berbeda halnya dengan Dennis (2012), Jogiyanto (2008) berpandangan bahwa untuk suatu proses yang melakukan kedua-duanya (menggunakan dan *update* simpanan data) dapat dipilih salah satu (Gambar 4.14 atau Gambar 4.15).

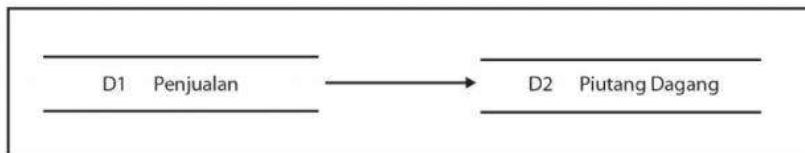


Gambar 4.15  
Update Data dengan Dua Aliran Data Terpisah

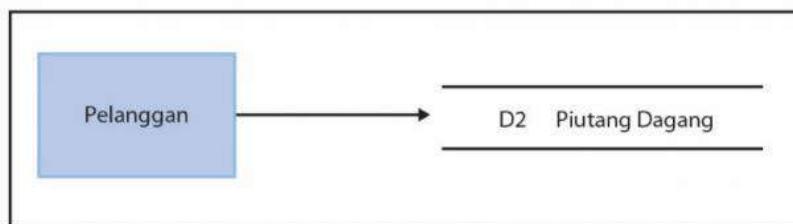
Dalam penggambaran simpanan data, perlu diperhatikan beberapa hal sebagai berikut.

- Hanya proses saja yang berhubungan dengan simpanan data, karena yang menggunakan atau mengubah data pada simpanan data adalah suatu proses.
- Update* dapat berupa proses:
  - menambah atau menyimpan record baru atau dokumen baru ke dalam simpanan data;
  - menghapus record atau mengambil dokumen dari simpanan data;
  - mengubah nilai data di suatu record atau di suatu dokumen yang ada di simpanan data.

Contoh penggambaran proses penyimpanan data yang salah disajikan pada Gambar 4.16 dan Gambar 4.17.



Gambar 4.16  
Koneksi Antar Penyimpanan Data

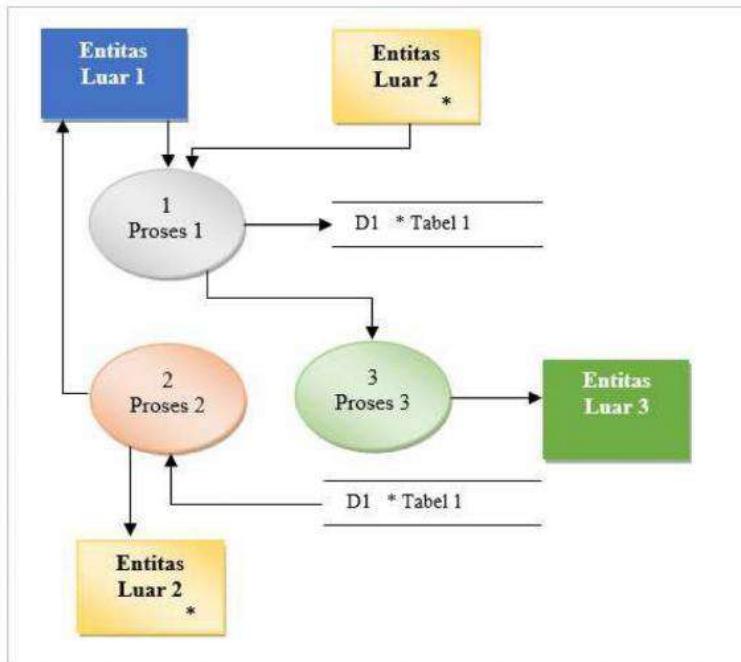


Gambar 4.17  
Koneksi Entitas Luar dengan Penyimpanan Data

Agar garis arus data tertata rapi (tidak saling berpotongan sehingga tampak menjadi ruwet), maka sebuah simpanan data atau entitas luar dapat digambar lebih dari satu kali pada tempat yang berbeda dalam satu gambar, seperti pada Gambar 4.18.

Penjelasan mengenai duplikasi simpanan data dan entitas luar pada Gambar 4.18 adalah sebagai berikut:

Misalkan *Proses 1* menyimpan data ke D1, dan *Proses 2* akan menggunakan data pada D1. Dalam kondisi normal, ketika *Proses 2* akan menggunakan data dari D1, maka dibuat garis arus dari D1 (yang berdekatan dengan *Proses 1*) menuju ke *Proses 2*, dan itu akan memotong beberapa garis arus yang lainnya. Agar garis arus tidak saling berpotongan sehingga tampak menjadi ruwet, D1 digambar kembali pada lokasi yang berdekatan dengan *Proses 2*, sehingga lebih mudah membuat garis arus dari D1 ke *Proses 2*. Tanda "\*\*\*" (satu buah) pada D1 menandakan bahwa tabel tersebut muncul (digambar) lebih dari satu kali pada lokasi yang berbeda dalam gambar. Jika terdapat Tabel yang lainnya yang mengalami kasus seperti D1 (misalnya: D3), maka tabel tersebut diberi tanda "\*\*\*" (dua buah) sebagai pertanda bahwa tabel tersebut merupakan tabel ke-2 yang juga muncul (di gambar) lebih dari satu kali, demikian seterusnya. Adapun tabel yang penggambarannya tidak duplikat, tidak perlu diberi tanda "\*\*\*".



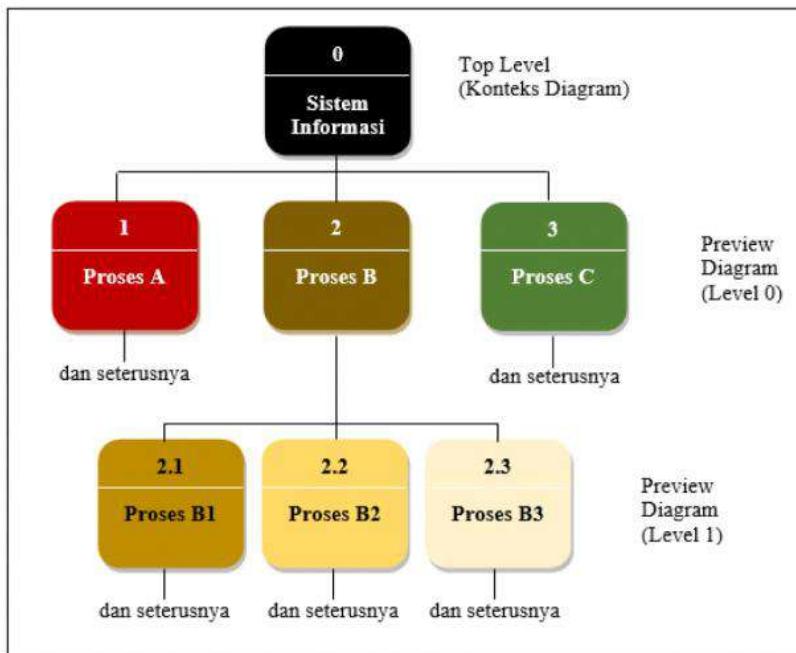
Gambar 4.18  
Duplikasi Simpanan Data dan Entitas Luar

Prosedur menggambar duplikasi pada entitas luar sama seperti prosedur menggambar duplikasi penyimpanan data. Sebagai contoh (pada gambar 4.18), *Proses 1* menerima masukan dari *Entitas Luar 2*. Pada keadaan lain, *Proses 2* menghasilkan luaran menuju ke *Entitas Luar 2*. Agar garis arus tidak saling berpotongan sehingga tampak menjadi ruwet, *Entitas Luar 2* digambarkan berulang pada lokasi yang berdekatan dengan *Proses 2*, dengan memberikan kode "\*" seperti tata cara penggambaran duplikat pada penyimpanan data.

## B. MENGGUNAKAN *DATA FLOW DIAGRAM* UNTUK MENJELASKAN PROSES BISNIS

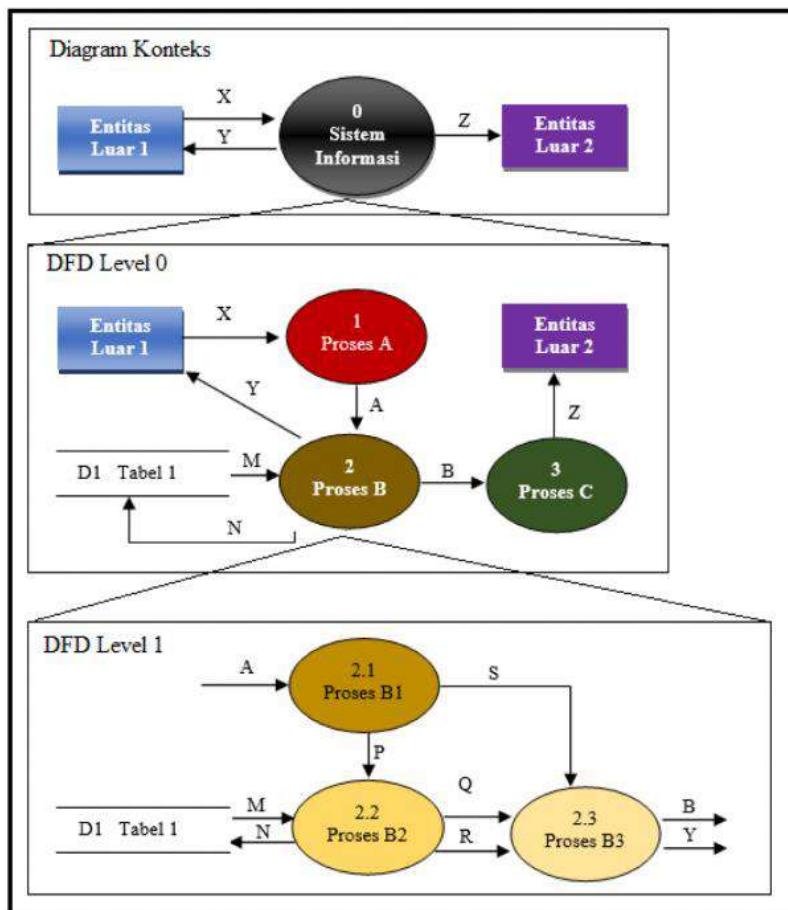
Sebagian besar proses bisnis terlalu rumit untuk dijelaskan dalam satu *Data Flow Diagram* (DFD). Oleh karena itu sebagian besar model proses terdiri dari satu set DFD. DFD pertama menyediakan ringkasan dari keseluruhan sistem, dengan DFD tambahan memberikan lebih banyak dan lebih detail tentang setiap bagian dari keseluruhan proses bisnis. Dengan demikian, satu prinsip penting dalam pemodelan proses dengan DFD adalah *dekomposisi* proses bisnis menjadi hierarki DFD (Jogiyanto, 2008), dengan setiap level di bawah hierarki mewakili ruang lingkup lebih sedikit tetapi lebih detail. Istilah lain untuk *dekomposisi* proses adalah *breakdown* sebuah proses ke level-level

selanjutnya untuk mendetailkan proses tersebut (Sulianta, 2017). Gambar 4.19 menunjukkan bagan berjenjang (*hierarchy chart*) yang mengilustrasikan bagaimana satu proses bisnis dapat didekomposisi menjadi beberapa level DFD.



Gambar 4.19  
Bagan Berjenjang

Berdasarkan Gambar 4.19 dilakukan dekomposisi DFD, dimulai dengan penggambaran *top level* (diagram konteks), selanjutnya penggambaran diagram level 0, level 1, dan seterusnya (Dennis, 2012). Jumlah level pada DFD tergantung model dekomposisi level pada *hierarchy chart*.



Gambar 4.20  
Hubungan Antar Level pada DFD

### 1. Diagram Konteks

DFD pertama dalam setiap model proses bisnis, baik sistem manual atau sistem terkomputerisasi, adalah diagram konteks (lihat Gambar 4.20). Diagram konteks menunjukkan seluruh sistem dalam konteks dengan lingkungannya. Semua model proses memiliki satu diagram konteks.

Diagram konteks menunjukkan keseluruhan proses bisnis hanya sebagai satu proses (yaitu sistem itu sendiri) dan menunjukkan aliran data ke dan dari entitas luar. Penyimpanan data biasanya tidak termasuk dalam diagram konteks, kecuali jika penyimpanan data itu *dimiliki* oleh sistem lain. Misalnya, sistem informasi yang digunakan oleh perpustakaan universitas yang mencatat siapa yang telah meminjam buku kemungkinan akan memeriksa database sistem informasi pendaftar siswa (sebagai sistem lain) untuk melihat apakah seorang siswa saat ini terdaftar di universitas. Dalam diagram konteks ini, penyimpanan data sistem informasi pendaftar siswa dapat ditampilkan pada diagram konteks karena itu merupakan eksternal untuk sistem perpustakaan, tetapi digunakan oleh sistem perpustakaan. Namun, banyak organisasi

menganggap ini sebagai *entitas luar* yang disebut *Sistem Informasi Registrasi Mahasiswa*, bukan sebagai penyimpan data.

## 2. Diagram Level 0

DFD berikutnya disebut diagram level 0 atau DFD level 0 (lihat Gambar 4.20). Diagram level 0 menunjukkan semua proses pada tingkat penomoran pertama (yaitu proses nomor 1 hingga 3), penyimpanan data, entitas luar, dan aliran data di antara mereka. Tujuan dari DFD level 0 adalah untuk menunjukkan semua proses tingkat tinggi utama dari sistem dan bagaimana mereka saling terkait. Semua model proses memiliki satu dan hanya satu DFD level 0.

Prinsip kunci lain dalam membuat DFD adalah keseimbangan. Menyeimbangkan berarti memastikan bahwa semua informasi yang disajikan dalam DFD di satu tingkat secara akurat disajikan dalam DFD tingkat berikutnya. Mari kita bandingkan diagram konteks dengan DFD level 0 pada Gambar 4.20 untuk melihat bagaimana keduanya seimbang. Terlihat bahwa entitas luar (1, 2) identik antara dua diagram (diagram konteks dan DFD Level 0) dan bahwa data mengalir ke dan dari entitas luar dalam diagram konteks (X, Y, Z) juga muncul di DFD level 0. DFD level 0 menggantikan proses tunggal diagram konteks (selalu bernomor 0) dengan tiga proses (A, B, C), menambahkan penyimpanan data (D1), dan mencakup dua aliran data tambahan yang tidak ada dalam diagram konteks (aliran data) A dari proses 1 ke proses 2; aliran data B dari proses 2 ke proses 3.

Tiga proses dan dua aliran data ini terdapat dalam proses 0. Mereka tidak ditampilkan pada diagram konteks karena mereka adalah komponen internal proses 0. Diagram konteks sengaja menyembunyikan beberapa kompleksitas sistem untuk membuatnya lebih mudah dimengerti oleh pembaca. Hanya setelah pembaca memahami diagram konteks, analis *membuka* proses 0 untuk menampilkan operasi internalnya dengan mendekomposisi diagram konteks ke DFD level 0, yang menunjukkan lebih detail tentang proses dan aliran data di dalam sistem.

## 3. Diagram Level 1

Dengan cara yang sama dengan diagram konteks yang dengan sengaja menyembunyikan beberapa kompleksitas sistem, DFD level 0 juga demikian. DFD level 0 hanya menunjukkan interaksi proses tingkat tinggi utama dalam sistem. Setiap proses pada DFD level 0 dapat didekomposisi menjadi DFD yang lebih eksplisit, yang disebut diagram level 1 (DFD level 1), yang menunjukkan cara kerjanya secara lebih rinci. DFD yang diilustrasikan pada Gambar 4.20 hanya DFD pada level 1.

Secara umum, semua model proses memiliki diagram level 1 sebanyak yang ada pada proses diagram level 0; setiap proses di DFD level 0 akan didekomposisi menjadi DFD level 1 secara tersendiri, sehingga DFD level 0 pada Gambar 4.20 akan memiliki tiga DFD level 1 (satu untuk proses A, satu untuk proses B, dan satu untuk proses C). Untuk mempermudah, pada contoh ini hanya menunjukkan 1 DFD level 1, yaitu DFD

untuk proses 2 (Proses B). Proses di DFD level 1 diberi nomor berdasarkan proses yang diuraikan. Dalam contoh ini, yang diuraikan adalah proses 2, sehingga proses di DFD level 1 ini diberi nomor 2.1, 2.2, dan 2.3.

Proses 2.1, 2.2, dan 2.3 adalah anak-anak dari proses 2, atau proses 2 adalah induk dari proses 2.1, 2.2, dan 2.3. Sebagai ilustrasi, himpunan anak-anak dan orang tua mesti identik; mereka hanya berbeda dalam cara pandang terhadap suatu hal yang sama. Ketika proses induk didekomposisi menjadi anak-anak, anak-anaknya harus benar-benar melakukan semua fungsi induknya, dengan cara yang sama seperti memotong roti menghasilkan satu set irisan yang sepenuhnya membentuk roti secara utuh. Meskipun irisan mungkin tidak berukuran sama, set irisan identik dengan seluruh roti; tidak ada yang dihilangkan ketika mengiris roti.

Sekali lagi, sangat penting untuk memastikan bahwa DFD level 0 dan level 1 seimbang. DFD level 0 menunjukkan bahwa proses 2 (proses B) mengakses penyimpanan data D1 melalui aliran data M, memiliki satu aliran data input (A), dan memiliki tiga aliran data output (B, N, dan Y). Pemeriksaan pada DFD level 1 menunjukkan penyimpanan data dan aliran data yang sama. Sekali lagi, kita melihat bahwa empat aliran data baru telah ditambahkan (P, Q, R, S) pada tingkat ini. Aliran data ini terkandung dalam proses 2 (internal) dan karenanya tidak didokumentasikan dalam DFD level 0. Hanya ketika kita mendekomposisi atau membuka proses 2 melalui DFD level 1 kita dapat melihat bahwa keempat aliran data baru tersebut ada.

DFD level 1 menunjukkan lebih tepatnya proses mana yang menggunakan aliran data input A (proses 2.1/B1) dan yang menghasilkan aliran data output B dan Y (proses 2.3/B3). Namun jika diperhatikan, DFD level 1 tidak menunjukkan dari mana aliran data ini berasal atau pergi. Untuk menemukan sumber aliran data A, kita harus naik ke level 0 DFD, yang menunjukkan aliran data A yang berasal dari entitas luar 1. Demikian juga, jika kita mengikuti aliran data dari B di DFD level 0, hanya terlihat menuju ke proses 3 (C), tetapi tidak diketahui persis proses mana dalam proses 3 yang menggunakan (misalnya: proses 3.1, 3.2, atau lainnya). Untuk memastikan itu, kita harus memeriksa DFD level 1 untuk proses 3 (C), demikian seterusnya.

Contoh ini menunjukkan satu kelemahan pada dekomposisi DFD jika digambar di beberapa halaman. Untuk menemukan sumber dan tujuan pasti dari aliran data, orang harus sering mengikuti aliran data di beberapa DFD pada halaman yang berbeda. Beberapa alternatif penguraian DFD untuk pendekatan ini telah diusulkan. Alternatif paling umum adalah menunjukkan sumber dan tujuan aliran data ke dan dari entitas luar (serta penyimpanan data) di DFD tingkat bawah. Fakta bahwa menunjukkan aliran data ke atau dari penyimpanan data dan entitas luar, dapat secara signifikan menyederhanakan pembacaan beberapa halaman DFD, daripada melihat proses pada halaman DFD lainnya. Hanya saja, diperlukan area yang lebih luas untuk menggambarkan keseluruhan proses pada setiap level dalam satu area gambar.

#### 4. Deskripsi Proses

Tujuan deskripsi proses adalah untuk menjelaskan apa yang dilakukan proses dan memberikan informasi tambahan yang tidak diberikan oleh DFD. Ketika melalui fase demi fase dalam SDLC, analis secara bertahap beralih dari deskripsi kebutuhan secara umum menjadi deskripsi yang lebih detail dan selanjutnya diterjemahkan ke dalam bahasa pemrograman dengan tepat. Dalam kebanyakan kasus, suatu proses cukup mudah sehingga dengan adanya *definisi kebutuhan*, *kasus penggunaan/use case* (bukan *use case diagram*), dan *DFD dengan deskripsi teks sederhana* secara bersama-sama dapat memberikan detail proses yang cukup untuk mendukung kegiatan dalam fase desain. Namun, kadang-kadang terdapat suatu proses yang cukup rumit sehingga harus dituangkan dalam deskripsi proses yang lebih rinci yang menjelaskan logika yang terjadi di dalam proses. Beberapa teknik yang biasa digunakan untuk menggambarkan logika pemrosesan yang lebih kompleks adalah: *structured english (pseudocode)* atau *structured Bahasa Indonesia*, *decision trees*, dan *decision tables*. Proses yang sangat kompleks dapat menggunakan kombinasi *structured english* dan *decision trees* atau *decision tables*. Teknik lain yang dapat digunakan untuk menggambarkan struktur logika secara lebih rinci dari suatu proses adalah diagram alir (*flowchart*). Teknik-teknik tersebut berguna bagi analis sistem dalam menyampaikan pemahaman yang tepat tentang apa yang terjadi *di dalam* suatu proses. Karena teknik-teknik ini biasanya dibahas dalam bidang pemrograman, hal tersebut tidak akan dibahas secara keseluruhan pada fase analisis (beberapa akan dibahas pada kegiatan belajar berikutnya).

### C. MEMBUAT DATA FLOW DIAGRAM

Pembuatan Data Flow Diagram (DFD) dimulai dengan mempelajari informasi dalam dokumen *kasus penggunaan/use cases* (Dennis, 2012:149), atau merujuk pada *proses bisnis organisasi*, dan *definisi kebutuhan*. Dokumen sumber informasi tersebut (*use cases*, *proses bisnis organisasi*, dan *definisi kebutuhan*) dibuat atau dihasilkan oleh tim proyek bekerja sama dengan pengguna selama fase elisitasi kebutuhan (melalui wawancara atau sesi JAD). DFD biasanya dibuat oleh tim proyek dan kemudian ditinjau oleh pengguna. Tim proyek mempelajari dokumen sumber informasi tersebut dan menulis ulang sebagai DFD. Namun, karena DFD memiliki aturan formal tentang simbol dan sintaksis yang mungkin tidak terdapat pada dokumen sumber informasi tersebut, tim proyek terkadang harus merevisi beberapa informasi untuk membuatnya sesuai dengan aturan DFD. Jenis perubahan yang paling umum adalah *nama proses*, dan *input* dan *output* yang menjadi aliran data. Jenis perubahan paling umum lainnya adalah menggabungkan beberapa *input* dan *output* kecil dalam menjadi aliran data yang lebih besar di DFD (misalnya, menggabungkan tiga *input* terpisah, seperti *nama pelanggan*, *alamat pelanggan*, dan *nomor telepon pelanggan* ke dalam satu aliran data dengan nama *informasi pelanggan*).

Tim proyek biasanya menggunakan alat pemodelan proses atau *case tools* untuk menggambar model proses. Alat sederhana seperti Visio berisi set simbol DFD dan memungkinkan pembuatan dan modifikasi diagram dengan mudah. Alat pemodelan proses lainnya seperti BPWin dapat melakukan pemeriksaan sintaksis sederhana untuk memastikan bahwa penggambaran DFD setidaknya sudah benar. *Case tools* lengkap, seperti Visible Analyst Workbench, memberikan banyak kemampuan selain pemodelan proses (misalnya, pemodelan data). *Case tools* cenderung kompleks, dan meskipun memberi nilai tambah untuk proyek besar dan kompleks, seringkali harganya lebih mahal untuk dianggarkan dalam proyek sederhana.

Membangun model proses yang memiliki banyak level DFD biasanya memerlukan beberapa langkah. Beberapa analis lebih suka memulai pemodelan proses dengan memfokuskan terlebih dahulu pada diagram level 0. Analis lain memulai dengan membangun diagram konteks yang menunjukkan semua entitas luar dan aliran data yang berasal dari atau berakhir padanya.

Menurut Jogiyanto (2008), tahapan-tahapan dalam membuat DFD (PDFD atau LDFD) adalah:

1. mengidentifikasi semua entitas luar yang terlibat di sistem, dengan mengacu pada proses bisnis yang terdapat pada organisasi, atau mengacu pada hasil kajian kebutuhan bersama pengguna sistem;
2. mengidentifikasi semua *input* dan *output* yang terlibat dengan entitas luar;
3. menggambar Diagram Konteks;
4. membuat Bagan Berjenjang (*hierarchy chart*) untuk semua proses yang ada di sistem;
5. menggambar sketsa DFD untuk *overview diagram* (level 0) berdasarkan proses yang ada di Bagan Berjenjang;
6. menggambar DFD level berikutnya, yaitu level 1 dan seterusnya (sesuai Bagan Berjenjang);
7. menggabungkan semua proses yang digambar secara terpisah pada level 1 atau level berikutnya dalam satu gambar yang utuh.

Dennis (2012) mengemukakan tahapan-tahapan pembuatan DFD sebagai berikut:

1. membangun Diagram Konteks yang menunjukkan semua entitas luar dan aliran data yang berasal dari atau berakhir padanya;
2. membuat fragmen (penggalan) DFD untuk setiap kasus penggunaan (*use cases*) yang telah didesain;
3. menyusun fragmen DFD menjadi DFD level 0;
4. mengembangkan DFD level 1, berdasarkan langkah-langkah dalam setiap *use cases*, untuk lebih memperjelas bagaimana mereka beroperasi. Dalam beberapa kasus, DFD level 1 ini selanjutnya didekomposisi menjadi DFD level 2, DFD level 3, DFD level 4, dan seterusnya;
5. memvalidasi set DFD untuk memastikan sudah lengkap dan benar.

*Contoh Kasus:*

### **Pengembangan Sistem Informasi Perpustakaan**

Misalkan terdapat suatu prosedur/proses bisnis *manajemen pengelolaan perpustakaan* pada sebuah organisasi, atau prosedur bisnis yang ditetapkan berdasarkan hasil kajian bersama dengan pengguna sistem, sebagai berikut (proses bisnis ini hanya sebagai ilustrasi umum, dan situasinya dapat berbeda dengan situasi sebenarnya di suatu tempat tertentu).

1. Sebelum layanan peminjaman dijalankan, *petugas pengadaan koleksi* mempersiapkan koleksi pustaka, memberikan identitas pada setiap koleksi, menyusun koleksi pada rak yang tersedia sesuai kodennya masing-masing, dan *mencatat identitas setiap koleksi* ke dalam buku induk koleksi.
2. Jika terdapat *peminjam (anggota)* yang ingin meminjam sebuah koleksi perpustakaan:
  - a. *petugas layanan pustaka* terlebih dahulu mendata identitas anggota sebagai anggota perpustakaan ke dalam buku registrasi anggota, selanjutnya *menerbitkan kartu anggota*;
  - b. setelah terdaftar sebagai *anggota*, *anggota* dapat *mencari koleksi pustaka* pada buku induk koleksi, atau dapat mencari langsung pada rak koleksi;
  - c. *petugas layanan koleksi mendata/mencatat peminjaman* ke dalam buku induk peminjaman.
3. Jika terdapat anggota perpustakaan yang ingin mengembalikan koleksi, petugas layanan pustaka memeriksa data peminjaman pada buku induk peminjaman, dan mendata/mencatat pengembalian.
4. Pada setiap akhir bulan, petugas layanan koleksi mempersiapkan informasi dan laporan aktivitas perpusatakaan kepada *Kepala Unit Perpustakaan*. Informasi dan laporan berupa:
  - a. *daftar koleksi pustaka*,
  - b. *laporan peminjaman koleksi* setiap bulan.

Berdasarkan prosedur bisnis yang ada, akan dibuat DFD mengikuti tahapan pembuatan DFD yang dikemukakan oleh Jogiyanto (2008). Walaupun contoh yang diberikan di sini adalah untuk LDFD, namun pedoman atau prosedur ini juga berlaku untuk PDFD.

#### **1. Mengidentifikasi semua entitas luar yang terlibat di sistem.**

Berdasarkan prosedur bisnis *manajemen pengelolaan perpustakaan* terdapat entitas luar (perhatikan kata yang tercetak miring) berupa berikut ini.

- a. Petugas pengadaan koleksi pustaka (Petugas Pengadaan)
- b. Peminjam/anggota perpustakaan (Anggota)
- c. Petugas layanan koleksi pustaka (Petugas Layanan)
- d. Kepala Unit Perpustakaan (Kepala Perpustakaan)

#### 4.24 Pemodelan Analisis dengan Pendekatan Terstruktur

Entitas luar ini merupakan entitas di luar sistem (sistem informasi). Entitas luar ini merupakan sumber arus data ke sistem informasi serta menjadi tujuan (penerima) arus data hasil dari proses sistem informasi, sehingga merupakan entitas di luar sistem informasi.

#### 2. Mengidentifikasi semua *input* dan *output* yang terlibat dengan entitas luar

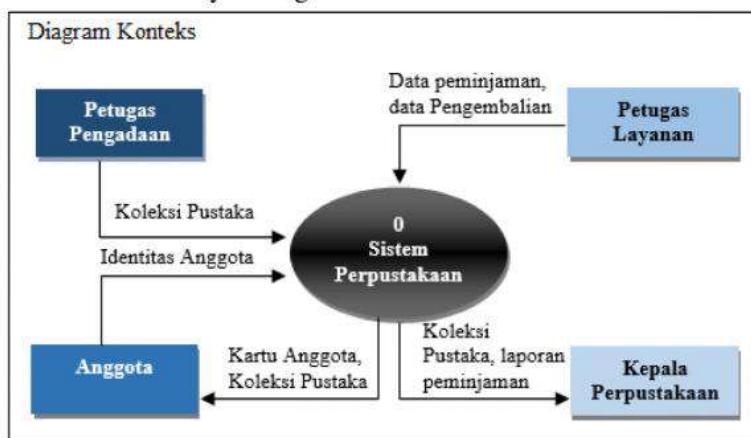
Untuk sistem perpustakaan ini, *input* dan atau *output* sistem yang terlibat dengan entitas luar (perhatikan kata yang tercetak miring pada proses bisnis) disajikan pada Gambar 4.21.

Entitas Luar	Input Sistem	Output Sistem
<b>Petugas Pengadaan Anggota</b>	Koleksi Pustaka Identitas anggota	Kartu anggota, Koleksi pustaka
<b>Petugas Layanan</b>	Data peminjaman, Data pengembalian	
<b>Kepala Perpustakaan</b>		Koleksi pustaka, Laporan peminjaman koleksi

Gambar 4.21  
*Input/Output* yang Terlibat pada Entitas Luar

#### 3. Menggambarkan Diagram Konteks

Diagram konteks dibuat dengan merujuk dan memperhatikan Gambar 4.21. Seluruh entitas luar dan *input/output* yang teridentifikasi pada Gambar 4.21 menjadi komponen DFD. Hasilnya sebagai berikut:



Gambar 4.22  
Diagram Konteks Sistem Perpustakaan

#### 4. Membuat Bagan Berjenjang

Bagan berjenjang (*hierarchy chart*) digunakan untuk mempersiapkan penggambaran DFD ke level-level lebih bawah. Bagan berjenjang dapat digambar

dengan menggunakan notasi proses yang digunakan pada DFD. Untuk Sistem Perpustakaan ini, semua proses yang teridentifikasi pada proses bisnis menjadi komponen proses pada Bagan Berjenjang. Berikut adalah *proses-proses utama* yang teridentifikasi pada proses bisnis pengelolaan perpustakaan.

- Pendataan awal (sebelum proses inti "peminjaman pustaka" dilaksanakan).
- Proses peminjaman koleksi pustaka.
- Proses pelaporan aktivitas perpustakaan.

Jika diurai lebih detail, maka ketiga inti proses tersebut akan menghasilkan proses-proses yang lebih rinci, sebagai berikut.

- Persiapan awal, terdiri atas kegiatan:
  - registrasi koleksi pustaka,
  - registrasi anggota.
- Proses peminjaman koleksi pustaka, terdiri atas kegiatan:
  - peminjaman koleksi pustaka,
  - pengembalian koleksi pustaka.
- Proses penyajian informasi dan pelaporan aktivitas pustaka, terdiri atas kegiatan:
  - penyajian informasi koleksi pustaka,
  - pelaporan aktivitas peminjaman koleksi pustaka.

Berdasarkan proses-proses yang lebih rinci tersebut, dibangun Bagan Berjenjang seperti pada Gambar 4.23.



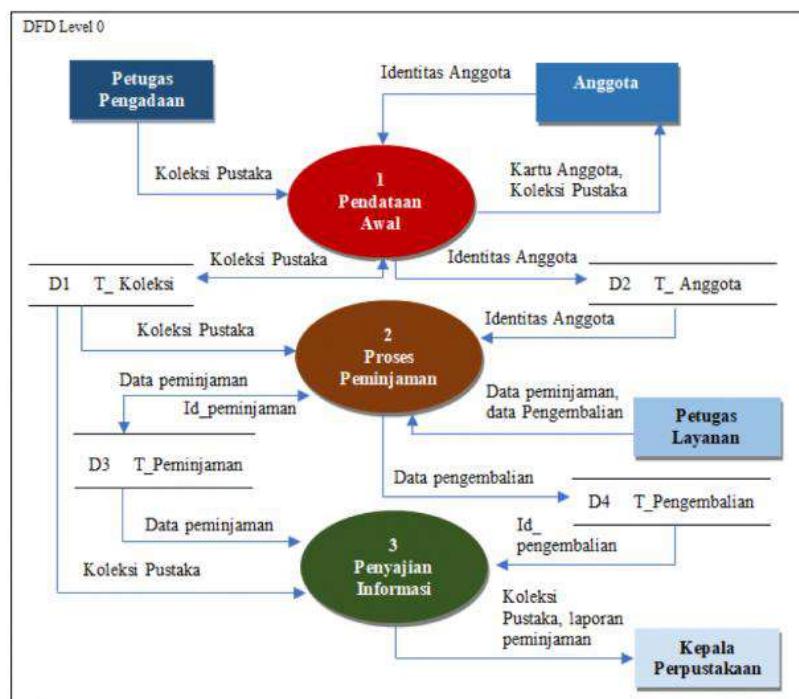
Gambar 4.23  
Diagram Berjenjang

Pada Diagram Berjenjang Gambar 4.23, jenjang pertama mewakili Diagram Konteks, jenjang ke-2 mewakili DFD level 0 yang merupakan proses-proses utama yang

teridentifikasi pada proses bisnis, dan jenjang ke-3 mewakili DFD level 1 yang merupakan proses-proses detail yang teridentifikasi pada proses bisnis.

### 5. Membuat DFD untuk *Overview Diagram (Level 0)*

DFD level 0 merujuk pada diagram berjenjang (jenjang ke-2), terdiri atas 3 proses utama yaitu: proses 1 (pendataan awal), proses 2 (proses peminjaman), dan proses 3 (penyajian informasi).



Gambar 4.24  
DFD Level 0 Sistem Perpustakaan

Sebagaimana telah dipaparkan sebelumnya bahwa prinsip kunci dalam membuat DFD adalah keseimbangan, yaitu memastikan bahwa semua informasi yang disajikan dalam DFD di satu tingkat secara akurat disajikan pada DFD tingkat berikutnya. Pada Gambar 4.22 (Diagram Konteks) terlihat bahwa 4 buah entitas luar (petugas pengadaan, anggota, petugas layanan, kepala perpustakaan) identik dengan entitas luar yang ada pada diagram level 0 (Gambar 4.24), dan bahwa data mengalir ke dan dari entitas luar dalam diagram konteks juga muncul di DFD level 0. DFD level 0 menggantikan proses tunggal diagram konteks dengan tiga proses (pendataan awal, peminjaman, dan penyajian informasi), menambahkan penyimpanan data (D1, D2, D3 dan D4). Pada diagram level 0 tidak terdapat aliran data dari satu proses ke proses lainnya. Hal ini disebabkan karena ilustrasi model DFD yang dibuat adalah DFD logika (LDFD). Pada sistem LDFD (sistem terkomputerisasi), interkoneksi data antara satu proses dengan

proses yang lain melalui perantara media penyimpanan data/tabel/database. Interkoneksi secara langsung antara sebuah proses dengan proses yang lainnya biasanya terjadi pada proses-proses yang berjalan secara manual.

Proses yang terjadi pada DFD level 0 (Gambar 4.24) dapat dijelaskan sebagai berikut.

a. *Proses 1 (Pendataan Awal)*

Proses *pendataan awal* terdiri atas 2 kegiatan, yaitu *pendataan koleksi pustaka* dan *registrasi anggota*.

- 1) Pada proses pendataan koleksi pustaka, sistem menerima *input* berupa *koleksi pustaka* dari entitas luar *petugas pengadaan*, memprosesnya dan menyimpan data (menghasilkan *output*) ke simpanan data D1 (tabel koleksi). Aliran data dari proses *pendataan awal* ke simpanan data D1 nampak 2 arah, menunjukkan bahwa proses 1 (*pendataan awal*) mengambil kembali (*query*) data dari simpanan data D1 untuk keperluan penyajian informasi *koleksi pustaka* ke entitas luar *anggota*.
- 2) Pada proses registrasi anggota, sistem menerima *input* berupa *identitas anggota* perpustakaan dari entitas luar *anggota*, memprosesnya dan menyimpan data (menghasilkan *output*) ke D2 (tabel anggota). Pada saat yang sama, sistem juga memproses *identitas anggota* dan menghasilkan *output* berupa *kartu anggota* ke entitas luar *anggota*.

b. *Proses 2 (Peminjaman Koleksi)*

Proses *peminjaman* terdiri atas 2 kegiatan utama, yaitu proses peminjaman dan proses pengembalian pustaka.

- 1) Pada proses peminjaman, sistem menerima *input* berupa *koleksi pustaka* dari simpanan data D1, *identitas anggota* dari simpanan data D2, serta *data peminjaman* (misalnya: *tanggal peminjaman*, *tanggal harus dikembalikan*) dari entitas luar *petugas layanan*. Ketiga *input* tersebut merupakan komponen *input* untuk proses peminjaman. Proses peminjaman menghasilkan *output* berupa *data peminjaman* mengalir ke simpanan data D3.
- 2) Pada proses pengembalian, sistem menerima *input* berupa sebagian *data peminjaman* (*id\_peminjaman*) dari simpanan data D3 dengan aliran data 2 arah (pustaka yang dikembalikan adalah pustaka yang pernah dipinjam) dan *data pengembalian* (misalnya: *tanggal pengembalian*) dari entitas luar *petugas layanan*. Proses pengembalian menghasilkan *output* berupa *data pengembalian* mengalir ke simpanan data D4.

c. *Proses 3 (Penyajian Informasi)*

Proses *penyajian informasi* terdiri atas 2 kegiatan utama, yaitu proses *penyajian informasi koleksi pustaka* dan *pelaporan aktivitas peminjaman* koleksi pustaka.

- 1) Pada proses *penyajian informasi koleksi pustaka*, sistem menerima *input* berupa data *koleksi pustaka* dari simpanan data D1, memprosesnya dan menghasilkan *output* berupa *informasi koleksi pustaka* ke entitas luar *kepala perpustakaan*.
- 2) Pada proses *pelaporan aktivitas peminjaman* koleksi pustaka, sistem menerima *input* berupa *data peminjaman* dari simpanan data D3 dan data pengembalian (*id\_pengembalian*) dari simpanan data D4, memprosesnya dan menghasilkan *output* berupa *laporan peminjaman* koleksi pustaka.

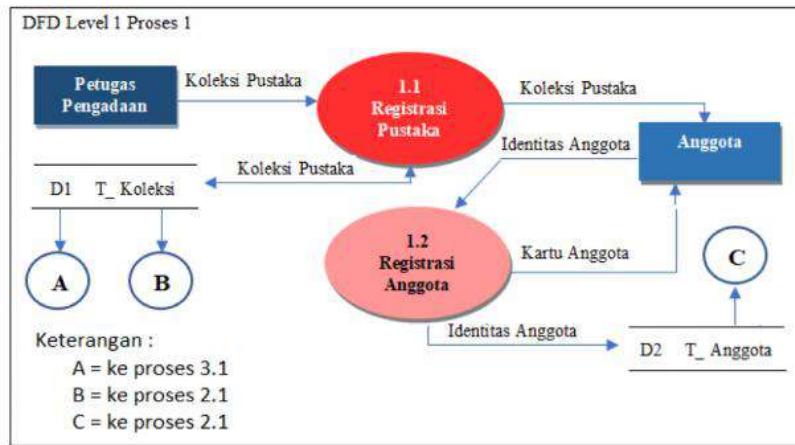
**6. Membuat DFD Level 1**

Berdasarkan paparan ketiga proses utama pada DFD level 0 (pendataan awal, peminjaman koleksi, dan penyajian informasi) diketahui bahwa terdapat proses-proses *tersembunyi* yang belum ditampilkan pada DFD Level 0 (masih menyatu di dalam proses utama), misalnya: *proses 1* (pendataan awal) yang pada DFD level 0 tampak hanya 1 proses, padahal di balik proses tersebut sebenarnya terdapat 2 proses (proses *pendataan koleksi* dan proses *registrasi anggota*). Proses-proses yang masih "tersembunyi" tersebut secara detail akan digambarkan pada DFD level 1.

DFD level 1 merujuk pada diagram berjenjang (jenjang ke-3), terdiri atas 6 detail proses, yaitu: proses 1.1 (registrasi pustaka) dan proses 1.2 (registrasi anggota) yang merupakan bagian (turunan) dari proses 1 (pendataan awal); proses 2.1 (peminjaman pustaka) dan proses 2.2 (pengembalian pustaka) yang merupakan bagian (turunan) dari proses 2 (proses peminjaman); serta proses 3.1 (penyajian koleksi) dan proses 3.2 (pelaporan peminjaman) yang merupakan bagian (turunan) dari proses 3 (penyajian informasi). Dengan demikian, DFD level 1 terdiri atas 3 serpihan gambar yaitu: DFD level 1 proses 1 (proses registrasi pustaka dan proses registrasi anggota), DFD level 1 proses 2 (proses peminjaman dan proses pengembalian), serta DFD level 1 proses 3 (proses penyajian koleksi dan proses pelaporan peminjaman).

a. *DFD Level 1 Proses 1*

DFD level 1 proses 1 terdiri atas 2 proses, yaitu proses 1.1 (registrasi pustaka) dan proses 1.2 (registrasi anggota).



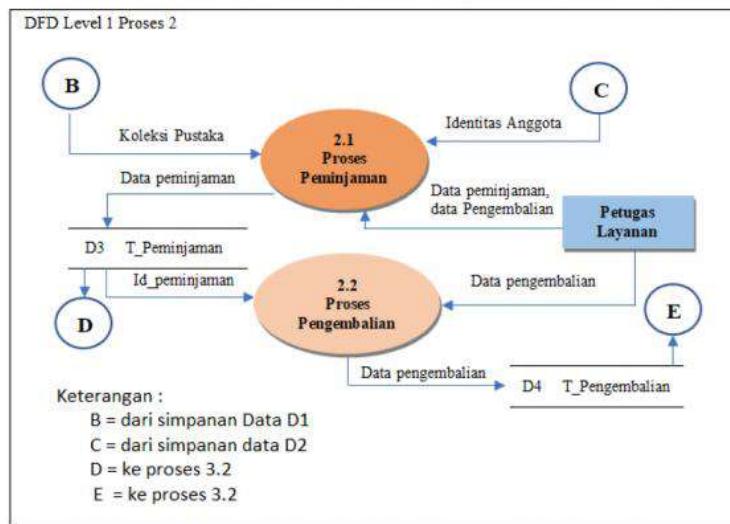
Gambar 4.25  
DFD Level 1 Proses 1 Sistem Perpustakaan

Pada Gambar 4.25 terlihat bahwa 2 buah entitas luar (petugas pengadaan, anggota) identik dengan entitas luar yang ada di diagram level 0 proses 1 (Gambar 4.24), dan bahwa data mengalir ke dan dari entitas luar dalam DFD level 0 proses 1 juga muncul di DFD level 1 proses 1. Hal tersebut menunjukkan semua informasi yang disajikan dalam DFD level 0 proses 1 secara akurat telah disajikan dalam DFD level 1 proses 1. Hanya saja aliran data *output* maupun aliran data *input* kedua entitas luar tersebut tidak lagi identik, sebab proses tunggal DFD level 0 proses 1 (pendataan awal) telah digantikan oleh dua proses pada DFD level 1 proses 1 (registrasi pustaka dan registrasi anggota). *Output* entitas luar *petugas pengadaan* menuju ke proses 1.1 (registrasi pustaka), sedangkan *output* entitas luar *anggota* menuju ke proses 1.2 (registrasi anggota). Demikian juga dengan *input* entitas luar *anggota* telah mengalami perubahan: input berupa *koleksi pustaka* berasal dari proses 1.1 (registrasi pustaka) dan input berupa *kartu anggota* berasal dari proses 1.2 (registrasi anggota).

Simpanan data juga mengalami perubahan *input* sebagai akibat pemecahan DFD level 0 proses 1 menjadi 2 proses baru (proses 1.1 dan proses 1.2). *Input* simpanan data D1 berasal dari proses 1.1 (registrasi pustaka), sedangkan *input* simpanan data D2 berasal dari proses 1.2 (registrasi anggota). Adapun *output* simpanan data D1 (A dan B) menjadi penghubung ke DFD level 1 proses 3.1 (penyajian koleksi) dan DFD level 1 proses 2.1 (proses peminjaman), dan simpanan data D2 (C) menjadi penghubung ke DFD level 1 proses 2.1 (proses peminjaman), yang akan digambarkan secara terpisah selanjutnya.

#### b. DFD Level 1 Proses 2

DFD level 1 proses 2 terdiri atas 2 proses, yaitu proses 2.1 (proses peminjaman) dan proses 2.2 (proses pengembalian).



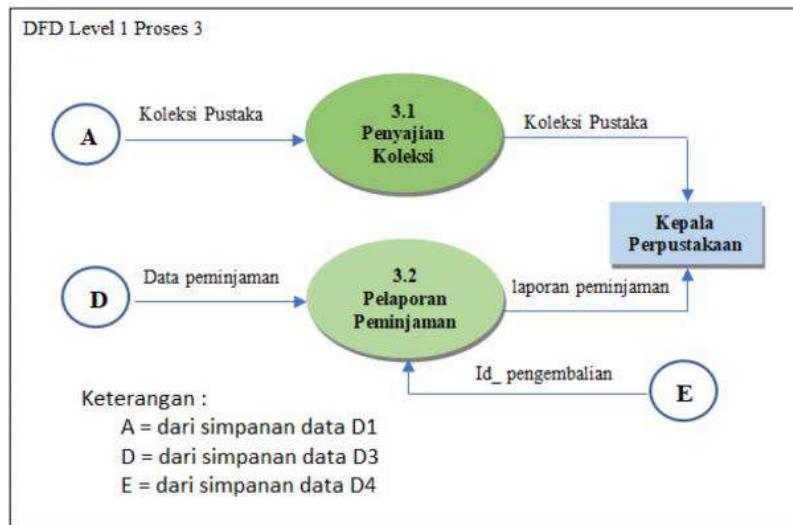
Gambar 4.26  
DFD Level 1 Proses 2 Sistem Perpustakaan

Pada Gambar 4.26, entitas luar (petugas layanan) identik dengan entitas luar yang ada di diagram level 0 proses 2 (Gambar 4.24), dan bahwa data mengalir ke dan dari entitas luar dalam DFD level 0 proses 2 juga muncul di DFD level 1 proses 2. Hanya saja aliran data *output* entitas luar tersebut tidak lagi identik, sebab proses tunggal DFD level 0 proses 2 (proses peminjaman) telah digantikan oleh dua proses pada DFD level 1 proses 2 (proses peminjaman dan proses pengembalian). *Output* entitas luar *petugas layanan* menuju ke proses 2.1 (proses peminjaman) dan menuju ke proses 2.2 (proses pengembalian).

Simpanan data juga mengalami perubahan *input* sebagai akibat pemecahan DFD level 0 proses 2 menjadi 2 proses baru (proses 2.1 dan proses 2.2). *Input* simpanan data D3 berasal dari proses 2.1 (proses peminjaman), sedangkan *input* simpanan data D4 berasal dari proses 2.2 (proses pengembalian). Adapun *output* simpanan data D3 (D) dan simpanan data D4 (E) menjadi penghubung ke DFD level 1 proses 3.2 (pelaporan peminjaman) yang akan digambarkan secara terpisah selanjutnya.

c. *DFD Level 1 Proses 3*

DFD level 1 proses 3 terdiri atas 2 proses, yaitu proses 3.1 (penyajian koleksi) dan proses 3.2 (pelaporan peminjaman).



Gambar 4.27  
DFD Level 1 Proses 3 Sistem Perpustakaan

Pada Gambar 4.27, entitas luar (kepala perpustakaan) identik dengan entitas luar yang ada di diagram level 0 proses 3 (Gambar 4.24), dan bahwa data yang mengalir ke entitas luar dalam DFD level 0 proses 3 juga muncul di DFD level 1 proses 3. Hanya saja aliran data *input* ke entitas luar tersebut tidak lagi identik, sebab proses tunggal DFD level 0 proses 3 (penyajian informasi) telah digantikan oleh dua proses pada DFD level 1 proses 3 (penyajian koleksi dan pelaporan peminjaman). *Input* entitas luar *kepala perpustakaan* telah mengalami perubahan: input berupa *koleksi pustaka* berasal dari proses 3.1 (penyajian koleksi) dan input berupa *laporan peminjaman* berasal dari proses 3.2 (pelaporan peminjaman).

#### 7. Menggabungkan proses yang digambar secara terpisah menjadi satu gambar yang utuh

Setelah semua proses pada DFD level 1 (atau level-level selanjutnya) digambar, maka keseluruhan proses DFD level 1 tersebut dapat digabungkan dalam satu diagram yang menyatu (jika medium penggambaran memungkinkan itu dilakukan). Penggabungan proses hanya dapat dilakukan pada proses-proses yang terdapat pada level yang sama, dan tidak dibenarkan menggabungkan proses-proses yang terdapat pada level yang berbeda. Misalnya, suatu proses (atau seluruh proses) yang terdapat pada DFD level 1 tidak dapat digabungkan dengan suatu proses (atau seluruh proses) yang terdapat pada DFD level 2 atau level lainnya.

#### D. PERBEDAAN DFD DENGAN *FLOWCHART* DAN KETERBATASAN DFD

DFD sangat berbeda dengan bagan alir (*flowchart*). Beberapa perbedaan mendasar adalah sebagai berikut.

1. Proses di DFD dapat beroperasi secara paralel, sehingga beberapa proses dapat dilakukan serentak. Hal ini merupakan kelebihan DFD dibandingkan dengan bagan alir yang cenderung hanya menunjukkan proses yang urut. Pada kenyataannya kegiatan-kegiatan proses dapat dilakukan secara tidak urut, yaitu secara paralel atau serentak, sehingga DFD dapat menggambarkan proses semacam ini dengan lebih mengena.
2. DFD lebih menunjukkan arus data di suatu sistem, sedang bagan alir sistem lebih menunjukkan arus dari prosedur dan bagan alir program lebih menunjukkan arus dari algoritma.
3. DFD tidak menunjukkan proses perulangan (*loop*) dan proses keputusan (*decision*), sedang bagan alir menunjukkannya.

Walaupun DFD mempunyai kelebihan-kelebihan, seperti dapat menggambarkan sistem secara terstruktur dengan memecah-mecah menjadi level lebih rendah (*decomposition*), dapat menunjukkan arus data di sistem, dapat menggambarkan proses paralel di sistem, dapat menunjukkan simpanan data, dapat menunjukkan kesatuan luar, DFD juga mempunyai beberapa keterbatasan, di antaranya:

1. DFD tidak dapat menunjukkan proses logika seperti proses keputusan (*decision*) dan proses perulangan (*loop*),
2. DFD tidak menunjukkan proses perhitungan.

Dengan keterbatasan tersebut, diperlukan alat bantu lain yang menyertainya untuk mendeskripsikan logika yang terjadi di dalam suatu proses secara lebih detail, untuk mendukung kegiatan dalam fase desain.



#### Latihan

Untuk memperdalam pemahaman Anda mengenai materi di atas, kerjakanlah latihan berikut!

- 1) Dalam analisis dan desain sistem informasi dikenal istilah *Pendekatan Terstruktur*. Jelaskan arti istilah tersebut! Apa kelebihan dan kekurangan sebagai sebuah pendekatan dalam analisis dan desain/perancangan sistem informasi?
- 2) Jelaskan perbedaan antara model proses logis dan model proses fisik! Berikan contoh masing-masing!

- 3) Jelaskan kegunaan DFD dalam fase analisis sistem informasi!
- 4) Jelaskan beberapa kesalahan umum dalam penggambaran DFD!
- 5) Perhatikan penggalan struktur/*hierarchy* proses akademik (perkuliahannya) berikut ini.

## SISTEM AKADEMIK

### 1. Persiapan Awal

#### 1.1 Registrasi

- 1.1.1 Registrasi Mata Kuliah
- 1.1.2 Registrasi Mahasiswa
- 1.1.3 Registrasi Dosen

#### 1.2 Persiapan Perangkat Perkuliahan

- 1.2.1 Membuat Jadwal Kuliah
- 1.2.2 Memprogramkan Mata Kuliah
- 1.2.3 Membuat Daftar Hadir Perkuliahan

### 2. Perkuliahan

#### 2.1 Proses x

- 2.1.1 Proses xx
- 2.2.2 Proses xxx

### 3. Publikasi Nilai

#### 3.1 Proses y

- 3.1.1 Proses yy
- 3.1.2 Prosesyyy

Pada proses Registrasi (proses 1.1):

- Staf administrasi Akademik (Staf ADM) mendaftarkan Mata Kuliah yang akan ditawarkan pada semester berjalan ke dalam sistem. Staf ADM juga mendaftarkan Dosen yang akan mengajar pada semester berjalan ke dalam sistem
- Mahasiswa yang akan mengikuti semester berjalan melakukan daftar ulang ke dalam sistem

Pada proses Persiapan Perangkat Perkuliahan (1.2):

- Staf ADM membuat Jadwal Perkuliahan berdasarkan data Mata Kuliah dan Dosen yang telah terdaftar dalam sistem
- Mahasiswa memprogramkan Mata Kuliah pada semester berjalan berdasarkan Jadwal Kuliah yang telah dibuat oleh Staf ADM
- Staf ADM membuat Daftar Hadir Perkuliahan berdasarkan Jadwal Kuliah

Pertanyaan:

Berdasarkan penggalan struktur/*hierarchy* proses dan proses bisnis tersebut di atas: Identifikasi proses apa saja yang terdapat pada *Level 2 Proses 1.1*, selanjutnya gambarkan DFDnya (DFD level 2 proses 1.1)!

*Petunjuk Jawaban Latihan*

- 1) Pendekatan terstruktur merupakan pendekatan formal untuk memecahkan masalah-masalah dalam proses bisnis menjadi bagian-bagian kecil yang saling berhubungan untuk selanjutnya dapat disatukan kembali menjadi satu kesatuan yang dapat dipergunakan untuk memecahkan masalah. Dekomposisi permasalahan dilakukan berdasarkan fungsi atau proses secara hirarki, mulai dari konteks sampai proses-proses yang paling kecil.

Kelebihan pendekatan terstruktur sebagai berikut.

- a) Bersifat visual dengan menggunakan analisis grafis, sehingga mudah dimengerti oleh pengguna sistem atau programmer.
- b) Merupakan pendekatan yang sudah lama diterapkan pada berbagai bidang industri sehingga pendekatan ini sudah matang dan layak untuk digunakan.

Kelemahan pendekatan terstruktur sebagai berikut.

- a) Berorientasi pada proses, sehingga tidak dapat digunakan dalam menganalisis kebutuhan nonfungsional.
- b) Tidak dapat memenuhi kebutuhan pemrograman berorientasi objek, karena model ini memang didesain khusus untuk mendukung konsep pemrograman terstruktur

- 2) Model proses logis adalah model yang menggambarkan proses yang berfokus pada bagaimana bisnis harus dijalankan, dengan tidak memperhitungkan bagaimana model tersebut akan diimplementasikan secara teknis (misalnya: terkomputerisasi atau manual, berbasis aplikasi desktop atau web, dan lain-lain).

Contoh: Model pengelolaan administrasi akademik (perkuliahan) berbasis komputer (tanpa menunjukkan penggunaan aplikasi komputer tertentu).

Model proses fisik adalah model yang menunjukkan bagaimana detail proses suatu sistem diimplementasikan (misalnya: referensi teknologi aktual/tertentu yang digunakan, format informasi yang bergerak melalui proses, dan interaksi manusia yang terlibat).

Contoh: Model aplikasi sistem pengelolaan administrasi akademik (perkuliahan) berbasis bahasa pemrograman PHP dan Basis Data MySQL.

- 3) Data Flow Diagram (DFD) digunakan dalam fase analisis sistem untuk menggambarkan aliran proses secara logika tentang suatu sistem yang sedang berjalan atau sistem baru yang akan dikembangkan, tanpa mempertimbangkan lingkungan fisik di mana aliran proses tersebut berjalan, apakah suatu proses terkomputerisasi atau manual, apakah informasi dikumpulkan melalui formulir kertas atau sistem elektronik, atau lingkungan fisik di mana data tersebut akan disimpan.
- 4) Kesalahan-kesalahan umum yang biasa terjadi dalam penggambaran DFD.
  - a) Sebuah proses mempunyai arus data input, namun tidak menghasilkan output (data terjebak dalam sistem).
  - b) Sebuah proses mempunyai arus data input, namun tidak menghasilkan suatu output (suatu kejadian yang mustahil).
  - c) Sebuah simpanan data mempunyai arus data input, namun data tersebut tidak pernah digunakan (data terjebak dalam sistem).
  - d) Sebuah simpanan data mempunyai arus data output, namun tidak pernah mendapatkan input (suatu kejadian yang mustahil).
  - e) Simpanan data berhubungan langsung dengan simpanan data lainnya atau dengan sebuah entitas luar. Hanya proses saja yang bisa berhubungan langsung dengan simpanan data.
  - f) Entitas luar berhubungan langsung dengan entitas luar lainnya. Entitas luar seharusnya dapat berhubungan dengan entitas luar lainnya melalui sebuah proses.
- 5) Proses yang terdapat dalam Level 2 Proses 1.1 adalah: proses 1.1.1 (Registrasi mata kuliah), proses 1.1.2 (Registrasi mahasiswa), dan proses 1.1.3 (Registrasi dosen).

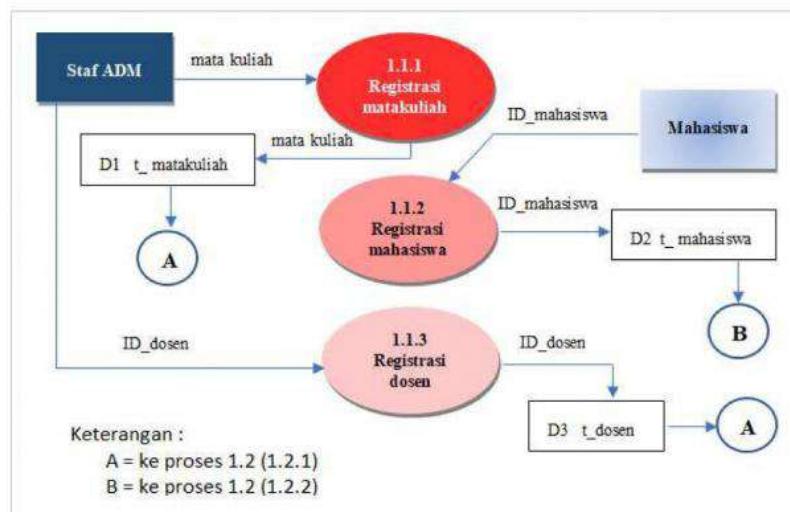
Pada contoh struktur/*hierarchy* SISTEM AKADEMIK di atas, terdapat 4 kelompok proses yang harus digambar pada level 2, yaitu: kelompok proses pada level 2 proses 1.1; kelompok proses pada level 2 proses 1.1.1; kelompok proses pada level 2 proses 1.1.2; dan kelompok proses pada level 2 proses 1.1.3.

Berikut adalah gambar untuk kelompok proses pada level 2 proses 1.1.

- a) Melibatkan entitas luar: Staf ADM dan Mahasiswa
- b) Melibatkan 3 buah proses, yaitu: proses 1.1.1 (Registrasi mata kuliah), proses 1.1.2 (Registrasi mahasiswa), dan proses 1.1.3 (Registrasi dosen).
- c) Melibatkan 3 buah simpanan data, yaitu: D1 (matakuliah), D2 (mahasiswa), dan D3 (dosen).

## d) Melibatkan arus data:

- (1) Arus data mata kuliah mengalir dari entitas luar Staf ADM ke proses 1.1.1
- (2) Arus data ID\_mahasiswa mengalir dari entitas luar Mahasiswa ke proses 1.1.2
- (3) Arus data ID\_Dosen mengalir dari entitas luar Staf ADM ke proses 1.1.3
- (4) Arus data mata kuliah mengalir dari proses 1.1.1 ke simpanan data D1(t\_matakuliah)
- (5) Arus data ID\_mahasiswa mengalir dari proses 1.1.2 ke simpanan data D2(t\_mahasiswa)
- (6) Arus data ID\_Dosen mengalir dari proses 1.1.3 ke simpanan data D3(t\_dosen)
- (7) Arus data matakuliah dari simpanan data D1 (t\_matakuliah) akan digunakan oleh proses 1.2 (1.2.1/membuat jadwal kuliah)
- (8) Arus data ID\_mahasiswa dari simpanan data D2 (t\_mahasiswa) akan digunakan oleh proses 1.2 (1.2.2/memprogramkan Mata Kuliah)
- (9) Arus data ID\_Dosen dari simpanan data D3 (t\_dosen) akan digunakan oleh proses 1.2 (1.2.1/membuat jadwal kuliah)

**Rangkuman**

Terdapat empat simbol yang digunakan pada DFD yaitu: proses, arus data, penyimpanan data, dan entitas luar. Suatu proses adalah kegiatan yang melakukan sesuatu. Setiap proses memiliki nama (frasa kata kerja), deskripsi, dan angka yang menunjukkan keterkaitan dengan proses lain dan proses turunannya. Setiap proses harus memiliki setidaknya satu output dan biasanya memiliki setidaknya satu input. Aliran

data adalah potongan data atau objek, memiliki nama (kata benda) dan deskripsi, bermula atau berakhir pada suatu proses (atau keduanya). Penyimpanan data adalah file manual atau file komputer, memiliki nomor pengenal, nama (kata benda), dan setidaknya terdapat satu aliran data input dan satu aliran data output. Entitas luar adalah orang, organisasi, atau sistem di luar ruang lingkup sistem, memiliki nama (kata benda) dan deskripsi.

Setiap set DFD dimulai dengan diagram konteks, DFD level 0, dan memiliki beberapa DFD level 1, DFD level 2, dan seterusnya. Setiap elemen pada DFD level yang lebih tinggi (misalnya: aliran data, penyimpanan data, dan entitas eksternal) harus muncul pada DFD level yang lebih rendah. Jika tidak, maka antara level tidak akan seimbang.

DFD dibuat dari *use cases* atau merujuk pada proses bisnis organisasi. Pertama, tim membangun diagram konteks yang menunjukkan semua entitas luar dan data yang mengalir masuk dan keluar dari sistem. Kedua, tim membuat fragmen (penggalan) DFD untuk setiap kasus penggunaan yang menunjukkan bagaimana kasus penggunaan pertukaran data dengan entitas luar dan penyimpanan data. Dapat juga terlebih dahulu dibuat *hierarchy chart* untuk memetakan proses-proses secara bertingkat dan lebih rinci dalam sistem. Ketiga, fragmen DFD ini disusun menjadi DFD level 0. Keempat, mengembangkan DFD level 1 dan level-level seterusnya berdasarkan *hierarchy chart*. Kelima, menggabungkan semua proses yang digambar secara terpisah pada level 1 atau level-level di bawahnya dalam 1 gambar yang utuh. Keenam, memvalidasi set DFD untuk memastikan bahwa DFD sudah lengkap dan benar dan tidak mengandung kesalahan sintaks atau semantik.

Analis jarang membuat DFD secara sekaligus dengan hasil yang sempurna, jadi pembuatan dengan model iterasi per proses yang menghasilkan DFD pada *multipage* penting dilakukan untuk memastikan jelas dan mudah dibaca, sebelum disatukan secara lengkap dalam satu halaman.



### Tes Formatif 1

Pilihlah satu jawaban yang paling tepat!

- 1) Terdapat dua jenis Data Flow Diagram yaitu ....
  - A. *Logical* dan *Conceptual DFD*
  - B. *Logical* dan *Physical DFD*
  - C. *Physical* dan *Conceptual DFD*
  - D. *Physical* dan *Procedural DFD*
  
- 2) Dalam penggambaran DFD, berikut ini dapat dikategorikan sebagai arus data, *kecuali* ....
  - A. input untuk komputer dan tampilan atau output di layar komputer
  - B. formulir atau dokumen dasar, laporan tercetak, menyalin file

- C. ucapan langsung dari seseorang, surat-surat atau memo, isian pada buku agenda
  - D. data yang ditransmisikan dari satu komputer ke komputer yang lain
- 3) Berikut ini yang dapat dikategorikan sebagai simpanan data adalah ....
- A. suatu file atau database di sistem komputer
  - B. suatu agenda atau buku, folder atau map di meja seseorang
  - C. suatu tabel acuan manual
  - D. semua jawaban A, B, dan C benar
- 4) Alasan yang paling tepat mengapa DFD digambarkan secara bertingkat (beberapa level) adalah ....
- A. agar proses bisnis lebih mudah dijelaskan/dipahami
  - B. agar rapi dan lebih mudah dalam penggambaran
  - C. penggambaran DFD dengan cakupan sistem yang besar tidak dapat ditulis (dimuat) dalam 1 lembar kertas
  - D. karena aturan penggambaran mengharuskan demikian
- 5) Jumlah level yang ideal dalam penggambaran DFD adalah ....
- A. cukup pada level 1
  - B. hingga pada level 2
  - C. tergantung pada dekomposisi proses yang dilakukan pada *hierarchy chart*
  - D. tergantung pada tingkat kerumitan proses bisnis yang menjadi acuan penggambaran DFD
- 6) Penggambaran DFD dapat disertai dengan deskripsi proses untuk memberikan informasi tambahan mengenai logika proses yang lebih rinci, yang tidak diberikan oleh DFD. Berikut adalah beberapa teknik yang dapat digunakan sebagai alat deskripsi proses pada DFD, *kecuali* ....
- A. *Structured English (Pseudocode)*
  - B. *Decision Trees*
  - C. *Flowchart*
  - D. *Entity Relationship Diagram*
- 7) Misalkan sebuah proses DFD diberi nomor 5.4.3. Nomor proses tersebut termasuk dalam DFD level ....
- A. DFD level 1 proses 5
  - B. DFD level 2 proses 5
  - C. DFD level 2 proses 5.4
  - D. DFD level 3 proses 5.4

- 8) Walaupun DFD memiliki beberapa kelebihan, DFD juga memiliki kelemahan, yaitu ....
- tidak dapat digunakan untuk menggambarkan suatu proses yang paralel
  - tidak dapat menunjukkan proses logika seperti: keputusan dan perulangan
  - DFD tidak menunjukkan proses perhitungan
  - jawaban B dan C benar
- 9) Walaupun namanya Diagram Arus Data, namun bukan berarti bahwa DFD dapat digunakan untuk pemodelan Data dalam sistem basis data. Pernyataan ini adalah ....
- Benar
  - Salah
- 10) Sebuah simpanan data dan atau entitas luar tidak dapat dibuat duplikat dalam 1 set penggambaran DFD. Pernyataan ini adalah ....
- Benar
  - Salah

Cocokkanlah jawaban Anda dengan Kunci Jawaban Tes Formatif 1 yang terdapat di bagian akhir modul ini. Hitunglah jawaban yang benar. Kemudian, gunakan rumus berikut untuk mengetahui tingkat penguasaan Anda terhadap materi Kegiatan Belajar 1.

$$\text{Tingkat Penguasaan} = \frac{\text{Jumlah Jawaban yang Benar}}{\text{Jumlah Soal}} \times 100$$

Arti tingkat penguasaan

<70%

70% - 79%

80% - 89%

90% - 100%

kurang

cukup

baik

baik sekali

Apabila mencapai tingkat penguasaan 80% atau lebih, Anda dapat meneruskan dengan Kegiatan Belajar 2. **Bagus!** Jika masih di bawah 80%, Anda harus mengulangi materi Kegiatan Belajar 1, terutama bagian yang belum dikuasai.

## Bagan Alir

**S**eperti yang telah diuraikan pada kegiatan belajar sebelumnya bahwa dalam kebanyakan kasus, suatu proses cukup mudah dipahami dan dideskripsikan sehingga dengan adanya *definisi kebutuhan*, *kasus penggunaan/use case* (bukan *use case diagram*), dan DFD dengan *deskripsi teks sederhana* secara bersama-sama dapat menyajikan proses yang cukup untuk mendukung kegiatan dalam fase desain. Namun, kadang-kadang terdapat suatu proses yang cukup rumit sehingga harus dituangkan dalam deskripsi proses yang lebih rinci yang menjelaskan logika yang terjadi di dalam proses. Penggunaan DFD memiliki keterbatasan dalam menjelaskan logika pemrosesan, terutama logika pemrosesan yang cukup kompleks, seperti proses keputusan (*decision*) dan proses perulangan (*loop*). Dengan keterbatasan tersebut, diperlukan alat bantu lain yang menyertainya untuk mendeskripsikan logika yang terjadi di dalam suatu proses secara lebih detail, untuk mendukung kegiatan dalam fase desain. Dalam kegiatan belajar ini akan dibahas beberapa jenis diagram alir sebagai alat bantu untuk mendukung DFD menyampaikan pemahaman yang tepat tentang apa yang terjadi *di dalam* suatu proses, dengan menggambarkan logika yang terjadi di dalam suatu sistem atau dalam suatu proses secara lebih detail.

Bagan alir (*flowchart*) adalah bagan-bagan (*chart*) yang menunjukkan alir (*flow*) di dalam program atau prosedur sistem secara logika (Jogiyanto, 2008). Bagan alir digunakan terutama untuk alat bantu komunikasi dan untuk dokumentasi.

Ada beberapa macam bagan alir, seperti: bagan alir sistem (*systems flowchart*), bagan alir dokumen (*document flowchart*), bagan alir program (*program flowchart*), bagan alir skematik (*schematic flowchart*), bagan alir proses (*process flowchart*), dan sebagainya. *Flowchart* yang akan dibahas dalam kegiatan belajar ini adalah: bagan alir sistem (*systems flowchart*), bagan alir dokumen (*document flowchart*), dan bagan alir program (*program flowchart*).

Beberapa hal yang menjadi pedoman dasar dalam menggambar suatu bagan alir (*flowchart*) sebagai berikut.

1. Bagan alir sebaiknya digambar dari atas ke bawah dan mulai dari bagian kiri dari suatu halaman.
2. Kegiatan atau sebuah proses di dalam bagan alir harus ditunjukkan dengan jelas, dengan menggunakan suatu kata yang mewakili kegiatan atau proses tersebut, misanya: *persiapan*, *dokumen hitungan gaji*, dan sebagainya.

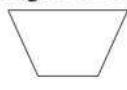
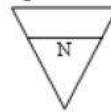
3. Penggambaran bagan alir harus ditunjukkan dari mana kegiatan akan dimulai dan di mana akan berakhir.
4. Masing-masing kegiatan di dalam bagan alir harus di dalam urutan yang semestinya.
5. Kegiatan yang terpotong dan akan disambung di tempat lain harus ditunjukkan dengan jelas menggunakan simbol penghubung.
6. Menggunakan simbol-simbol bagan alir yang standar.

#### A. BAGAN ALIR SISTEM (SYSTEM FLOWCHART)

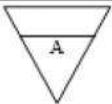
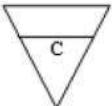
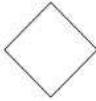
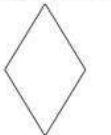
*System flowchart* merupakan bagan yang menunjukkan kerja atau apa yang sedang dikerjakan di dalam sistem atau sub sistem secara menyeluruh dan menjelaskan urutan dari prosedur-prosedur yang ada di dalamnya, dengan menggunakan simbol-simbol tertentu. Dengan kata lain, *system flowchart* ini merupakan deskripsi secara grafik dari urutan prosedur-prosedur yang terkombinasi yang membentuk suatu sistem atau sub sistem. Prosedur yang dimaksudkan dapat berupa prosedur proses pada sistem bisnis maupun prosedur proses operasi pada program aplikasi (bukan prosedur logika program aplikasi: prosedur logika program aplikasi digambarkan dengan *program flowchart*).

*System flowchart* terdiri dari data yang mengalir melalui sistem dan proses yang mentransformasikan data itu. Data dan proses dalam *system flowchart* dapat digambarkan secara *online* atau secara *offline*.

*System flowchart* digambar dengan menggunakan simbol-simbol seperti berikut:

Simbol	Keterangan
Dokumen 	Menunjukkan dokumen input atau output, baik untuk proses manual, mekanik, maupun komputer.
Awal / Akhir proses 	Menunjukkan awal proses atau akhir proses
Kegiatan Manual 	Menunjukkan pekerjaan manual
Simpanan offline 	File non-komputer yang disimpan berdasarkan urut angka ( <i>numerical</i> )

4.42 Pemodelan Analisis dengan Pendekatan Terstruktur

	File non-komputer yang disimpan berdasarkan urut huruf ( <i>alphabetical</i> )
	File non-komputer yang disimpan berdasarkan urut tanggal ( <i>cronological</i> )
Kartu Plong 	Input/output yang menggunakan kartu plong/berlubang ( <i>punched card</i> )
Proses 	Proses dari operasi program komputer
Operasi Luar 	Operasi yang dilakukan di luar proses operasi komputer
Keputusan 	Proses keputusan (logika percabangan)
Pengurutan offline 	Proses pengurutan data di luar proses komputer
Pita Magnetik 	Input/output menggunakan pita magnetik
Hardisk 	Input/output menggunakan hardisk
Disket 	Input/output menggunakan disket

Drum magnetik 	Input/output menggunakan drum magnetik
Pita kertas berlubang 	Input/output menggunakan pita kertas berlubang
Keyboard 	Input yang menggunakan online keyboard
Display 	Output yang ditampilkan di monitor
Hub. Komunikasi 	Proses transmisi data melalui saluran komunikasi
Garis Alir 	Arus dari proses
Penghubung 	Ke halaman yang sama
 	Ke halaman lain (yang berbeda)
Penjelasan 	Penjelasan dari suatu proses

Gambar 4.28  
Simbol-simbol System Flowchart

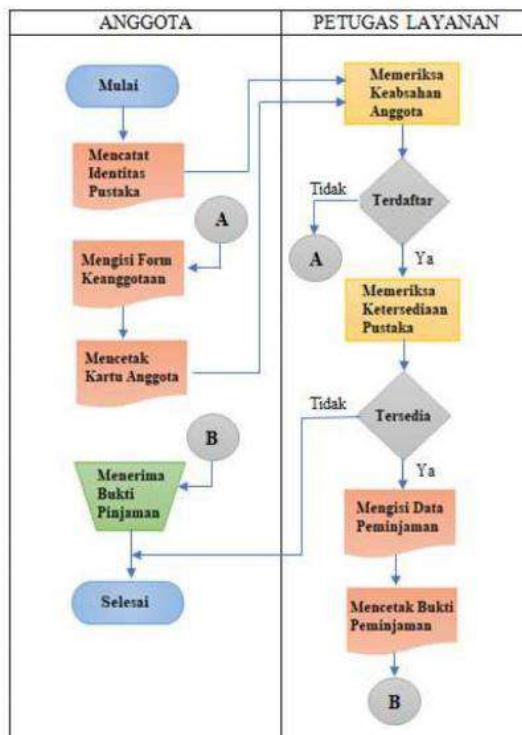
*Contoh*

### Proses Bisnis Peminjaman Pustaka

Misalkan terdapat prosedur bisnis "Peminjaman Pustaka" pada sistem Perpustakaan, sebagai berikut.

1. *Anggota* mempersiapkan kartu anggota dan mencatat identitas pustaka yang akan dipinjam pada form identitas pustaka yang akan dipinjam, selanjutnya menyerahkan formulir tersebut ke *petugas layanan* perpustakaan.
2. *Petugas layanan* memeriksa keabsahan anggota pada database anggota. Jika anggota tidak terdaftar, anggota dapat mengisi formulir keanggotaan untuk diproses menjadi anggota, serta menerima kartu anggota.
3. Jika anggota sudah terdaftar, petugas memeriksa ketersediaan pustaka yang diminta oleh anggota di dalam database koleksi. Jika pustaka yang akan dipinjam tidak tersedia, proses peminjaman berakhir.
4. Jika pustaka tersedia, petugas mengisi formulir peminjaman, mencetak bukti peminjaman dan menyerahkan koleksi pustaka serta slip peminjaman kepada anggota.

Prosedur bisnis untuk proses *peminjaman pustaka* di atas dapat digambarkan dalam *System Flowchart*, seperti pada Gambar 4.29.



Gambar 4.29  
System Flowchart untuk proses Peminjaman Pustaka pada Sistem Perpustakaan

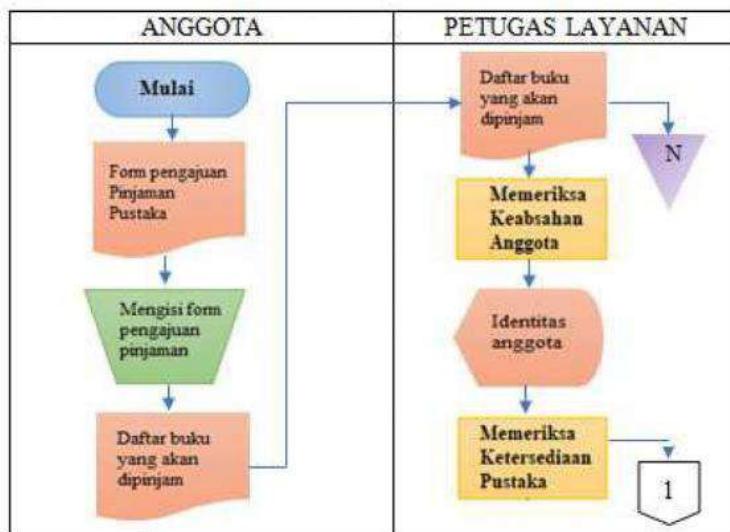
Gambar 4.29 memperlihatkan *system flowchart* untuk proses *peminjaman pustaka* (sub sistem) pada sistem perpustakaan. *System flowchart* juga dapat digunakan untuk menggambarkan keseluruhan proses pada sistem perpustakaan.

Penggambaran *system flowchart* dapat dilakukan dengan sistem *swimlanes* (sistem partisi) seperti pada Gambar 4.29 (seperti "jalur berenang"). Setiap partisi diberi *header* berupa aktor (pelaku) proses. Model penggambaran lainnya adalah dengan tidak menggunakan sistem partisi, dengan cara menyertakan secara langsung pelaku pada setiap nama proses yang digambarkan. Misalnya: *anggota mencatat identitas pustaka, petugas layanan memeriksa keabsahan anggota*, dan seterusnya.

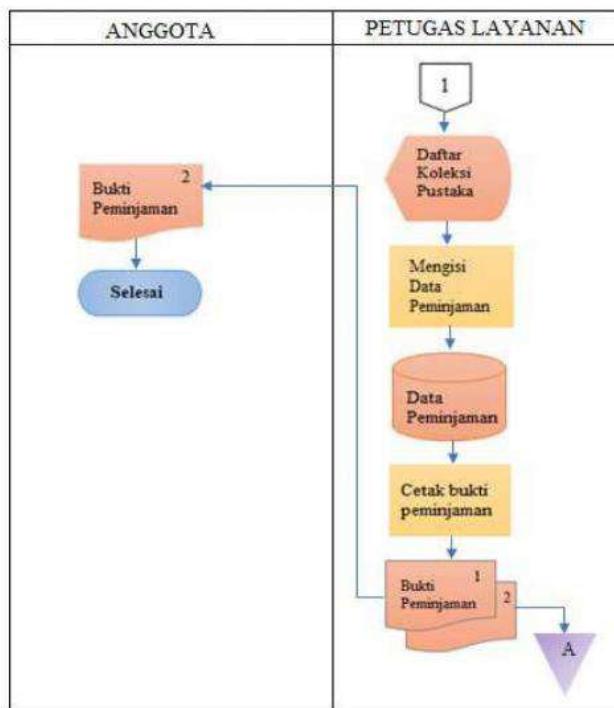
## B. BAGAN ALIR DOKUMEN

Bagan alir dokumen (*document flowchart*), sering juga disebut bagan alir formulir (*form flowchart*) atau *paperwork flowchart*, adalah bagan alir yang menunjukkan arus dari formulir, baik formulir *input* maupun formulir *output* termasuk tembusan-tembusannya. Bagan alir dokumen menggunakan simbol-simbol yang sama dengan yang digunakan pada bagan alir sistem.

Prinsip penggambaran bagan alir dokumen cukup sederhana, yaitu: sebuah proses menghasilkan luaran berupa dokumen atau form *input* dan atau form *output*, selanjutnya dokumen-dokumen tersebut menjadi pedoman untuk melakukan proses selanjutnya dan kembali menghasilkan form *input* dan atau form *output*, demikian seterusnya.



Gambar 4.30  
Dokumen Flowchart Proses Peminjaman Pustaka



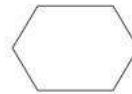
Gambar 4.31  
Lanjutan

Misalkan prosedur bisnis untuk proses *peminjaman pustaka* akan digambarkan dalam bentuk *document flowchart*, hasil penggambarannya disajikan seperti pada Gambar 4.31.

Perbedaan mendasar antara *system flowchart* dan *document flowchart* adalah bahwa *system flowchart* menekankan pada penyajian aktivitas dan urutan proses yang terjadi pada sistem bisnis, termasuk logika proses yang terjadi dalam sistem bisnis. Adapun *document flowchart* menekankan pada dokumen-dokumen (formulir) yang timbul dalam setiap proses sistem bisnis, baik dokumen *input* maupun dokumen *output*, tanpa memperhatikan logika prosedural sistem. Logika prosedural akan membantu proses perancangan logika program, sedangkan *document flowchart* dapat membantu proses perancangan fisik program (antarmuka *input* dan *output*) pada fase perancangan sistem.

### C. BAGAN ALIR PROGRAM

Bagan alir program (*program flowchart*) adalah bagan yang menjelaskan secara rinci langkah-langkah dari proses program komputer (Jogiyanto, 2008). Bagan alir program dibuat dengan menggunakan simbol-simbol seperti pada Gambar 4.32.

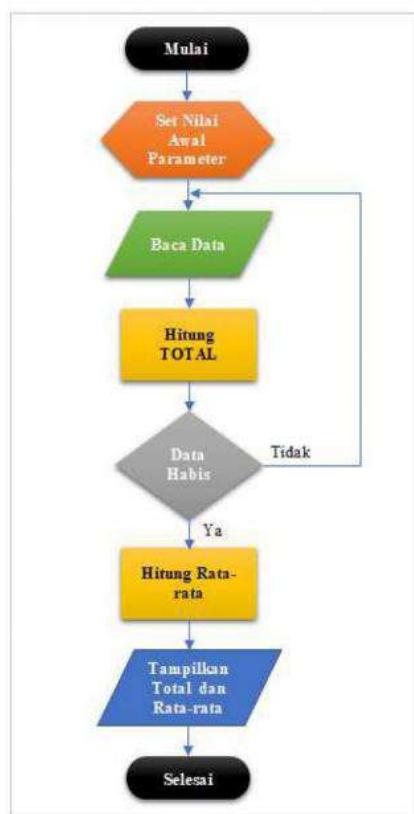
Simbol	Keterangan
Awal / Akhir proses 	Menunjukkan awal proses atau akhir proses
Persiapan 	Memberi nilai awal suatu parameter
Input/Output 	Mewakili data (kegiatan) input/output
Proses 	Proses operasi komputer
Proses terdefinisi 	Menunjukkan suatu operasi yang rinciannya ditunjukkan di tempat lain
Keputusan 	Proses keputusan (logika percabangan)
Garis Alir 	Arus dari proses
Penghubung 	Ke halaman yang sama
	Ke halaman lain (yang berbeda)

**Gambar 4.32**  
Simbol-simbol *Program Flowchart*

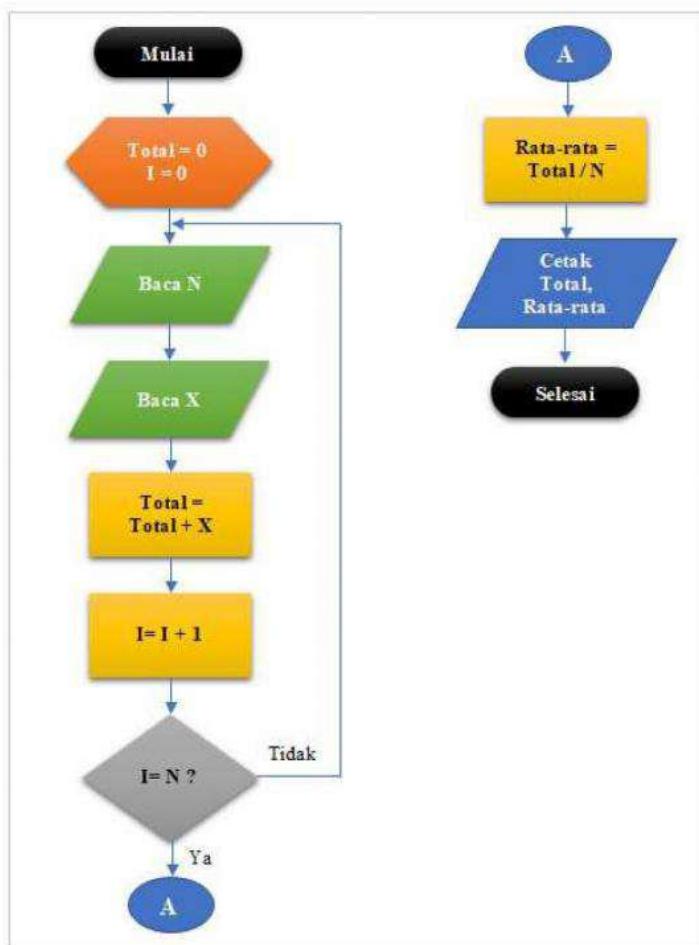
Pada penggambaran program *flowchart* terdapat dua jenis metode, yaitu *Conceptual Flowchart* dan *Detail Flowchart*. *Conceptual Flowchart* menggambarkan

tentang alur dari suatu pemecahan masalah secara global saja, sedangkan *Detail Flowchart* menggambarkan alur pemecahan masalah secara rinci.

Terdapat dua jenis bagan alir program, yaitu bagan alir logika program (*program logic flowchart*) dan bagan alir program komputer terinci (*detailed computer program flowchart*). Bagan alir logika program digunakan untuk menggambarkan tiap-tiap langkah di dalam program komputer secara logika. Bagan alir logika program ini dipersiapkan oleh analis sistem. Bagan alir program komputer terinci (*detailed computer program flowchart*) digunakan untuk menggambarkan instruksi-instruksi program komputer secara terinci. Bagan alir ini dipersiapkan oleh pemrogram, dan biasanya dikemukakan pada fase desain sistem. Gambar 4.33 menunjukkan bagan alir logika program, sedangkan Gambar 4.34 menunjukkan bagan alir program komputer terinci.



Gambar 4.33  
Bagan Alir Logika Program Menghitung Rata-rata



Gambar 4.34  
Bagan Alir Program Komputer Terinci Menghitung Rata-rata



### Latihan

Untuk memperdalam pemahaman Anda mengenai materi di atas, kerjakanlah latihan berikut!

- 1) Jelaskan apa yang dimaksud dengan *flowchart*! Apa keterkaitannya dengan DFD?
- 2) Jelaskan beberapa aturan dasar dalam menggambar *flowchart*!
- 3) Gambarkan *program flowchart* untuk proses 2.1 (DFD Gambar 4.26), dengan merujuk pada contoh "**Proses Bisnis Peminjaman Pustaka**"!

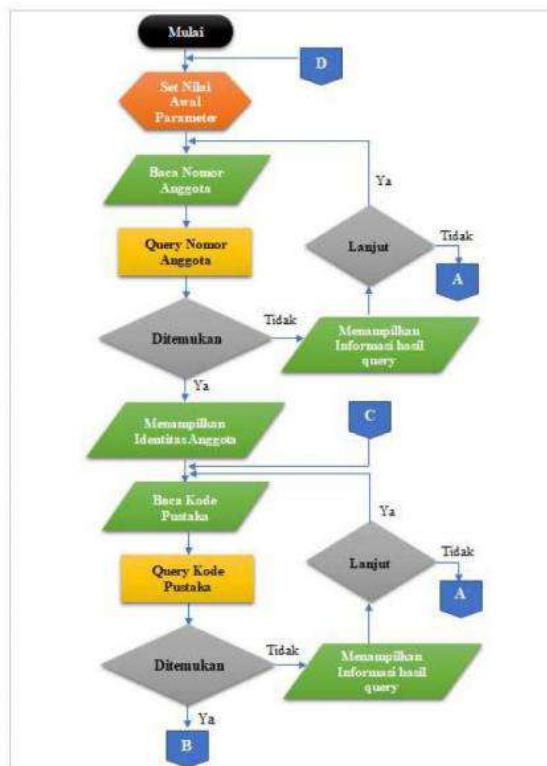
*Petunjuk Jawaban Latihan*

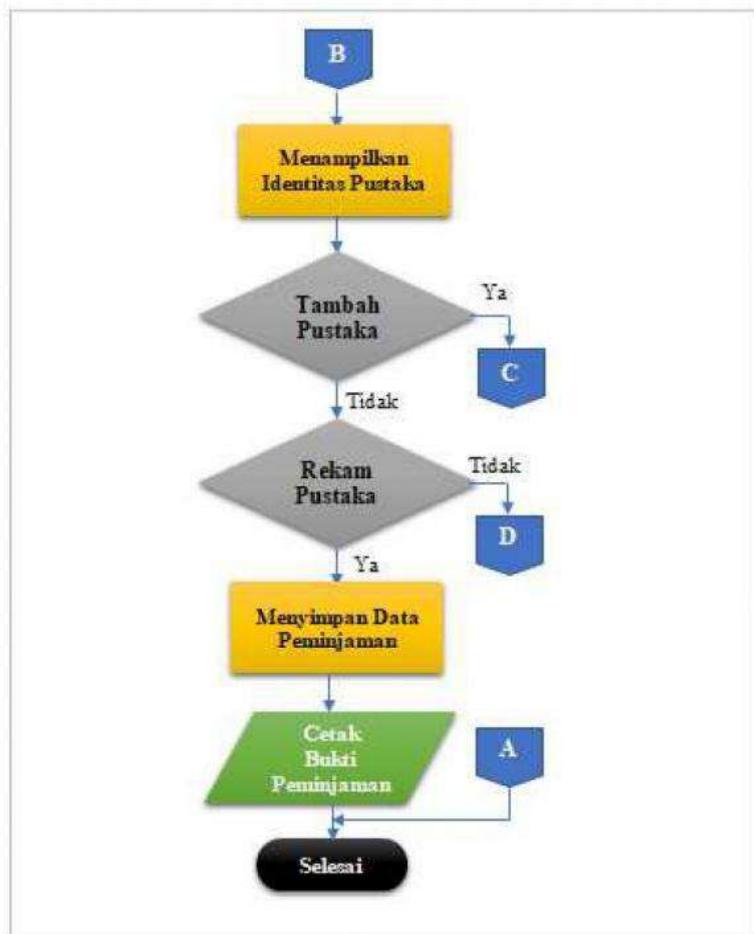
- 1) *Flowchart* adalah bagan-bagan yang menunjukkan alir di dalam program atau prosedur sistem secara logika.  
DFD memiliki keterbatasan dalam menjelaskan logika pemrosesan, terutama logika pemrosesan yang cukup kompleks seperti proses keputusan dan proses perulangan. *Flowchart* adalah salah satu *tools* yang dapat membantu DFD mendeskripsikan proses secara lebih rinci, dengan menjelaskan logika yang terjadi di dalam proses.
- 2) Beberapa aturan dasar dalam penggambaran *flowchart*:
  - a) menggunakan simbol-simbol yang standar;
  - b) penggambaran dilakukan dimulai dari atas ke bawah dan dari bagian kiri dari suatu halaman;
  - c) penamaan proses harus jelas (menggunakan kata yang mewakili objek atau proses dalam diagram, misanya: persiapan, formulir, dan sebagainya);
  - d) tertera dengan jelas dari mana kegiatan akan dimulai dan di mana akan berakhir;
  - e) kegiatan atau proses disajikan dalam urutan yang logis.
- 3) Tahap pembuatan program *flowchart* "proses peminjaman pustaka":
  - a) Mengembangkan prosedur proses menjadi lebih detail, termasuk memunculkan prosedur-prosedur logika pemrosesan (dalam hal ini adalah logika program) seperti keputusan atau perulangan. Misalkan: dalam kasus ini, prosedur logika program "proses peminjaman" dikembangkan lebih detail menjadi:
    - (1) Program membaca dan memverifikasi nomor anggota yang dientri
    - (2) Jika Nomor Anggota tidak ditemukan, program menampilkan informasi bahwa data tidak ditemukan, selanjutnya menawarkan apakah akan mengulang proses atau akan mengakhiri proses.
      - (a) Jika memilih mengakhiri proses, program berakhir
      - (b) Jika memilih melanjutkan proses, program akan kembali membaca Nomor Anggota
    - (3) Jika nomor anggota ditemukan, program menampilkan data Identitas Anggota
    - (4) Program membaca Kode Pustaka yang dientri
    - (5) Jika Kode Pustaka tidak ditemukan, program menampilkan informasi bahwa data tidak ditemukan, selanjutnya menawarkan apakah akan mengulang proses atau akan mengakhiri proses
      - (a) Jika memilih mengakhiri proses, program berakhir

- (b) Jika memilih melanjutkan proses, program akan kembali membaca Kode Pustaka
- (6) Jika Kode Pustaka ditemukan, program menampilkan data Identitas Pustaka pada keranjang pustaka
- (7) Program menawarkan apakah akan menambah pustaka lainnya
  - (a) Jika ia, program mengulang proses membaca Kode Pustaka
  - (b) Jika tidak, program menawarkan apakah akan menyimpan data
- (8) Jika memilih menyimpan data, program menyimpan data Peminjaman, mencetak bukti peminjaman dan mengakhiri proses
- (9) Jika memilih tidak menyimpan data, program membatalkan proses, dan kembali ke posisi awal proses.

b) Menggambar Bagan Alir Program:

Berdasarkan detail prosedur logika program yang telah disusun, selanjutnya dilakukan penggambaran Bagan Alir Program, hasilnya seperti berikut:





### Rangkuman

DFD dengan deskripsi teks sederhana yang menyertainya dapat menyajikan prosedur proses yang cukup untuk mendukung kegiatan dalam fase desain (desain fisik sistem). Namun, kadang-kadang terdapat suatu proses yang cukup rumit sehingga harus dituangkan dalam deskripsi proses yang lebih rinci (terutama proses keputusan dan proses perulangan) menggunakan beberapa *tools* pendukung, termasuk di dalamnya adalah *flowchart*.

*System flowchart* menggambarkan kerja atau apa yang sedang dikerjakan di dalam sistem atau sub sistem secara menyeluruh dan menjelaskan urutan dari prosedur-prosedur yang ada di dalamnya. *Document flowchart* menggambarkan arus dari formulir, baik formulir input maupun formulir output termasuk tembusan-tembusannya, sedangkan *program flowchart* menggambarkan secara rinci langkah-langkah dari proses program komputer.

Penggambaran *flowchart* menggunakan simbol-simbol yang standar dan mengikuti beberapa aturan dasar seperti: mulai penggambaran dari atas ke bawah

dan dari kiri ke kanan, menunjukkan titik awal dan titik akhir penggambaran, serta dalam urutan proses yang logis (urutan yang semestinya).



### Tes Formatif 2

Pilihlah satu jawaban yang paling tepat!

- 1) Bagan alir yang menunjukkan kerja atau apa yang dikerjakan dalam sistem/sub sistem dan menjelaskan urutan dari prosedur-prosedur yang ada di dalamnya, dinamakan ....
  - A. Bagan Alir Program (*Program Flowchart*)
  - B. Bagan Alir Sistem (*System Flowchart*)
  - C. Bagan Alir Dokumen (*Document Flowchart*)
  - D. Bagan Alir Data (*Data Flowchart*)
- 2) Bagan alir yang menunjukkan arus dari formulir, baik formulir *input* maupun formulir *output* termasuk tembusan-tembusannya, dinamakan ....
  - A. Bagan Alir Dokumen (*Documen Flowchart*)
  - B. Bagan Alir Formulir (*Form Flowchart*)
  - C. Bagan Alir Sistem (*System Flowchart*)
  - D. jawaban A dan B benar
- 3) Penekanan dalam penggambaran bagan alir program adalah pada ....
  - A. apa yang dikerjakan dalam sistem dan urutan prosedurnya
  - B. dokumen-dokumen atau formulir yang muncul dalam proses sistem
  - C. logika prosedur dalam sebuah proses sistem
  - D. aliran data dalam sebuah sistem
- 4) Bagan alir program terdiri dari dua macam, yaitu ....
  - A. bagan alir logika program dan bagan arus data
  - B. bagan alir dokumen dan bagan alir formulir
  - C. bagan alir dokumen dan bagan alir program komputer terinci
- 5) Simbol bagan alir sistem di bawah ini menunjukkan ....  



  - A. operasi yang dilakukan di luar proses operasi komputer
  - B. proses dari operasi program komputer
  - C. kegiatan input/output
  - D. jawaban A dan B benar

- 6) Kegiatan input/output pada program *flowchart* digambarkan menggunakan simbol ....
- A. B. C. D.
- 7) Berikut adalah simbol yang digunakan baik pada Bagan Alir Sistem maupun pada Bagan Alir Program, *kecuali* ....
- A. B. C. D.
- 8) Bagan alir sistem (*system flowchart*) dapat dipergunakan oleh analis untuk menyampaikan hasil analisisnya kepada pemakai sistem dalam fase analisis sistem. Pernyataan ini adalah ....
- A. Benar  
B. Salah
- 9) Simbol-simbol yang digunakan pada penggambaran program *flowchart* adalah simbol-simbol yang menunjukkan alat sebagai media input, output, serta penyimpanan dalam proses pengolahan data. Pernyataan ini adalah ....
- A. Benar  
B. Salah
- 10) Bagan alir logika program dan bagan alir program komputer terinci keduanya disiapkan oleh analis sistem. Pernyataan ini adalah ....
- A. Benar  
B. Salah

Cocokkanlah jawaban Anda dengan Kunci Jawaban Tes Formatif 2 yang terdapat di bagian akhir modul ini. Hitunglah jawaban yang benar. Kemudian, gunakan rumus berikut untuk mengetahui tingkat penguasaan Anda terhadap materi Kegiatan Belajar 2.

Tingkat Penguasaan =

$$\frac{\text{Jumlah Jawaban yang Benar}}{\text{Jumlah Soal}} \times 100$$

Arti tingkat penguasaan

<70%

70% - 79%

80% - 89%

90% - 100%

kurang

cukup

baik

baik sekali

Apabila mencapai tingkat penguasaan 80% atau lebih, Anda dapat meneruskan dengan modul selanjutnya. **Bagus!** Jika masih di bawah 80%, Anda harus mengulangi materi Kegiatan Belajar 2, terutama bagian yang belum dikuasai.

## Kunci Jawaban Tes Formatif

### *Tes Formatif 1*

- 1) B
- 2) B
- 3) D
- 4) A
- 5) C
- 6) D
- 7) C
- 8) D
- 9) A
- 10) B

### *Tes Formatif 2*

- 1) B
- 2) D
- 3) C
- 4) D
- 5) B
- 6) A
- 7) B
- 8) A
- 9) B
- 10) B

## Glosarium

- Converging Data Flow** : Gabungan beberapa arus data dari sumber yang berbeda, secara bersama-sama menuju ke tujuan yang sama
- Data Flow** : Simbol Data Flow Diagram yang menunjukkan elemen data atau informasi yang mengalir (baik yang masuk maupun yang keluar) dalam suatu sistem informasi
- Data Flow Diagram** : Diagram yang digunakan untuk menggambarkan proses sistem secara logika (Logical Data Flow Diagram) maupun secara fisik (Physical Data Flow Diagram) dalam sistem terstruktur
- Data Store** : Simbol Data Flow Diagram yang menunjukkan kumpulan data yang disimpan dalam beberapa cara
- Deskripsi Proses** : Penjelasan tambahan mengenai apa yang dilakukan oleh suatu proses, yang tidak diberikan oleh Data Flow Diagram
- Detailed Computer Program Flowchart** : Bagan alir program komputer terinci
- Diverging Data Flow** : Tembusan dari arus data yang sama dari sumber yang sama ke tujuan yang berbeda.
- Document Flowchart** : Bagan yang menunjukkan alir dokumen dalam sebuah proses
- Entitas Luar** : Simbol Data Flow Diagram yang menunjukkan sistem yang berada di luar sistem informasi yang dikembangkan tetapi dapat berinteraksi langsung dengan sistem informasi yang dikembangkan
- Flowchart** : Bagan yang menunjukkan alir sesuatu dalam suatu program atau prosedur sistem
- Hierarchy Chart** : Bagan yang digunakan untuk memecah proses menjadi beberapa tingkatan (sub-sub proses)

- Context Diagram : Diagram teratas dari suatu Data Flow Diagram  
Packet of Data : Dua atau lebih data mengalir dari suatu sumber
- Preview Diagram : Diagram-diagram tingkat selanjutnya dalam Data Flow Diagram, setelah Diagram Konteks (diagram level 0, level 1, dan seterusnya)
- Program Flowchart : Bagan yang menunjukkan alir logika prosedur dalam sebuah proses
- Program Logic Flowchart : Bagan alir logika program
- System Flowchart : Bagan yang menunjukkan alir kerja dan urutan prosedur dalam sebuah proses
- Use Cases : Tools yang menggambarkan bagaimana pengguna eksternal memicu suatu peristiwa (kasus) yang harus ditanggapi oleh sistem informasi

## Daftar Pustaka

- Dennis, A., Wixom, B. H., & Roth, R.M. (2012). *Systems analysis & design*. 5th Edition. United States of America: John Wiley & Sons, Inc.
- Jogiyanto, H.M. (2008). *Analisis & desain sistem informasi: pendekatan terstruktur teori dan praktik aplikasi bisnis*. Edisi 3. Yogyakarta: ANDI.
- McLeod, R. (2004). *Sistem informasi manajemen*. Edisi 8. Jakarta: Indeks.
- Sommerville, I. (2016). *Software engineering*. 10th Edition. USA: Pearson.
- Sulianta, F. (2017). *Teknik perancangan arsitektur sistem informasi*. Yogyakarta: ANDI.

**MSIM4302**  
**Edisi 1**

## **MODUL 05**

# **Pemodelan Analisis dengan Pendekatan Berorientasi Objek**

Bahar S.T., M.Kom.

## Daftar Isi

### Modul 05 5.1

Pemodelan Analisis dengan Pendekatan Berorientasi Objek

<b>Kegiatan Belajar 1</b> Paradigma Berorientasi Objek	5.4
<b>Latihan</b>	5.21
<b>Rangkuman</b>	5.22
<b>Tes Formatif 1</b>	5.23
<b>Kegiatan Belajar 2</b> Pemodelan Fungsional dengan <i>Use Case Diagram</i>	5.26
<b>Latihan</b>	5.43
<b>Rangkuman</b>	5.45
<b>Tes Formatif 2</b>	5.45
<b>Kunci Jawaban Tes Formatif</b>	5.48
<b>Glosarium</b>	5.49
<b>Daftar Pustaka</b>	5.51



## Pendahuluan

Pada pemodelan sistem yang menggunakan pendekatan tradisional, dekomposisi masalah terjadi berbasis proses (proses-sentris) atau berbasis data (data-sentris). Ketika memodelkan sistem dunia nyata, antara proses dan data saling terkait sangat erat sehingga sulit untuk memilih salah satu di antaranya (data atau proses) sebagai fokus utama. Ketidaksesuaian dengan dunia nyata ini memunculkan metodologi baru berorientasi objek yang menggunakan urutan fase SDLC berbasis RAD, tetapi berusaha untuk menyeimbangkan penekanan antara proses dan data. Hal ini dilakukan dengan memfokuskan dekomposisi masalah pada objek yang berisi data dan proses.

Pendekatan berorientasi objek untuk mengembangkan sistem informasi, secara teknis, dapat menggunakan metodologi tradisional yang disajikan pada Modul 2 (*waterfall, parallel, V-model, iterative, system prototyping, dan throwaway prototyping*). Namun demikian, pendekatan berorientasi objek paling terkait dengan metodologi pengembangan *iterative/RAD*. Perbedaan utama antara pendekatan tradisional seperti desain terstruktur dan pendekatan berorientasi objek adalah bagaimana masalah terurai.

Setelah mempelajari modul ini mahasiswa diharapkan secara lebih rinci mampu:

1. menjelaskan konsep kerja dan karakteristik dasar sistem berorientasi/berbasis objek;
2. menjelaskan konsep analisis dan desain sistem berorientasi objek;
3. menjelaskan tentang *Unified Modeling Language* (UML) dan diagram-diagram yang terdapat di dalamnya;
4. menjelaskan tentang *use case diagram* sebagai penggerak utama untuk semua teknik diagram UML;
5. menjelaskan langkah-langkah dalam membuat *use case diagram*;
6. menjelaskan *use case description* sebagai alat untuk merinci detail informasi tentang *use case*;
7. menjelaskan notasi-notasi yang terdapat dalam *use case diagram*;
8. membuat *use case diagram* untuk memodelkan fungsionalitas sistem.

## Paradigma Berorientasi Objek

**S**uatu hal yang menarik perhatian pada kegiatan analisis dan desain sistem saat ini adalah penerapan pendekatan berorientasi objek. Pendekatan berorientasi objek memandang sistem sebagai kumpulan objek mandiri, termasuk data dan proses. Seperti diketahui, metodologi analisis dan desain sistem tradisional dapat berupa data-sentris (berpusat pada data) atau proses-sentris (berpusat pada proses). Sampai pada pertengahan tahun 1980-an, pengembang sistem harus memisahkan data dan proses untuk membangun sistem yang dapat berjalan pada komputer *mainframe* pada masa itu. Seiring peningkatan daya prosesor pada satu sisi dan penurunan harga prosesor pada sisi yang lain, pendekatan berorientasi objek menjadi layak dipertimbangkan. Konsekuensinya, pengembang fokus pada pembangunan sistem yang lebih efisien, yang memungkinkan analis untuk bekerja dengan data dan proses sistem secara bersamaan sebagai sebuah objek. Benda-benda ini (data dan proses) dapat dibangun sebagai potongan-potongan individu dan kemudian disatukan untuk membentuk suatu sistem.

Meskipun beberapa kalangan merasa bahwa beralih ke sistem berorientasi objek akan secara radikal mengubah bidang analisis dan desain sistem dan SDLC, penggabungan objek adalah suatu proses yang berkembang di mana teknik berorientasi objek secara bertahap diintegrasikan ke dalam SDLC utama. Oleh karena itu, penting bagi analis untuk memahami apa itu sistem berorientasi objek dan mengapa itu populer di dunia industri, serta menjadi akrab dengan beberapa teknik berorientasi objek populer yang mungkin perlu digunakan pada proyek pengembangan sistem.

Salah satu tantangan utama mempelajari pendekatan berorientasi objek untuk mengembangkan sistem informasi adalah munculnya terminologi-terminologi baru yang begitu banyak. Dalam kegiatan belajar ini, akan dikemukakan tinjauan umum tentang karakteristik dasar sistem berorientasi objek, memperkenalkan analisis dan desain sistem berorientasi objek dasar dan proses terpadu, serta memberikan tinjauan umum *tools/bahasa pemodelan UML (Unified Modeling Language)*.

### A. KARAKTERISTIK DASAR SISTEM BERORIENTASI OBJEK

Sistem berorientasi objek berfokus pada struktur dan perilaku sistem dalam bentuk modul-modul kecil yang mencakup data dan proses. Modul-modul kecil ini

dikenal sebagai objek. Berikut akan dijelaskan karakteristik dasar sistem berorientasi objek.

### 1. Kelas dan Objek

Kelas (*class*) adalah *template* umum yang digunakan untuk mendefinisikan dan membuat *instance* (contoh) spesifik, atau objek (Dennis, 2012). Dalam bahasa yang lebih sederhana, kelas adalah representasi dari objek-objek yang sejenis. Istilah lain dari sebuah kelas adalah *blue print* atau cetakan yang digunakan untuk menciptakan objek. Jika diibaratkan sebuah cetakan kue, cetakan kue itu adalah kelas sedangkan kue adalah objeknya.

Istilah lain yang membingungkan adalah *instance*. Sebuah objek adalah sebuah *instance* dari sebuah kelas. Mungkin akan lebih mudah dimengerti dengan terlebih dahulu memahami arti dari *instance*; *instance* dalam Bahasa Indonesia bisa diartikan *contoh*. Kalau kita artikan lagi, objek adalah *instance* (contoh) dari sebuah kelas. Objek dan *instance* ini secara praktiknya sama. Sementara itu, objek (*object*) adalah gambaran dari entitas, baik di dunia nyata maupun berupa konsep.

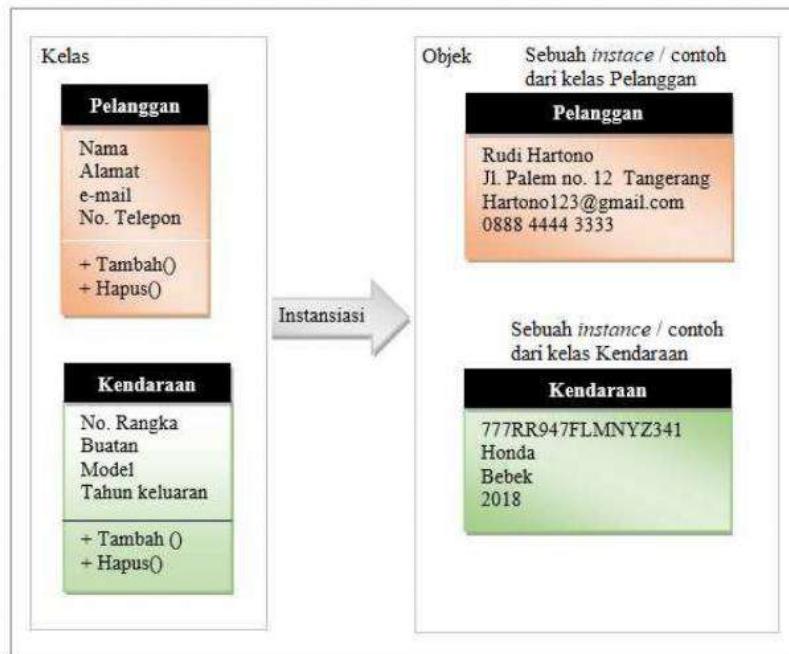
Contoh:

- a. Objek yang mewakili dunia nyata: komputer, mobil, dan lain-lain.
- b. Objek yang berbentuk konsep: transaksi bank, permintaan pembelian, dan lain-lain.

Setiap objek dikaitkan dengan kelas. Sebagai contoh, semua objek yang menangkap/menampung informasi tentang pasien dapat jatuh ke dalam kelas yang disebut kelas *pasien*. Setiap objek memiliki atribut (misalnya: nama pasien, alamat pasien, dan tanggal lahir pasien) dan metode (misalnya: memasukkan contoh pasien baru atau menghapus data). Dalam sebuah kelas yang bernama *pasien*, mungkin terdapat objek seperti: Ari, Bandung, 7/4/1980; Anisa, Jakarta, 18/6/1985; dan Rudy, Surabaya, 20/1/1975.

Contoh yang bukan kelas *pasien* adalah: Rahman, Bandung, 13/12/1990; Blacky, Semarang, 10/11/2018. Rahman yang lahir di Bandung pada tanggal 12 Desember 1990 adalah seorang warga masyarakat yang dirawat di sebuah puskesmas, sedangkan Blacky yang lahir di Surabaya pada tanggal 10 Nopember 2018 adalah seekor kucing yang dirawat di sebuah klinik hewan. Dalam kasus ini, kumpulan manusia dan hewan bukanlah objek yang dapat disebut kelas, karena tidak sejenis, sebagaimana definisi kelas yang merupakan representasi dari objek-objek yang sejenis. Gambar 5.1 memperlihatkan gambaran lebih detail tentang kelas dan objek.

## 5.6 Pemodelan Analisis Dengan Pendekatan Berorientasi Objek



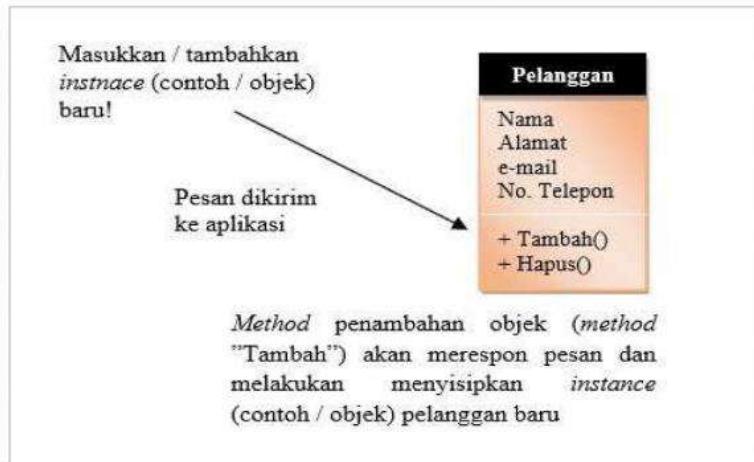
Gambar 5.1  
Kelas dan Objek

Salah satu aspek yang membingungkan pada pengembangan sistem berorientasi objek adalah kenyataan bahwa di sebagian besar bahasa pemrograman berorientasi objek, baik kelas maupun *instance* kelas dapat memiliki atribut (*attribute*) dan *method*. Atribut biasa disebut *properties/field*, merupakan variabel *global* yang dimiliki oleh sebuah kelas yang tidak melakukan operasi, tetapi kepadanya dilakukan operasi sehingga dapat mengubah nilai dari atribut tersebut. Atribut dapat memiliki hak akses *private*, *public*, maupun *protected*. Sebuah atribut yang dinyatakan sebagai *private* hanya dapat diakses secara langsung oleh kelas yang membungkusnya, sedangkan kelas lainnya tidak dapat mengakses atribut ini secara langsung. Sebuah atribut yang dinyatakan sebagai *public* dapat diakses secara langsung oleh kelas lain di luar kelas yang membungkusnya. Sebuah atribut yang dinyatakan sebagai *protected* tidak dapat diakses secara langsung oleh kelas lain di luar kelas yang membungkusnya, kecuali kelas yang mengaksesnya adalah kelas turunan dari kelas yang membungkusnya.

Atribut dan *method* kelas cenderung digunakan untuk memodelkan atribut (atau *method*) yang menangani masalah yang terkait dengan semua *instance* kelas. Misalnya, untuk membuat objek pelanggan baru, pesan dikirim ke kelas pelanggan untuk membuat *instance* baru itu sendiri. Namun, dari sudut pandang analisis dan desain sistem, fokus utama bahasan adalah pada atribut dan *method* objek, dan bukan pada kelas.

## 2. Message dan Method

*Method* mengimplementasikan perilaku objek. Sebuah *method* adalah sebuah tindakan yang dapat dilakukan oleh objek. *Method* sangat mirip *function* (fungsi) atau prosedur (*procedure*) dalam bahasa pemrograman tradisional seperti C, COBOL, atau Pascal. *Messages* (pesan) adalah informasi yang dikirim ke objek untuk memicu *method*. Pesan pada dasarnya adalah *function* atau *procedure* panggilan dari satu objek ke objek lain.



Gambar 5.2  
Ilustrasi *Messages* dan *Method*

*Contoh:*

Jika terdapat pelanggan baru yang akan bergabung ke organisasi, sistem akan mengirim pesan menyisipkan/menambahkan pelanggan baru tersebut ke objek pelanggan. Objek pelanggan akan menerima pesan (instruksi) dan melakukan apa yang perlu dilakukan untuk memasukkan pelanggan baru ke dalam sistem (menjalankan *methodnya*), seperti pada Gambar 5.2.

## 3. Enkapsulasi dan Menyembunyikan Informasi

Ide enkapsulasi (*encapsulation*) dan penyembunyian informasi saling terkait dalam sistem yang berorientasi objek. Pendekatan berorientasi objek menggabungkan proses dan data menjadi entitas holistik (objek). Enkapsulasi merupakan suatu mekanisme untuk menyembunyikan atau memproteksi suatu proses dari kemungkinan interferensi atau penyalahgunaan dari luar sistem dan sekaligus menyederhanakan penggunaan sistem tersebut (Dennis, 2012). Teknik enkapsulasi memastikan pengguna sebuah objek tidak dapat mengganti keadaan dari sebuah objek dengan cara yang tidak layak. Hanya metode dalam objek tersebut yang diberi ijin untuk mengakses keadaannya. Setiap objek mengakses *interface* yang menyebutkan bagaimana objek

lainnya dapat berinteraksi dengannya. Objek lainnya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut.

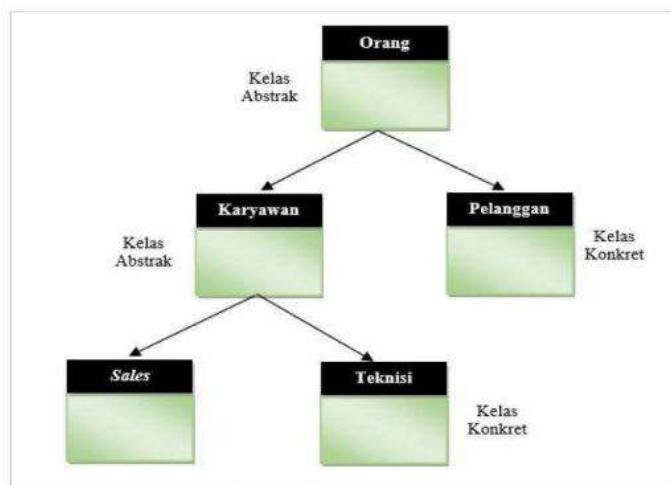
Konsep penyembunyian informasi bukanlah hal yang baru. Penyembunyian informasi pertama kali diperkenalkan dalam pengembangan sistem terstruktur. Prinsip penyembunyian informasi adalah bahwa hanya informasi yang diperlukan untuk menggunakan sebuah modul perangkat lunak yang dipublikasikan kepada pengguna modul. Hal ini berarti bahwa hanya informasi yang diperlukan untuk diteruskan ke modul dan informasi yang dikembalikan dari modul yang diterbitkan. Tepatnya, bagaimana modul mengimplementasikan fungsionalitas yang diperlukan adalah hal yang tidak berkaitan. Tidak perlu dipedulikan bagaimana objek melakukan fungsinya, yang terpenting adalah *function* tersebut terjadi. Dalam sistem berorientasi objek, kombinasi antara enkapsulasi dan prinsip penyembunyian informasi menunjukkan bahwa prinsip penyembunyian informasi diterapkan pada objek, dan bukan menerapkannya pada *function* atau *process*. Dengan demikian, objek diperlakukan seperti kotak hitam.

Faktanya bahwa menggunakan objek dengan memanggil *method* adalah kunci pada sistem penggunaan kembali objek (*reusability*), karena hal itu dapat melindungi kerja internal objek dari perubahan oleh sistem luar, dan itu akan menjaga sistem tidak terpengaruh ketika perubahan dilakukan pada objek. Pada Gambar 5.2, sebuah pesan (tambah pelanggan baru) dikirim ke suatu objek, namun algoritma internal yang diperlukan untuk menanggapi pesan tersebut disembunyikan pada bagian lain dari sistem. Satu-satunya informasi yang perlu diketahui oleh suatu objek adalah serangkaian operasi, atau *method*, yang dapat dilakukan oleh objek lain dan pesan apa yang perlu dikirim untuk memicu mereka.

#### 4. *Inheritance*

*Inheritance* (pewarisan) merupakan konsep mewariskan *attribute* dan *method* yang dimiliki oleh sebuah *class* kepada *class* turunannya. Dengan konsep ini *class* yang dibuat cukup mendefinisikan *attribute* dan *method* yang spesifik di dalamnya, sedangkan *attribute* dan *method* yang lebih umum akan didapatkan dari *class* yang menjadi induknya (Dennis, 2012).

Literatur pemodelan data menyarankan penggunaan metode pewarisan untuk mengidentifikasi tingkat objek yang lebih tinggi, atau lebih umum. Kumpulan atribut dan *method* yang umum dapat diorganisir ke dalam *superclass/kelas induk (parent class)* berada di atas, dan *subclass/kelas anak (child class)* berada di bagian bawah.



Gambar 5.3  
Hierarki Kelas

*Contoh:*

Pada Gambar 5.3, dapat diilustrasikan seseorang adalah *superclass* untuk kelas *karyawan* dan kelas *pelanggan*. *Karyawan*, pada gilirannya, adalah *superclass* bagi tenaga penjualan (*Sales*) dan *Teknisi* toko. Perhatikan bagaimana suatu kelas (misalnya: *Karyawan*) dapat berfungsi sebagai *superclass* dan *subclass* secara bersamaan. Hubungan antara kelas dan *superclass*-nya dikenal sebagai hubungan *A-Kind-Of* (jenis). Sebagai contoh, pada Gambar 5.3, seorang *Sales* adalah jenis *Karyawan*, yang merupakan jenis *Orang*.

*Subclass* mewarisi atribut dan *method* dari *superclass* di atasnya. Artinya, setiap *subclass* berisi atribut dan *method* dari *superclass* induknya. Gambar 5.3 menunjukkan bahwa baik *Karyawan* dan *Pelanggan* adalah *subclass* *Orang* dan oleh karena itu akan mewarisi atribut dan *method* kelas *Orang*. Sistem pewarisan membuatnya lebih mudah untuk mendefinisikan kelas. Alih-alih mengulangi atribut dan *method* dalam kelas *Karyawan* dan *Pelanggan* secara terpisah, atribut dan *method* yang umum untuk keduanya ditempatkan di kelas *Orang* dan diwarisi oleh kelas-kelas di bawahnya. Perhatikan, lebih efisien mana objek dengan sistem hierarki kelas daripada objek yang sama tanpa hierarki, pada Gambar 5.4.

*Contoh lain:*

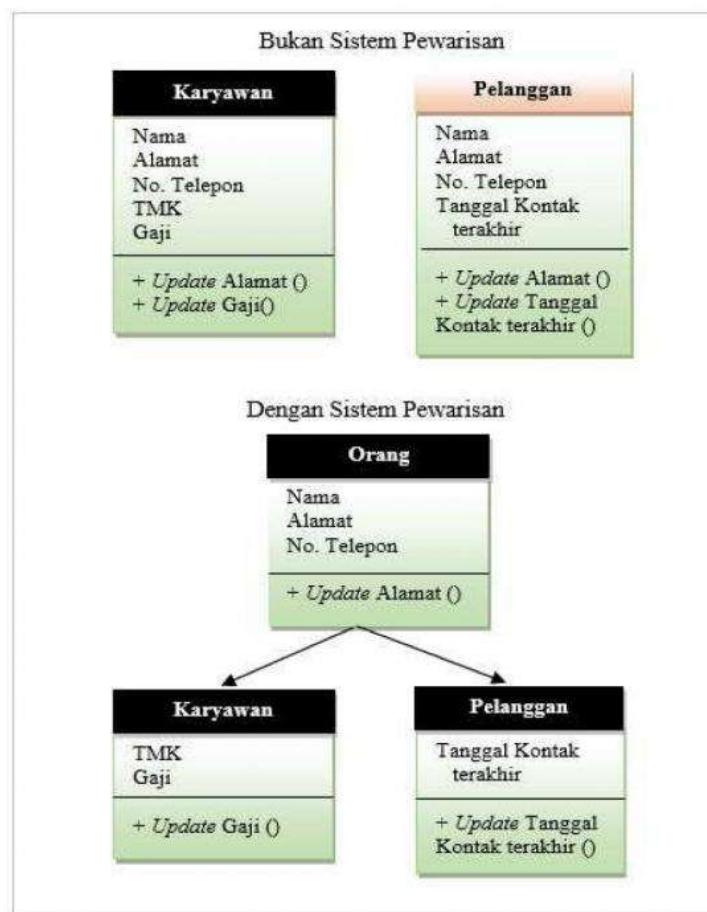
Kelas *mamalia* memiliki seluruh sifat yang dimiliki oleh *binatang*, demikian halnya juga *Macan*, *Kucing*, *Paus*, dan *Monyet* memiliki seluruh sifat yang diturunkan dari kelas *mamalia*. Dengan konsep ini, karakteristik yang dimiliki oleh kelas *binatang* cukup didefinisikan dalam kelas *binatang* saja. Kelas *mamalia* tidak perlu mendefinisikan ulang apa yang telah dimiliki oleh kelas *binatang*, karena sebagai kelas turunannya, ia akan mendapatkan karakteristik dari kelas *binatang* secara otomatis.

## 5.10 Pemodelan Analisis Dengan Pendekatan Berorientasi Objek

Demikian juga dengan kelas *Macan*, *Kucing*, *Paus*, dan *Monyet*, hanya perlu mendefinisikan karakteristik spesifik yang dimiliki oleh kelasnya masing-masing.

Dengan memanfaatkan konsep pewarisan ini dalam pemrograman, maka hanya perlu mendefinisikan karakteristik yang lebih umum yang didapatkan dari kelas di mana ia diturunkan.

Sebagian besar kelas di seluruh hierarki akan mengarah ke *instance*; setiap kelas yang memiliki *instance* disebut *kelas konkret (concrete class)*. Misalnya, jika Ary Mulyadi dan Irhami adalah contoh *instance* dari kelas *Pelanggan*, *Pelanggan* akan dianggap sebagai kelas yang konkret. Beberapa kelas tidak menghasilkan *instance*, karena mereka digunakan hanya sebagai *template* untuk kelas lain yang lebih spesifik (terutama kelas-kelas yang terletak tinggi dalam hierarki). Mereka adalah kelas abstrak (*abstract class*). *Orang* adalah contoh dari kelas abstrak.



Gambar 5.4  
*Inheritance*

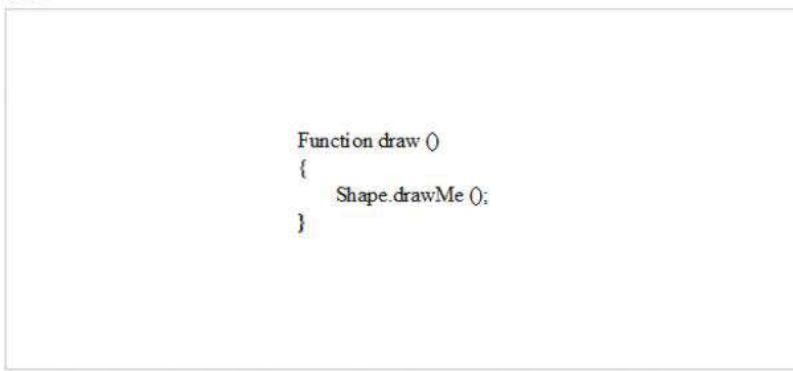
Beberapa keuntungan sistem *inheritance* ini seperti: (1) *subclass* menyediakan *state/behavior* yang spesifik yang membedakannya dengan *superclass*, hal ini akan memungkinkan programmer untuk menggunakan ulang *source code* dari *superclass* yang telah ada; (2) programmer dapat mendefinisikan *superclass* khusus yang bersifat generik, yang disebut *kelas abstrak*, untuk mendefinisikan kelas dengan *behavior* dan *state* secara umum.

## 5. Polimorfisme dan Pengikatan Dinamis

Polimorfisme (*polymorphism*) berarti bahwa pesan yang sama dapat ditafsirkan secara berbeda oleh berbagai kelas objek (Dennis, 2012). Misalnya, perintah "tambah *pasien*" adalah sesuatu yang berbeda dari "tambah *janji ketemu*". Dengan demikian, informasi yang berbeda perlu dikumpulkan dan disimpan. Untungnya, kita tidak perlu khawatir dengan bagaimana sesuatu dilakukan saat menggunakan objek. Kita cukup mengirim pesan ke objek, dan objek itu akan bertanggung jawab untuk menafsirkan pesan dengan tepat.

*Contoh:*

Misalkan kita akan mengembangkan sistem berbasis grafis. Saat pengguna ingin menggambar sesuatu, entah itu garis, atau lingkaran, atau elips, sistem akan memunculkan perintah *gambar*. Sistem akan mengenali berbagai bentuk gambar; masing-masing dengan perilakunya sendiri-sendiri. Seandainya pengguna akan menggambar sebuah garis, maka perintah untuk menggambar garis akan dibangkitkan. Tanpa *polymorphism*, kode program untuk fungsi gambar mungkin akan terlihat seperti Gambar 5.5.



Gambar 5.5  
Contoh Kode Program tanpa *Polymorphism*

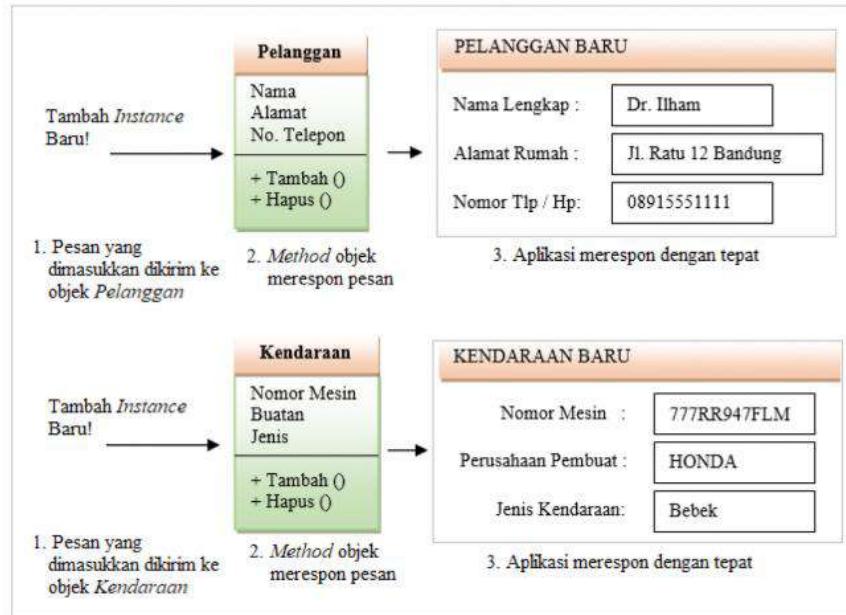
Dengan *polymorphism*, kode program untuk menggambar dilakukan hanya dengan memanggil fungsi *DrawMe()* untuk setiap objek yang akan digambar, seperti yang diperlihatkan pada Gambar 5.6.

## 5.12 Pemodelan Analisis Dengan Pendekatan Berorientasi Objek

```
Function Shape.DrawMe ()  
{  
    Switch Shape.Type  
    Case "Circle"  
        Shape.drawCircle () ;  
    Case "Elipse"  
        Shape.drawElipse () ;  
    Case "Line"  
        Shape.drawLine () ;  
    EndSwitch  
}
```

Gambar 5.6  
Contoh Kode Program dengan *Polymorphism*

Pada Gambar 5.6, masing-masing bentuk (lingkaran, garis, dan elips) memiliki fungsi DrawMe() untuk menggambar bentuknya masing-masing. Perhatikan contoh lainnya pada Gambar 5.7, bagaimana setiap objek merespon dengan tepat (dan berbeda), meskipun pesannya identik.



Gambar 5.7  
Polymorphism dan Encapsulation

Polimorfisme dimungkinkan melalui pengikatan dinamis (*dynamic binding*). Pengikatan dinamis adalah teknik yang menunda mengidentifikasi suatu jenis objek hingga waktu berjalan (*run-time*). Dengan demikian, *method* spesifik yang sebenarnya dipanggil tidak dipilih oleh sistem berorientasi objek sampai sistem berjalan. Ini

berbeda dengan pengikatan statis (*static binding*). Dalam sistem yang terikat secara statis, jenis objek akan ditentukan pada saat kompilasi. Oleh karena itu, pengembang harus memilih *method* mana yang harus dipanggil, alih-alih membikarkan sistem untuk melakukannya. Inilah sebabnya mengapa dalam sebagian besar bahasa pemrograman tradisional ditemukan logika keputusan yang rumit berdasarkan pada berbagai jenis objek dalam suatu sistem. Misalnya, dalam bahasa pemrograman tradisional, alih-alih mengirim pesan "Gambar dirimu" ke berbagai jenis objek grafis yang disebutkan sebelumnya, kita harus menulis logika keputusan dengan menggunakan pernyataan kasus (*case*) atau serangkaian pernyataan "jika" (*if*) untuk menentukan objek grafis apa yang akan digambar, dan harus memberi nama setiap *function* menggambar secara berbeda (misalnya: *draw-square*, *draw-circle*, atau *draw-triangle*). Ini jelas membuat sistem jauh lebih rumit dan lebih sulit untuk dipahami.

## B. ANALISIS DAN DESAIN SISTEM BERORIENTASI OBJEK

Hingga 1995, konsep berorientasi objek sangat populer, tetapi diimplementasikan dengan berbagai cara oleh pengembang yang berbeda. Setiap pengembang memiliki metodologi dan notasi sendiri (misalnya: Booch, Coad, Moses, OMT, OOSE, dan SOMA1). Kemudian, pada tahun 1995, Perangkat Lunak Rasional menyatukan tiga pemimpin industri untuk menciptakan pendekatan tunggal dalam pengembangan sistem berorientasi objek. Grady Booch, Ivar Jacobson, dan James Rumbaugh bekerja untuk membuat seperangkat teknik diagram yang dikenal sebagai *Unified Modeling Language* (UML). Menurut penciptanya, pendekatan berorientasi objek apapun untuk mengembangkan sistem informasi harus memiliki sifat (Dennis, 2012): (1) *use case driven*, (2) *architecture centric*, (3) *iterative* dan *incremental*.

### 1. Use Case Driven

*Use case driven* berarti *use case* adalah alat pemodelan utama yang digunakan untuk mendefinisikan perilaku sistem. Sebuah *use case* menggambarkan bagaimana pengguna berinteraksi dengan sistem untuk melakukan beberapa aktivitas, seperti *mengambil pemesanan*, *membuat reservasi*, atau *mencari informasi*. *Use case* digunakan untuk mengidentifikasi dan mengkomunikasikan kebutuhan untuk sistem kepada programmer yang harus menulis sistem.

*Use case* pada dasarnya sederhana, yaitu hanya fokus pada satu aktivitas pada satu waktu. Sebaliknya, model proses sistem yang telah dipaparkan sebelumnya (model tradisional seperti DFD) jauh lebih kompleks karena mengharuskan pengembang untuk membuat model seluruh sistem. Dalam model proses tradisional, setiap kegiatan bisnis didekomposisi menjadi satu set subproses, yang selanjutnya didekomposisi menjadi lebih banyak subproses, dan seterusnya. Sebaliknya, *use case* fokus hanya pada satu aktivitas pada satu waktu, sehingga pengembangan model jauh lebih sederhana.

Seperti yang telah kita lihat pada pembahasan tentang DFD, pengembangan sistem tradisional dapat memanfaatkan *use case* untuk mendekomposisi proses bisnis lengkap menjadi subproses-subproses. Pendekatan berorientasi objek menekankan pada hanya satu aktivitas *use case* pada satu waktu dan mendistribusikan *use case* tersebut melalui serangkaian objek yang saling berkomunikasi dan berkolaborasi. Fokus tunggal ini membantu membuat analisis dan desain menjadi lebih sederhana.

## 2. *Architecture Centric*

Setiap pendekatan modern dalam analisis dan desain sistem harus berbasis pada arsitektur. *Architecture Centric* berarti bahwa arsitektur yang mendasari sistem yang dikembangkan mendorong spesifikasi, konstruksi, dan dokumentasi sistem. Ada tiga pandangan arsitektur yang terpisah dari suatu sistem, namun saling terkait, yaitu: fungsional, statis, dan dinamis. *Tampilan fungsional* menggambarkan perilaku eksternal sistem dari perspektif pengguna. Pada permukaan, pandangan ini terkait erat dengan pendekatan pemodelan proses dalam analisis dan desain terstruktur. *Tampilan statis* menggambarkan struktur sistem dalam hal atribut, metode, kelas, hubungan, dan pesan. Pandangan ini sangat mirip dengan pendekatan pemodelan data dalam analisis dan desain terstruktur. *Tampilan dinamis* menggambarkan perilaku internal sistem dalam hal pesan yang dikirimkan antara objek dan menyatakan perubahan dalam suatu objek. Pandangan ini dalam banyak hal menggabungkan proses dan pendekatan pemodelan data, karena pelaksanaan *method* dapat menyebabkan perubahan keadaan pada objek penerima.

## 3. *Iterative dan Incremental*

Pendekatan berorientasi objek menekankan pengembangan iteratif dan inkremental yang menjalani pengujian terus menerus sepanjang umur proyek. Setiap iterasi sistem akan membawa sistem lebih dekat dengan kebutuhan akhir pengguna.

## 4. **Manfaat Analisis dan Desain Sistem Berorientasi Objek**

Sebelumnya telah dijelaskan beberapa konsep utama yang terkait pendekatan berorientasi objek untuk pengembangan sistem. Mahasiswa mungkin bertanya-tanya bagaimana konsep-konsep ini mempengaruhi kinerja tim proyek. Jawabannya sederhana, konsep seperti polimorfisme, enkapsulasi, dan pewarisan memungkinkan analis untuk memecah sistem yang kompleks menjadi komponen (modul-modul) yang lebih kecil agar lebih mudah dikelola, lebih mudah dikerjakan secara individu, dan menyatukannya kembali membentuk suatu sistem. Modularitas ini membuat pengembangan sistem lebih mudah dipahami, lebih mudah untuk dibagikan di antara anggota tim proyek untuk dikerjakan secara paralel, dan lebih mudah untuk berkomunikasi dengan pengguna dan mengkonfirmasi seberapa baik sistem memenuhi kebutuhan pengguna.

Dengan membagi pekerjaan dalam modul-modul kecil dan konsep penggerjaan paralel, tim proyek sebenarnya menciptakan suatu bagian yang dapat digunakan kembali (tanpa harus dibuat dari awal) pada sistem lain, atau digunakan sebagai titik awal untuk proyek lain. Pada akhirnya, ini dapat menghemat waktu, karena proyek baru tidak harus dimulai dari awal.

Akhirnya, banyak orang berpendapat bahwa *berorientasi objek* adalah cara yang jauh lebih realistik untuk berpikir tentang realitas (dunia nyata). Dalam menggunakan atau mengoperasikan sistem, pengguna biasanya tidak memikirkan tentang data atau proses; sebaliknya, mereka hanya melihat suatu sistem sebagai kumpulan unit logis yang telah mengandung keduanya (data dan proses). Jadi berkomunikasi menggunakan model berbasis objek dapat meningkatkan interaksi antara pengguna dan analis atau pengembang. Gambar 5.8 merangkum konsep-konsep utama dari pendekatan berorientasi objek dan bagaimana masing-masing berkontribusi pada manfaat yang baru saja dijelaskan (Dennis, 2012).

KONSEP	DUKUNGAN	BERDAMPAK PADA
<i>Class, Object, Method, Dan Message</i>	<ul style="list-style-type: none"> <li>- Cara memikirkan bisnis secara lebih realistik</li> <li>- Unit yang terpadu, yang berisi data dan proses</li> </ul>	<ul style="list-style-type: none"> <li>- Komunikasi yang lebih baik antara pengguna dan analis atau pengembang</li> <li>- Penggunaan kembali suatu objek</li> <li>- Memiliki sistem yang terpadu</li> </ul>
<i>Encapsulation Dan Information Hiding</i>	Penggabungan (interkoneksi) unit secara bebas (fleksibel)	<ul style="list-style-type: none"> <li>- Penggunaan kembali objek</li> <li>- Lebih sedikit efek yang ditimbulkan jika terjadi perubahan pada objek atau sistem itu sendiri</li> <li>- Memiliki desain sistem yang dapat diinterkoneksi secara fleksibel</li> </ul>
<i>Inheritance</i>	Mengizinkan penggunaan kelas sebagai <i>template</i> standar bagi kelas lain yang diciptakan	<ul style="list-style-type: none"> <li>- Mengurangi kerangkapan proses</li> <li>- Pembuatan kelas baru yang lebih cepat</li> <li>- Standar dan konsisten di seluruh bagian yang dikembangkan</li> <li>- Kemudahan dalam mendukung pengecualian</li> </ul>
<i>Polymorphism</i>	Jumlah pesan yang minimal, yang ditafsirkan oleh objek itu sendiri	<ul style="list-style-type: none"> <li>- Aktivitas Pemrograman menjadi lebih sederhana</li> <li>- Kemudahan dalam mengganti atau mengubah objek dalam suatu sistem</li> <li>- Lebih sedikit efek yang ditimbulkan jika terjadi perubahan pada objek atau sistem itu sendiri</li> </ul>

KONSEP	DUKUNGAN	BERDAMPAK PADA
<i>Use Case Driven</i>	Mengizinkan pengguna dan analis untuk fokus pada bagaimana pengguna akan berinteraksi dengan sistem untuk melakukan satu aktivitas	<ul style="list-style-type: none"> <li>- Pemahaman yang lebih baik dalam pengumpulan kebutuhan pengguna</li> <li>- Komunikasi yang lebih baik antara pengguna dan analis</li> </ul>
<i>Architecture Centric, tampilan fungsional, tampilan statis, dan tampilan dinamis</i>	Melihat sistem yang berkembang dari berbagai sudut pandang	<ul style="list-style-type: none"> <li>- Pemahaman yang lebih baik dan pemodelan kebutuhan pengguna</li> <li>- Penggambaran sistem informasi yang lebih lengkap</li> </ul>
Pengembangan berulang dan inkremental	Pengujian dan penyempurnaan secara berkelanjutan pada sistem yang dikembangkan	<ul style="list-style-type: none"> <li>- Memenuhi kebutuhan nyata pengguna</li> <li>- Sistem dengan kualitas yang lebih tinggi</li> </ul>

Gambar 5.8  
Manfaat Pendekatan Berorientasi Objek

### C. UNIFIED MODELING LANGUAGE

*Unified Modeling Language* (UML) adalah istilah berbasis objek dan teknik diagram untuk memodelkan proyek pengembangan sistem dari analisis hingga desain. UML adalah keluarga notasi grafis yang didukung oleh meta-model tunggal, yang membantu pendeskripsian dan mendesain sistem perangkat lunak, khususnya sistem yang dibangun menggunakan pemrograman berorientasi objek.

Salah satu pertanyaan menarik tentang UML adalah apakah UML digunakan untuk pemodelan konseptual atau untuk pemodelan perangkat lunak. Sebagian besar orang akrab dengan penggunaan UML dalam pemodelan perangkat lunak. Di dalam perspektif perangkat lunak, elemen-elemen UML memetakan secara langsung elemen-elemen dalam sebuah sistem perangkat lunak. Dalam perspektif konseptual, UML mewakili deskripsi konsep-konsep sebuah bidang kajian. Tidak ada aturan-aturan baku dalam perspektif, sebagai buktinya, terdapat sejumlah besar perbedaan dalam penggunaan UML (Fowler, 2005). Beberapa peranti lunak (*software*) secara otomatis membaca kode program (*source code*) untuk menghasilkan diagram UML (*reverse engineering*), atau sebaliknya UML yang telah dirancang dapat menghasilkan kode program yang akan menjadi pijakan dan panduan untuk mengawali pengembangan aplikasi (*forward engineering*). Hal tersebut merupakan sebuah perspektif perangkat lunak. Jika kita menggunakan diagram UML untuk mencoba memahami prosedur

sistem *pengumpulan aset* bersama-sama dengan kelompok akuntan, maka kita menggunakan UML dalam kerangka pikir konseptual.

Perbedaan cara pandang lainnya adalah tentang apa sebenarnya inti UML. Sebagian besar pengguna UML, terutama yang membuat sketsa berpendapat bahwa inti UML adalah diagram. Meskipun demikian, pencipta UML memandang diagram sebagai hal sekunder, sedangkan inti UML adalah meta-model. Diagram-diagram hanya representasi dari meta-model.

Jadi, setiap kali akan membaca buku yang menyebutkan UML, sangat penting untuk mengetahui sudut pandang si pengarang. Baru setelah itu kita dapat memahami argumen-argumen yang acapkali dilontarkan secara berbeda oleh orang-orang yang berbeda pandangan.

*Object Management Group* (OMG), sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP, mendefinisikan UML sebagai sebuah bahasa grafis untuk memvisualisasikan (notasi grafis), menentukan/menspesifikasi (memodelkan secara tepat, jelas, dan lengkap), membangun (dimensi desain untuk dapat diimplementasikan dalam bahasa pemrograman), dan mendokumentasikan (dari fase awal hingga fase akhir) artefak sistem perangkat lunak secara intensif. UML menawarkan cara standar untuk membuat cetak biru sistem, termasuk hal-hal konseptual seperti proses bisnis dan fungsi sistem serta hal-hal konkret seperti pernyataan dalam bahasa pemrograman, skema basis data, dan *reusable software component* (David, 2008).

*Object Modeling Technique* (OMT) telah berevolusi sejak tahun 1990-an, hingga menghasilkan UML versi 0.8 pada tahun 1995 dan UML versi 0.9 pada tahun 1996. Pada tahun 1997 lahir UML versi 1.1 dengan 8 buah diagram, yaitu: Use Case Diagram, Activity Diagram, Sequence Diagram, Collaboration Diagram, Class Diagram, Statechart Diagram, Componen Diagram, dan Deployment Diagram. Pada tahun 1999, lahir UML versi 1.3 dengan 9 buah diagram (dengan penambahan Business Use Case Diagram), dan pada tahun 2001 lahir UML versi 1.4 menjadi 10 buah diagram, dengan penambahan Object Diagram (Nugroho, 2005).

Pada tahun 2002 lahirlah UML versi 2.0, menjadi 14 buah diagram, dengan penambahan dan penggantian, yaitu: Use Case Diagram, Activity Diagram, Sequence Diagram, Communication Diagram (Collaboration Diagram pada versi 1.x), Class Diagram, Behavioral State Machine & Protocol State Machine Diagram (Statechart Diagram pada versi 1.x), Componen Diagram, Deployment Diagram, Composite Structure Diagram, Interaction Overview Diagram, Object Diagram, Package Diagram, dan Timing Diagram. Gambar 5.9 memberikan ikhtisar diagram ini. Diagram dipecah menjadi dua kelompok utama yaitu: untuk memodelkan struktur suatu sistem dan untuk memodelkan perilaku (Dennis, 2012).

Diagram struktur (*structure diagram*) mendefinisikan arsitektur statis dari sebuah model. Diagram ini digunakan untuk memodelkan hal-hal yang membentuk model, yaitu kelas, obyek, antarmuka, dan komponen fisik. Diagram ini juga digunakan untuk memodelkan relasi dan ketergantungan antar elemen. Secara spesifik, diagram struktur digunakan untuk mewakili data dan hubungan statis yang ada dalam sistem.

Diagram perilaku (*behavior diagram*) menangkap berbagai variasi interaksi dan status yang terjadi seketika dalam model dari waktu ke waktu; melacak bagaimana sistem akan bertindak pada lingkungan nyata, dan mengamati efek dari sebuah operasi atau peristiwa, termasuk hasil-hasilnya. Secara spesifik, diagram perilaku menyediakan analis cara untuk menggambarkan hubungan dinamis antara *instance* atau objek yang merepresentasikan sistem, memungkinkan pemodelan perilaku dinamis objek secara individu sepanjang siklus hidup mereka. Diagram perilaku mendukung analis dalam memodelkan kebutuhan fungsional sistem yang dikembangkan.

Tergantung pada fase apa dalam proses pengembangan sistem, diagram yang berbeda memainkan peran yang sangat penting. Dalam beberapa kasus, diagram yang sama digunakan selama proses pengembangan. Dalam hal ini, diagram dimulai dari yang bersifat konseptual dan abstrak. Ketika sistem berkembang, diagram juga berkembang ke arah yang lebih rinci, yang pada akhirnya mengarah pada pembuatan dan pengembangan kode program. Dengan kata lain, diagram dimulai dari mendokumentasikan kebutuhan untuk membuat desain. Secara keseluruhan, notasi yang konsisten, integrasi antara teknik diagram, dan penerapan diagram di seluruh proses pengembangan menjadikan UML bahasa yang kuat dan fleksibel untuk analis dan pengembang sistem.

NAMA DIAGRAM	PENGGUNAAN	FASE
<b>Digaram Struktur (<i>Structure Diagrams</i>)</b>		
<i>Class</i>	Mengilustrasikan hubungan antara kelas yang dimodelkan dalam sistem.	Analisis, Desain
<i>Object</i>	<ul style="list-style-type: none"> <li>- Mengilustrasikan hubungan antara objek yang dimodelkan dalam sistem.</li> <li>- Berfungsi ketika aktual instance dari kelas akan meng-komunikasikan model secara lebih baik.</li> </ul>	Analisis, Desain
<i>Package</i>	Mengelompokkan elemen UML lainnya secara bersama-sama untuk membentuk tingkat konstruksi yang lebih tinggi.	Analisis, Desain, Implementasi
<i>Deployment</i>	Menunjukkan arsitektur fisik sistem. Dapat juga digunakan untuk menunjukkan komponen perangkat lunak yang digunakan pada arsitektur fisik	Desain Fisik, Implementasi
<i>Component</i>	Mengilustrasikan hubungan fisik antara komponen perangkat lunak.	Desain Fisik, Implementasi
<i>Composite Structure</i>	Mengilustrasikan struktur internal kelas — yaitu, hubungan di antara bagian-bagian kelas	Analisis, Desain
<b>Diagram Perilaku (<i>Behavioral Diagrams</i>)</b>		
<i>Use Case</i>	Menangkap kebutuhan bisnis untuk sistem dan menggambarkan interaksi antara sistem dan lingkungannya.	Analisis
<i>Activity</i>	Menggambarkan arus kerja (proses) bisnis yang tidak tergantung pada kelas, arus aktivitas dalam <i>use case</i> , atau desain terperinci dari suatu <i>method</i>	Analisis, Desain

NAMA DIAGRAM	PENGGUNAAN	FASE
<i>Sequence</i>	Memodelkan perilaku objek dalam <i>use case</i> . Berfokus pada perintah berdasarkan waktu dari sebuah aktivitas.	Analisis, Desain
<i>Communication</i>	Memodelkan perilaku objek dalam <i>use case</i> . Berfokus pada komunikasi di antara sekumpulan objek yang berkolaborasi dari suatu aktivitas.	Analisis, Desain
<i>Interaction Overview</i>	Menggambarkan tinjauan umum tentang aliran kontrol suatu proses.	Analisis, Desain
<i>Timing</i>	Mengilustrasikan interaksi yang terjadi di antara sekumpulan objek dan perubahan status yang objek-objek tersebut lalui sepanjang sumbu waktu.	Analisis, Desain
<i>Behavioral State Machine</i>	Memeriksa perilaku suatu kelas.	Analisis, Desain
<i>Protocol State Machine</i>	Mengilustrasikan dependensi di antara berbagai antarmuka kelas.	Analisis, Desain

Sumber: Dennis (2012)

Gambar 5.9  
Ringkasan Diagram UML 2.0

Tergantung pada fase apa dalam proses pengembangan sistem, diagram yang berbeda memainkan peran yang sangat penting. Dalam beberapa kasus, diagram yang sama digunakan selama proses pengembangan. Dalam hal ini, diagram dimulai dari yang bersifat konseptual dan abstrak. Ketika sistem berkembang, diagram juga berkembang ke arah yang lebih rinci, yang pada akhirnya mengarah pada pembuatan dan pengembangan kode program. Dengan kata lain, diagram dimulai dari mendokumentasikan kebutuhan untuk membuat desain. Secara keseluruhan, notasi yang konsisten, integrasi antara teknik diagram, dan penerapan diagram di seluruh proses pengembangan menjadikan UML bahasa yang kuat dan fleksibel untuk analis dan pengembang sistem.

Perlu diingat bahwa UML bukanlah metodologi, sehingga tidak secara formal mengamanatkan bagaimana menerapkan suatu teknik diagram tertentu (Dennis, 2012). Banyak organisasi yang bereksperimen dengan UML dan mencoba memahami bagaimana mengadopsi tekniknya ke dalam metodologi analisis dan desain sistem mereka. Dalam banyak kasus, diagram UML hanya digunakan untuk menggantikan sebuah teknik terstruktur tertentu (misalnya: *class diagram* menggantikan diagram ERD).

Fowler (2005) berpandangan bahwa meskipun UML menyediakan sejumlah bentuk diagram untuk membantu memodelkan sebuah aplikasi, tidak berarti UML menyediakan seluruh diagram yang dibutuhkan. Di banyak kasus, diagram yang berbeda dapat berguna dan orang tidak perlu ragu-ragu menggunakan diagram non-UML jika tidak ada diagram UML yang memenuhi kebutuhan pada suatu kesempatan.

Orang-orang juga sering bertanya mengenai dari mana harus memulai UML dan apakah harus menggunakan seluruh diagram dalam UML? Tidak ada orang (bahkan para pembuat UML), mengerti atau menggunakan semuanya. Kebanyakan orang menggunakan sebagian kecil UML dalam pekerjaannya. Orang harus mencari bagian UML yang berguna baginya dan rekan kerjanya.

Untuk mulai bekerja dengan UML, disarankan untuk memusatkan perhatian pada lima diagram dasar UML: Class Diagram, Use Case Diagram, Sequence Diagram, Activity Diagram, dan State Machine Diagram.

#### D. DIAGRAM UML DASAR

Seseorang dapat menjelaskan secara luas cara menggunakan UML untuk mengembangkan sistem, akan tetapi modul ini tidak memiliki banyak ruang untuk menjelaskan secara keseluruhan. Lima teknik diagram UML telah mendominasi proyek berorientasi objek: *class diagram*, *use case diagram*, *sequence diagram*, *activity diagram*, dan *state machine diagram*. Teknik diagram lainnya berguna untuk tujuan khusus tertentu. Kelima teknik dasar ini membentuk inti UML seperti yang dibahas dalam buku ini.

Kelima teknik diagram ini terintegrasi dan digunakan bersama-sama untuk menggantikan DFD dan ERD di SDLC tradisional. Diagram *use case* biasanya digunakan untuk merangkum himpunan *use case* untuk fungsi-fungsi logis dari sistem (atau keseluruhan sistem). Pada beberapa situasi tertentu, sebelum menggambar *use case*, didahului dengan menggambarkan *activity diagram* yang menyajikan proses bisnis sistem, untuk mengidentifikasi fungsi-fungsi logis dalam bisnis. Kemudian *class diagram*, *sequence diagram*, dan *state machine diagram* digunakan untuk mendefinisikan lebih jauh sistem yang dikembangkan dari berbagai perspektif.

Diagram *use case* selalu dibuat pertama, tetapi urutan diagram lainnya setelah *use case* dibuat tergantung pada proyek dan preferensi pribadi para analis. Sebagian besar analis mulai dengan *class diagram* (menunjukkan objek apa yang terkandung di dalamnya dan bagaimana mereka terkait, seperti ERD) atau *sequence diagram* (menunjukkan bagaimana objek berinteraksi secara dinamis, seperti DFD) dan *activity diagram* untuk menunjukkan konteks *use case* serta rincian logika bagaimana sebuah *use case* yang rumit berjalan. Akan tetapi dalam praktiknya, proses ini sering berulang. Mengembangkan *sequence diagram* sering menyebabkan perubahan dalam *class diagram* dan sebaliknya, sehingga analis sering bergerak bolak-balik di antara keduanya, sehingga masing-masing berubah pada saat mereka mendefinisikan sistem. Secara umum, *state machine diagram* dikembangkan kemudian, setelah *class diagram* telah didefinisikan.

Dalam kegiatan belajar selanjutnya, kita akan mulai dengan *use case diagram* untuk memodelkan fungsional sistem, berpindah ke *activity diagram* untuk memodelkan proses bisnis dan atau logika proses, *class diagram* dan *object diagram*

untuk memodelkan struktur sistem, diakhiri dengan *sequence diagram* dan *state machine diagram* untuk memodelkan perilaku sistem.



## Latihan

Untuk memperdalam pemahaman Anda mengenai materi di atas, kerjakanlah latihan berikut!

- 1) Jelaskan konsep kerja sistem berorientasi objek!
- 2) Jelaskan beberapa karakteristik dasar sistem berorientasi objek berikut: *object; class; method; concrete class; inheritance*!
- 3) Bagaimana pendekatan berbasis objek berbeda dari pendekatan berbasis data dan proses untuk pengembangan sistem?
- 4) Bagaimana pendekatan berbasis objek dapat meningkatkan proses pengembangan sistem?
- 5) Apa itu UML? Bagaimana itu mendukung pendekatan berbasis objek untuk pengembangan sistem?

### Petunjuk Jawaban Latihan

- 1) Pendekatan berorientasi objek memandang sistem sebagai kumpulan objek mandiri, termasuk data dan proses. Dengan kata lain, sistem berorientasi objek berfokus pada struktur dan perilaku sistem dalam bentuk modul-modul kecil (yang dikenal sebagai objek) yang mencakup data dan proses.
- 2) Beberapa karakteristik dasar dalam konteks sistem berorientasi objek.
  - a) *Object*: perwujudan/gambaran dari suatu entitas, baik di dunia nyata maupun yang berupa konsep
  - b) *Class*: perwakilan atau representasi dari *object-object* yang sejenis. Istilah lainnya adalah *blue print* atau *template* yang digunakan untuk menciptakan *object*.
  - c) *Method*: suatu tindakan (*function* atau *procedure*) yang dapat dilakukan oleh *object* di dalam suatu *class*. Mirip dengan *function* (fungsi) atau *procedure* (prosedur) dalam bahasa pemrograman tradisional.
  - d) *Concrete class*: *class* yang memiliki/menghasilkan *instance*
  - e) *Inheritance*: merupakan konsep mewariskan *attribute* dan *method* yang dimiliki oleh sebuah *class* kepada *class* turunannya. Dengan konsep ini *class* yang dibuat cukup mendefinisikan *attribute* dan *method* yang spesifik di dalamnya, sedangkan *attribute* dan *method* yang lebih umum akan didapatkan dari *class* yang menjadi induknya.

- 3) Perbedaan mendasar antara kedua pendekatan tersebut terletak pada bagaimana proses penguraian masalah. Pada pendekatan berbasis objek, dekomposisi masalah adalah pada objek yang berisi data dan proses, sedangkan pada pendekatan tradisional, dekomposisi masalah adalah berbasis proses atau berbasis data (antara data dan proses harus dipisahkan). Antara data dan proses ketika memodelkan sistem dunia nyata, antara proses dan data terkait sangat erat sehingga sulit untuk memilih salah satu sebagai fokus utama.
- 4) Pendekatan berbasis objek dapat meningkatkan proses pengembangan sistem:
  - a) Konsep seperti polimorfisme, enkapsulasi, dan pewarisan memungkinkan analis untuk memecah sistem yang kompleks menjadi komponen (modul-modul) yang lebih kecil agar lebih mudah dikelola, lebih mudah dikerjakan secara individu, dan menyatukannya kembali membentuk suatu sistem. Modularitas ini membuat proses pengembangan sistem lebih mudah dipahami, lebih mudah untuk dibagikan di antara anggota tim proyek untuk dikerjakan secara paralel (mempercepat proses pengembangan), dan lebih mudah untuk berkomunikasi dengan pengguna dan mengkonfirmasi seberapa baik sistem memenuhi kebutuhan pengguna.
  - b) Dengan membagi pekerjaan dalam modul-modul kecil dan konsep pelaksanaan paralel, tim proyek sebenarnya menciptakan suatu bagian yang dapat digunakan kembali (tanpa harus dibuat dari awal) pada sistem lain, atau digunakan sebagai titik awal untuk proyek lain. Konsep ini dapat menghemat waktu, karena proyek baru tidak harus dimulai dari awal.
- 5) UML (*Unified Modeling Language*) adalah bahasa pemodelan yang terdiri atas sekumpulan alat (*tools*) yang digunakan untuk melakukan abstraksi/memodelkan sebuah sistem atau perangkat lunak berbasis objek. UML memungkinkan analis untuk menguraikan masalah kompleks menjadi komponen atau modul-modul yang lebih kecil (objek), sehingga lebih mudah dikelola melalui serangkaian notasi yang diterima secara umum.



## Rangkuman

---

Teknik berorientasi objek memandang sistem sebagai kumpulan objek mandiri yang mengandung/mengintegrasikan data dan proses. Ide-ide yang mendasari teknik berorientasi objek adalah ide-ide lama yang berevolusi dari pendekatan tradisional. Ide-ide ini sekarang menjadi praktis karena rasio antara biaya (yang murah) dan peningkatan kinerja perangkat keras komputer modern, jika dibandingkan dengan rasio sebaliknya atas kedua hal tersebut di masa lalu. Saat ini, biaya pengembangan perangkat lunak modern terutama terdiri dari biaya yang terkait dengan pengembangan perangkat lunak itu sendiri dan bukan biaya untuk perangkat komputernya. Oleh karena itu, pendekatan

berorientasi objek untuk mengembangkan sistem informasi sangat menjanjikan dalam mengendalikan biaya ini.

Objek adalah orang, tempat, atau benda yang akan diperoleh informasinya. Setiap objek memiliki atribut yang memberi gambaran tentang objek tersebut dan perilakunya, yang dijelaskan oleh *method* yang menentukan apa yang dapat dilakukan objek. Objek dikelompokkan ke dalam kelas, yang merupakan kumpulan objek berupa berbagai atribut dan *method* umum. Kelas-kelas dapat diatur dengan cara hierarkis di mana kelas tingkat rendah atau subkelas mewarisi atribut dan *method* dari super kelas di atasnya, untuk mengurangi redundansi dalam pembangunan sistem. Objek berkomunikasi satu sama lain dengan mengirim pesan, yang memicu *method*. Polimorfisme memungkinkan pesan untuk ditafsirkan secara berbeda oleh berbagai jenis objek. Bentuk komunikasi ini memungkinkan kita untuk memperlakukan objek sebagai kotak hitam dan mengabaikan cara kerja dalam objek. Enkapsulasi dan penyembunyian informasi memungkinkan suatu objek menyembunyikan proses dan data di dalamnya dari objek lain.

Analisis dan desain berorientasi objek menggunakan UML memungkinkan analis untuk menguraikan masalah kompleks menjadi komponen yang lebih kecil dan lebih mudah dikelola melalui serangkaian notasi yang diterima secara umum. UML adalah seperangkat teknik diagram yang menyediakan representasi grafis yang cukup kaya untuk memodelkan proyek pengembangan sistem apapun, mulai dari analisis hingga implementasi. Biasanya, pendekatan analisis dan desain yang paling obyektif saat ini adalah menggunakan UML untuk menggambarkan sistem yang dikembangkan. Pengguna tidak berpikir dalam hal data atau proses, melainkan hanya melihatnya sebagai kumpulan objek yang berkolaborasi. Dengan demikian, analisis dan desain yang berorientasi objek menggunakan UML memungkinkan analis untuk berinteraksi dengan pengguna, menggunakan objek dari lingkungan pengguna, dan mengantikan serangkaian sistem yang memisahkan proses dan data.



### Tes Formatif 1

Pilihlah satu jawaban yang paling tepat!

- 1) Metodologi pengembangan sistem berorientasi objek mempunyai tiga karakteristik utama, *kecuali* ....
  - A. *encapsulation*
  - B. *inheritance*
  - C. *prototyping*
  - D. *polymorphism*
- 2) Sesuatu yang dapat disamakan maknanya dengan *object* adalah ....
  - A. *attribute*
  - B. *instance*
  - C. *method*
  - D. *message*

#### 5.24 Pemodelan Analisis Dengan Pendekatan Berorientasi Objek

---

- 3) Variabel global yang dimiliki oleh sebuah kelas, yang tidak melakukan operasi, namun sebaliknya kepadanya dilakukan operasi sehingga dapat mengubah nilai dari atribut tersebut, disebut ....
  - A. *attribute*
  - B. *instance*
  - C. *method*
  - D. *message*
- 4) Suatu mekanisme menyembunyikan atau memproteksi suatu proses dari kemungkinan interferensi atau penyalahgunaan dari luar sistem, dan hanya metode dalam objek tersebut yang diberi ijin untuk mengakses keadaannya, disebut ....
  - A. *inheritance*
  - B. *encapsulation*
  - C. *polymorphism*
  - D. semua jawaban salah
- 5) Suatu konsep yang memungkinkan digunakannya suatu *interface* yang sama untuk memerintah suatu *object* agar melakukan suatu tindakan yang mungkin secara prinsip sama tetapi secara proses berbeda, dikenal dengan istilah ....
  - A. *inheritance*
  - B. *encapsulation*
  - C. *polymorphism*
  - D. *method*
- 6) Beberapa kelas tidak menghasilkan *instance*, karena mereka digunakan hanya sebagai *template* untuk kelas lain yang lebih spesifik (terutama kelas-kelas yang terletak tinggi dalam hierarki). Jenis kelas tersebut adalah ....
  - A. *concrete class*
  - B. *superclass*
  - C. *subclass*
  - D. semua jawaban A, B, dan C salah
- 7) Konsep *inheritance* pada pendekatan berorientasi objek dapat memberikan dampak berupa ....
  - A. mengurangi kerangkapan proses
  - B. penulisan prosedur yang standar dan konsisten di seluruh bagian sistem yang dikembangkan
  - C. proses penciptaan kelas baru yang lebih cepat
  - D. semua jawaban A, B, dan C benar

- 8) Kelompok diagram UML yang mendukung analis dalam memodelkan kebutuhan fungsional sistem yang dikembangkan, adalah ....
- Behavior Diagram*
  - Structure Diagram*
  - Use Case Diagram*
  - semua jawaban A, B, dan C benar
- 9) Diagram berikut yang termasuk kelompok *Structure Diagram* dalam UML adalah ....
- Use Case Diagram*
  - Class Diagram*
  - Sequence Diagram*
  - Activity Diagram*
- 10) Diagram berikut termasuk kelompok *Behavior Diagram* dalam UML, *kecuali* ....
- Use Case Diagram*
  - Class Diagram*
  - State Machine Diagram*
  - Activity Diagram*

Cocokkanlah jawaban Anda dengan Kunci Jawaban Tes Formatif 1 yang terdapat di bagian akhir modul ini. Hitunglah jawaban yang benar. Kemudian, gunakan rumus berikut untuk mengetahui tingkat penguasaan Anda terhadap materi Kegiatan Belajar 1.

$$\text{Tingkat Penguasaan} = \frac{\text{Jumlah Jawaban yang Benar}}{\text{Jumlah Soal}} \times 100$$

Arti tingkat penguasaan

<70%	70% - 79%	80% - 89%	90% - 100%
kurang	cukup	baik	baik sekali

Apabila mencapai tingkat penguasaan 80% atau lebih, Anda dapat meneruskan dengan Kegiatan Belajar 2. **Bagus!** Jika masih di bawah 80%, Anda harus mengulangi materi Kegiatan Belajar 1, terutama bagian yang belum dikuasai.

## Pemodelan Fungsional dengan *Use Case* Diagram

Diagram *use case* adalah sebuah diagram yang menjelaskan apa yang harus dilakukan oleh sistem pada level konseptual. *Use case* adalah penggerak utama untuk semua teknik diagram UML. Diagram *use case* menggambarkan fungsi dasar sistem, yaitu apa yang dapat dilakukan oleh pengguna dan bagaimana sistem harus merespon tindakan pengguna. Masing-masing teknik diagram UML dibangun atas dasar fungsionalitas dengan cara yang berbeda serta memiliki tujuan yang berbeda (seperti dijelaskan pada Gambar 5.9). Pada tahap awal analisis, analis pertama-tama mengidentifikasi satu *use case* untuk setiap bagian utama dari sistem dan membuat dokumentasi yang menyertai (narasi atau skenario tentang bagaimana sistem tersebut digunakan) untuk menggambarkan masing-masing fungsi secara rinci.

*Use case* digunakan untuk menjelaskan dan mendokumentasikan interaksi yang diperlukan antara pengguna dengan sistem dalam menyelesaikan suatu tugas. *Use case* dibuat untuk membantu tim pengembangan memahami langkah-langkah yang terlibat dalam mencapai tujuan pengguna secara lebih lengkap. Setelah dibuat, *use case* sering dapat digunakan untuk mendapatkan kebutuhan fungsional yang lebih rinci untuk sistem baru.

Biasanya, diagram *use case* dibuat di awal SDLC, ketika analis mengumpulkan dan menentukan kebutuhan untuk sistem (tahap elisitasi kebutuhan) (Fowler, 2005). Pada saat itu analis dapat dengan mudah langsung berkomunikasi kepada pengguna seperti halnya ketika akan membuat DFD. Penting untuk diingat bahwa *use case* mewakili sebuah gambaran eksternal sebuah sistem. Oleh karena itu, tidak ada korelasi antara *use case* dan *class* di dalam sistem.

Seringkali kita mendengar orang-orang berbicara mengenai *use case sistem* dan *use case bisnis*. Pada umumnya, *use case sistem* merupakan sebuah interaksi dengan perangkat lunak, sedangkan sebuah *use case bisnis* berbicara tentang bagaimana sebuah bisnis menanggapi sebuah kejadian/peristiwa (Fowler, 2005).

Seorang analis dapat menggunakan diagram *use case* untuk menggambarkan fungsionalitas sistem pada tingkat yang sangat tinggi (fungsi-fungsi sistem secara umum) (Dennis, 2012). Jauh sebelumnya, Fowler (2005) telah mengingatkan bahaya kesalahan besar dalam penggambaran *use case* adalah ketika membuatnya sangat rumit dan akhirnya macet. *Use case* yang terlalu banyak dan detail, tidak akan ada yang dapat membaca dan memahaminya. Jika kita kekurangan, setidaknya kita telah memiliki

sebuah dokumentasi yang pendek yang dapat dibaca oleh orang, yang merupakan langkah awal untuk memunculkan pertanyaan-pertanyaan yang lebih detail.

Sejalan dengan pandangan Fowler (2005) dan Dennis (2012), Sugiarti (2013) mengemukakan bahwa seringkali sebuah *use case* dianggap sebagai sebuah *function* atau item menu aplikasi. Hal ini keliru sebab *function* menjelaskan apa yang dilakukan oleh sistem secara detail, sedangkan *use case* menggambarkan fungsi-fungsi sistem secara umum.

#### *Contoh Kasus*

Seseorang menggambarkan sebuah diagram *use case* tentang apa yang dilakukan oleh seorang *actor* (seorang dosen) dalam empat *case* yaitu: aktivitas *menambah*, *menghapus*, *merubah*, dan *mencetak* kegiatan tri dharma perguruan tinggi dosen pada sebuah aplikasi.

Pada contoh kasus di atas, menurut pandangan Sugiarti (2013), diagram tersebut telah memperlihatkan proses penguraian fungsi-fungsi (*functional decomposition*), yaitu mengurai proses (*memasukkan* data tri dharma) ke dalam bagian-bagian yang lebih kecil (*menambah*, *menghapus*, *mengubah*, dan *mencetak* data tri dharma). Hal tersebut kelihatannya keliru, karena mengurai sebuah proses menjadi proses-proses yang lebih detail. Analoginya seperti ini: kegiatan *menambah*, *menghapus*, *mengubah*, dan *mencetak* data tri dharma sejatinya tidak akan pernah dapat dilakukan jika sebelumnya tidak pernah melakukan kegiatan *memasukkan* data tri dharma. Dalam kasus ini, semestinya *use case*-nya adalah *memasukkan* data tri dharma, dan bukan *menambah*, *menghapus*, *mengubah*, dan *mencetak* kegiatan tri dharma.

Terlepas adanya perbedaan-perbedaan pandangan mengenai penggunaan *use case*, setidaknya *use case* telah ada untuk beberapa waktu dan hanya terdapat sedikit standarisasi pada penggunaannya. UML tidak menjelaskan isi penting sebuah *use case* dan hanya mengakuinya sebagai sebuah diagram yang penting. Hasilnya, orang-orang akan menemui opini-opini yang berbeda tentang *use case* (Fowler, 2005).

Membuat diagram *use case* dilakukan melalui dua langkah: Pertama, tim pengembang proyek bekerja sama dengan pengguna untuk menulis deskripsi *use case* berbasis teks; Kedua, tim proyek menerjemahkan kata kunci yang terdapat di dalam deskripsi *use case* ke dalam diagram formal. Baik deskripsi *use case* maupun diagram *use case* keduanya didasarkan pada kebutuhan yang diidentifikasi dari proses bisnis. Beberapa pendapat menyatakan keduanya (deskripsi *use case* maupun diagram *use case*) dapat juga didasarkan pada deskripsi diagram aktivitas (*activity diagram*), sehingga ada kalanya *activity diagram* dapat dibuat terlebih dahulu sebelum membuat diagram *use case* (Dennis, 2012).

Deskripsi *use case* berisi semua informasi yang diperlukan untuk menghasilkan diagram *use case*. Untuk melewati langkah *deskripsi use-case* dan pindah langsung ke membuat diagram *use-case* dan diagram lain yang mengikuti, pengguna sering mengalami kesulitan menggambarkan proses bisnis mereka hanya dengan

menggunakan diagram *use case*. Melalui pembuatan deskripsi *use case*, pengguna dapat menggambarkan rincian yang diperlukan dari setiap *use case* individu. Mengenai yang mana harus dibuat pertama (deskripsi *use case* atau diagram *use case*) secara teknis, sebenarnya tidak masalah. Keduanya harus dilakukan untuk sepenuhnya menggambarkan kebutuhan yang harus dipenuhi oleh sistem informasi. Dalam pembahasan ini, akan dimulai dengan menyajikan deskripsi *use case*, selanjut membahas bagaimana menggambarkan diagram *use case*.

Saat membuat *use case*, tim proyek harus bekerja sama dengan pengguna untuk mengumpulkan kebutuhan yang diperlukan untuk *use case*. Hal ini sering dilakukan melalui wawancara, sesi JAD, dan observasi. Mengumpulkan kebutuhan yang diperlukan untuk *use case* adalah proses yang relatif sederhana, tetapi butuh banyak latihan. Tempat yang baik untuk mencari *use case* potensial adalah representasi *activity diagram* dari proses bisnis. Dalam banyak kasus, aktivitas yang diidentifikasi dalam *activity diagram* akan menjadi *use case* dalam proses bisnis yang dimodelkan. Hal utama yang perlu diingat adalah bahwa setiap *use case* dikaitkan dengan hanya satu peran yang dimiliki pengguna dalam sistem. Misalnya, seorang resepsionis di klinik dokter dapat memainkan banyak peran seperti membuat janji, menjawab telepon, mengisi catatan medis, menyambut pasien, dan sebagainya. Selain itu, ada kemungkinan bahwa beberapa pengguna akan memainkan peran yang sama. Karena itu, *use case* harus dikaitkan dengan peran yang dimainkan oleh pengguna dan bukan dengan pengguna itu sendiri.

#### A. DESKRIPSI USE CASE

*Use case* kadang tidak cukup dinyatakan dengan diagram, melainkan perlu penjelasan lebih lanjut, yang sering disebut dengan deskripsi *use case* (*use case description*). Deskripsi *use case* berisi semua informasi yang diperlukan untuk membangun diagram-diagram lain yang mengikutinya. Deskripsi *use case* sering kali mengungkapkan informasi dalam cara yang kurang formal agar lebih mudah dimengerti bagi pengguna.

Gambar 5.10 menunjukkan contoh deskripsi *use case*. Secara singkat deskripsi *use case* terdiri atas dua bagian dasar yaitu: bagian **informasi umum**, berisi informasi mengenai nama *use case*, *ID use case*, prioritas, aktor, deskripsi singkat, kondisi awal, pemicu, kesimpulan, dan kondisi akhir; bagian **aliran peristiwa**, berisi urutan tindakan/skenario utama yang dilakukan oleh aktor dan skenario alternatif. Bentuk deskripsi *use case* lainnya (Dennis, 2012) mencantumkan informasi yang lebih detail, misalnya terdapat informasi mengenai *ID use case*, tipe *use case*, jenis relasi (*relationships*), dan beberapa informasi tambahan lainnya.

**Informasi umum** mengidentifikasi *use case* dan memberikan informasi latar belakang tentang *use case*.

1. *Nama use case*, berupa frasa kata kerja (misalnya: Membuat Janji Pertemuan).
2. *ID use case*, memberikan identitas unik untuk menemukan setiap *use case*.

3. *Prioritas*, tingkat prioritas pengimplementasian use case dalam sistem
4. *Aktor*, biasanya merupakan pemicu *use case*, yaitu orang atau benda yang memulai eksekusi *use case*.
5. *Deskripsi*, merupakan penjelasan singkat mengenai pekerjaan yang dilakukan *use case*, *fungsi*, dan hasil yang didapat dari pelaksanaan *use case*
6. *Kondisi awal*, kondisi sebelum pelaksanaan *use case*
7. *Conclusion* (kesimpulan), indikator selesainya pelaksanaan *use case*
8. *Kondisi akhir*, kondisi setelah pelaksanaan *use case*

**Aliran/alur peristiwa** merupakan langkah-langkah dalam proses bisnis, berupa urutan tindakan yang dilakukan oleh aktor dan urutan respon yang diberikan oleh sistem atas tindakan aktor. Ada dua kategori alur peristiwa yang dapat didokumentasikan: alur peristiwa normal dan alur alternatif atau keadaan luar biasa.

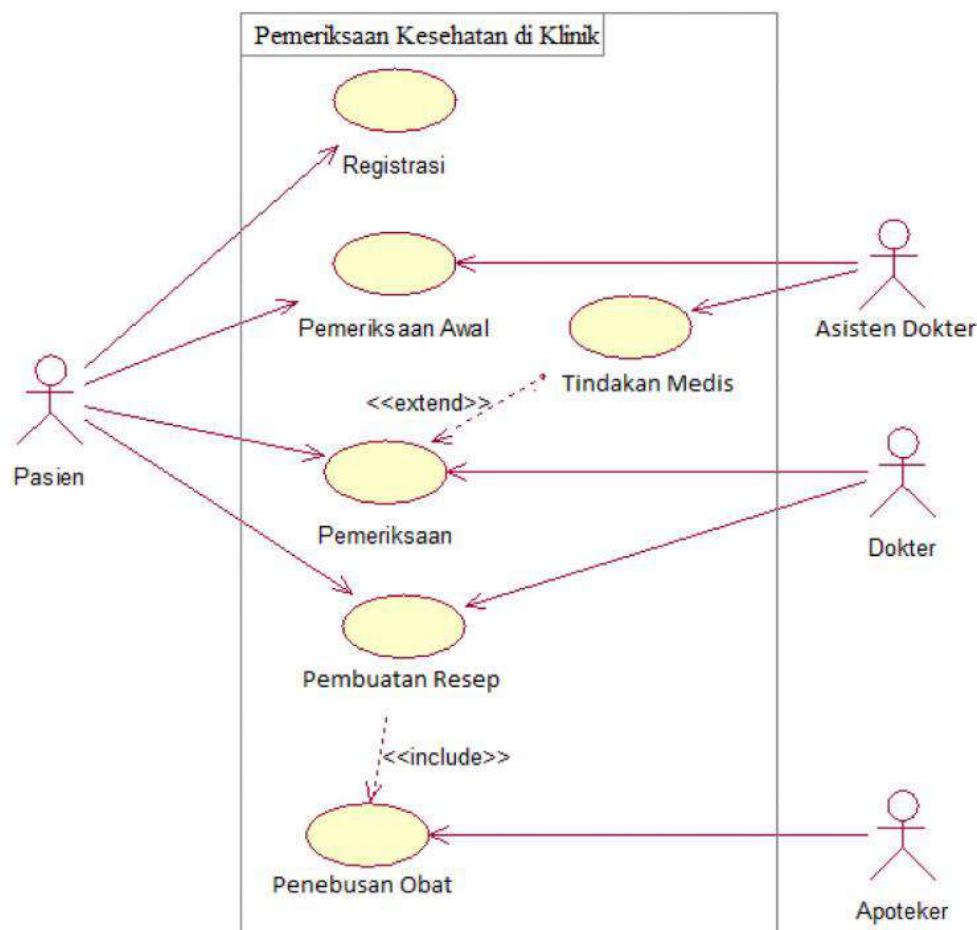
1. Alur peristiwa yang normal, hanya mencakup langkah-langkah yang umum dieksekusi dalam *use case*. Arus peristiwa yang normal harus diuraikan menjadi seperangkat aliran untuk menjaga aliran peristiwa sesederhana mungkin
2. Alur alternatif, didasarkan pada logika aliran kontrol dalam representasi *activity diagram* dari proses bisnis

<b>Nama use case</b>	<i>Login</i>	
<b>Prioritas</b>	Tinggi	
<b>Aktor</b>	Petugas Pengadaan, Petugas Lapangan, Kepala Perpustakaan	
<b>Deskripsi</b>	<i>Use-case</i> ini memungkinkan aktor untuk masuk ke sistem dengan kewenangan tertentu yang lebih dari kewenangan aktor <i>user umum</i> .	
<b>Kondisi Awal</b>	Pengguna belum <i>login</i> ke sistem.	
<b>Trigger</b>	<i>Use-case</i> ini akan berfungsi bila pengguna memberi perintah kepada sistem untuk menampilkan <i>form login</i> dan mengisinya.	
<b>Alur Peristiwa Normal</b>	<b>Aksi Aktor</b> <ol style="list-style-type: none"> <li>1. Pengguna memberi perintah kepada sistem untuk menampilkan <i>form login</i></li> <li>3. Pengguna mengisi <i>form login</i> dan men-submit-nya</li> </ol>	<b>Respon Sistem</b> <ol style="list-style-type: none"> <li>2. Sistem menampilkan <i>form login</i></li> <li>4. Sistem memproses data <i>login</i> yang dimasukkan oleh pengguna</li> <li>5. Sistem menampilkan konfirmasi hasil proses <i>login</i></li> <li>6. Sistem menampilkan tampilan utama untuk pengguna yang bersangkutan sesuai dengan wewenangnya</li> </ol>
<b>Alur Alternatif</b>	5.a Sistem akan mengeluarkan pesan <i>error</i> bila data-data <i>login</i> salah, dan mengarahkan pengguna untuk mengulangi proses tersebut.	
<b>Conclusion</b>	<i>Use case</i> ini selesai bila konfirmasi proses <i>login</i> dan tampilan utama telah ditampilkan.	
<b>Kondisi AKhir</b>	Pengguna sudah <i>login</i> ke dalam sistem.	

Gambar 5.10  
Contoh Deskripsi Use Case

## B. DIAGRAM USE CASE

Diagram *use case* menggambarkan dengan sederhana fungsi-fungsi utama sistem dan berbagai jenis pengguna yang akan berinteraksi dengannya. Sebagai contoh, Gambar 5.11 menyajikan diagram *use case* (*use case bisnis*) untuk sistem pemeriksaan kesehatan di klinik. Diagram *use case* tersebut memperlihatkan sebuah bisnis (pemeriksaan kesehatan) menanggapi beberapa kegiatan/fungsi dalam sistem pemeriksaan kesehatan. *Pasien* berinteraksi dengan sistem pada kegiatan *registrasi*; *pasien* dan *asisten dokter* berinteraksi dengan sistem pada kegiatan *pemeriksaan awal*; *pasien*, *dokter* dan *asisten dokter* berinteraksi dengan sistem pada kegiatan *pemeriksaan* dan *tindakan medis*; *pasien*, *dokter*, dan *apoteker* berinteraksi dengan sistem pada kegiatan *pembuatan resep* dan *penebusan obat*.



Gambar 5.11  
Contoh Diagram Use Case

Diagram *use case* memiliki beberapa elemen (notasi) dasar seperti disajikan pada Gambar 5.12.

NOTASI	DEFINISI
 <b>Nama Aktor</b>	<b>Aktor:</b> - Segala sesuatu yang dapat berinteraksi dengan sistem (orang, perangkat keras, atau objek/sistem lain) dan bersifat eksternal dari sistem - Diberi label beserta perannya - Dapat dikaitkan dengan aktor lain dengan panah berongga (spesialisasi/ superclass) - Ditempatkan di luar <i>boundary</i> sistem
 <b>Nama Use Case</b>	<b>Use Case:</b> - Merupakan bagian utama dari fungsionalitas sistem - Dapat menggunakan use case lain - Ditempatkan di dalam batas sistem. - Diberi label dengan frase kata kerja deskriptif
 <b>Atau</b>	<b>Relasi Asosiasi:</b> Menautkan aktor dengan <i>use case</i> yang berinteraksi dengannya.
  <b>&lt;&lt; defend on &gt;&gt;</b>	<b>Inheritance:</b> Manautkan suatu aktor dengan aktor lain  <b>Defends on:</b> Hubungan ketergantungan antar <i>use case</i> ( <i>extend / include</i> )
 <b>Nama Sistem</b>	<b>Boundary:</b> - Termasuk <i>nama sistem</i> di dalam atau di atas. - Merupakan ruang lingkup sistem. - Bersifat Opsional

Sumber: Dennis (2012)

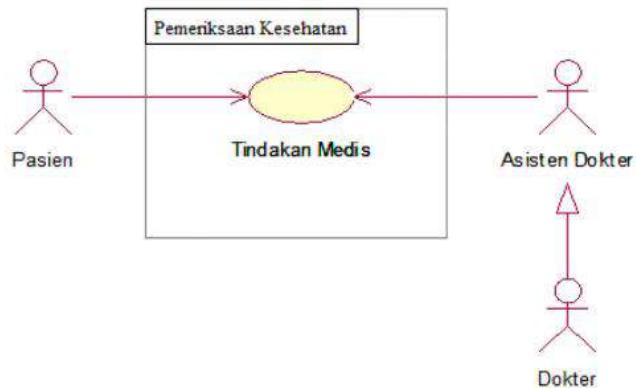
Gambar 5.12  
Elemen Diagram *Use Case* (Dennis, 2012)

## 1. Aktor

Aktor mirip dengan entitas luar/eksternal yang ditemukan dalam DFD, dilabeli dengan frasa kata benda. Seorang aktor bukan pengguna tertentu, tetapi sebuah peran yang dapat dimainkan pengguna saat berinteraksi dengan sistem. Jika kita memodelkan sistem janji temu di klinik dokter, petugas administrasi yang memasukkan informasi pasien tidak akan dianggap sebagai aktor, karena ia akan berada dalam lingkup sistem itu sendiri. Demikian juga dengan petugas administrasi apotik (bukan apoteker) yang melayani penebusan obat tidak akan dianggap sebagai aktor karena ia berada dalam lingkup sistem itu. Ini adalah aturan yang sama untuk entitas eksternal DFD (Dennis, 2012).

Terkadang aktor memainkan peran khusus di luar tipe aktor yang memainkan peran lebih umum. Misalnya, seorang *asisten dokter* dibekali beberapa pengetahuan medik yang sama dengan pengetahuan yang dimiliki oleh seorang *dokter*, sehingga ia dapat melakukan fungsi-fungsi tertentu yang melekat pada tugas seorang *dokter* (fungsi

yang diturunkan dari aktor *dokter*, kemudian diperluas dengan kewenangan utama yang dimiliki oleh *asisten dokter*), ditunjukkan oleh garis dengan segitiga berlubang (*generalization/inheritance*) di ujung *superclass* aktor yang lebih umum (dokter). Aktor khusus (asisten dokter) akan mewarisi perilaku *superclass* (dokter) dan memperluasnya dengan cara tertentu dalam menjalankan fungsinya (tindakan medis). Kasus tersebut dapat digambarkan seperti pada Gambar 5.13.



Gambar 5.13  
Diagram *Use Case* dengan Aktor Khusus (Relasi *Inheritance*)

## 2. Relasi Asosiasi

*Use case* terhubung ke aktor melalui hubungan/relasi asosiasi. Ini menunjukkan dengan *use case* yang mana para aktor berinteraksi (lihat Gambar 5.11). Garis yang ditarik dari aktor ke *use case* menggambarkan relasi asosiasi. Asosiasi biasanya mewakili komunikasi dua arah antara *use case* dan *actor*. Pada Gambar 5.11, kita melihat bahwa *instance* pasien dapat mengeksekusi *use case* *pendaftaran*. Tanda "\*" biasa juga disematkan di kedua ujung asosiasi yang mewakili multiplisitas (Dennis, 2012). Pada Gambar 5.14 kita melihat bahwa *instance* pasien dapat mengeksekusi *use case* *pendaftaran* sesering yang mereka inginkan, dan ada kemungkinan *use case* *pendaftaran* dijalankan oleh banyak pasien.



Gambar 5.14  
*Use Case* Diagram dengan Simbol Multiplisitas

### 3. Use Case

Sebuah *use case* menggambarkan proses utama yang dilakukan oleh sistem dalam beberapa cara. Proses itu dilabeli dengan frasa kata kerja deskriptif (mirip seperti proses DFD). Kita dapat mengatakan dari Gambar 5.11 bahwa sistem memiliki empat *use case* utama: registrasi, pemeriksaan awal, pemeriksaan detail, dan pembuatan resep.

Ada kalanya satu *use case* akan memperluas (*extend*) fungsionalitasnya dengan memasukkan perilaku tambahan dari *use case* lain, dan di antara keduanya dapat berjalan secara bersamaan. Pada Gambar 5.11, *use case pemeriksaan* (*use case pokok*) memperluas fungsionalitasnya dengan memasukkan perilaku tambahan dari *use case tindakan medis*. Pelaksanaan *use case tindakan medis* tergantung kepada pelaksanaan *use case pemeriksaan* (tidak dapat berdiri sendiri), dengan kata lain *use case tindakan medis* baru bisa dilakukan apabila *use case pemeriksaan* terpenuhi/dijalankan. Adapun *use case pemeriksaan* dapat berdiri sendiri.

Kondisi lainnya adalah sebuah *use case* menyertakan fungsionalitasnya (*include*) ke dalam fungsionalitas *use case* lainnya. Pada Gambar 5.11, *use case pembuatan resep* menyertakan fungsionalitasnya ke dalam fungsionalitas *use case menebus obat*. Pelaksanaan *use case penebusan obat* tergantung kepada pelaksanaan *use case pembuatan resep*. Dengan kata lain *use case penebusan obat* baru bisa dilakukan setelah *use case pembuatan resep* terpenuhi/selesai dijalankan. Kedua *use case* ini (*pembuatan resep* dan *penebusan obat*) adalah satu kesatuan kegiatan yang tidak dapat dipisahkan.

Ketergantungan *extend* berarti bahwa suatu bagian dari elemen (di garis tanpa panah) bisa disisipkan ke dalam elemen lain (di garis yang ada panah)/bisa bersamaan (paralel) prosesnya. Suatu *tindakan medis* tertentu dapat dilaksanakan berbarengan dengan kegiatan *pemeriksaan* pasien, namun tetap harus dimulai dengan pemeriksaan pasien.

Ketergantungan *include* berarti suatu bagian dari elemen (yang ada di garis tanpa panah) memicu eksekusi bagian dari elemen lain (yang ada di garis dengan panah). Pada kasus Gambar 5.11, suatu kegiatan *penebusan obat* tidak dapat dilakukan hingga menunggu proses *pembuatan resep* selesai dilaksanakan.

### 4. System Boundary

*Use case* tertuang dalam batas (*boundary*) sistem, yang merupakan kotak yang mewakili sistem dan dengan jelas menggambarkan bagian diagram mana yang eksternal atau internal. Nama sistem dapat muncul di dalam atau di atas kotak. Beberapa pandangan menyatakan bahwa penggambaran *system boundary* bersifat opsional, terutama pada penggambaran diagram *use case* yang kompleks.

### C. MEMBUAT DIAGRAM USE CASE

Perlu diketahui bahwa diagram *use case* mengomunikasikan informasi yang mirip dengan informasi yang ditemukan dalam diagram konteks dan diagram level 0 pada DFD.

#### 1. Mengidentifikasi *Use Case* dan membuat Deskripsi *Use Case*

Sebelum diagram *use case* dapat dibuat, akan sangat membantu jika terlebih dahulu melalui proses mengidentifikasi *use case* yang sesuai dengan fungsi utama sistem dan mengumpulkan dokumentasi/deskripsi *use case* untuk masing-masing *case*.

##### *Contoh Kasus*

Kita akan mengidentifikasi *use case* pada contoh kasus "Pengembangan Sistem Informasi Perpustakaan" (Modul 4, Kegiatan Belajar 1, bagian C). Misalkan berdasarkan Proses Bisnis yang diperoleh pada kegiatan elisitasi kebutuhan, diketahui fungsi-fungsi utama yang terdapat dalam aplikasi "Sistem Informasi Perpustakaan" berbasis Desktop dengan sistem jaringan komputer lokal (*Local Area Network/LAN*) adalah sebagai berikut.

- a. Para pengguna sistem (petugas pengadaan, petugas layanan, anggota, dan kepala perpustakaan) melakukan aktivitas *login* sebelum berinteraksi dengan sistem.
- b. Petugas pengadaan koleksi melakukan registrasi/menginput data koleksi pustaka ke dalam sistem aplikasi.
- c. Petugas layanan pustaka melakukan registrasi/menginput data calon anggota ke dalam sistem aplikasi.
- d. Anggota menampilkan data Koleksi Pustaka.
- e. Petugas layanan pustaka memproses/menginput data peminjaman pustaka oleh anggota ke dalam sistem aplikasi.
- f. Petugas layanan pustaka memproses/menginput data pengembalian pustaka oleh anggota ke dalam sistem aplikasi.
- g. Petugas Layanan dan atau Kepala Perpustakaan melihat/mencetak laporan peminjaman koleksi pustaka.

Pada contoh di atas, terdapat tujuh *use case* yang diidentifikasi. Pada kasus lain mungkin terdapat lebih dari tujuh atau kurang dari enam *use case*, tergantung pada hasil kesepahaman dalam proses elisitasi kebutuhan. Misalnya, dengan menambahkan beberapa *use case* untuk proses mencetak laporan. Ketujuh item *use case* tersebut adalah berikut ini.

- a. Login.
- b. Menambah data koleksi.
- c. Menambah data anggota.
- d. Menampilkan data koleksi.

- e. Menambah data peminjaman.
- f. Menambah data pengembalian.
- g. Mencetak laporan peminjaman.

Selanjutnya membuat deskripsi *use case* untuk setiap fungsionalitas sistem tersebut (seperti contoh Gambar 5.10), sehingga terdapat tujuh buah deskripsi *use case* yang harus dibuat (dimulai dari deskripsi *use case login* hingga deskripsi *use case mencetak laporan peminjaman*).

*Catatan 1:*

- a. Struktur *use case* di atas mungkin akan berbeda jika Sistem Informasi Perpustakaan yang dikembangkan berbasis Web. Misalnya, pada sistem berbasis Web, calon anggota dimungkinkan untuk melakukan registrasi secara mandiri.
  - b. Jika istilah *menambah data* dimaknai sebagai *memelihara data* (merekam, mengubah, menghapus), maka pada setiap *use case* menambah data, hanya perlu membuat satu deskripsi *use case* dengan menggabungkan proses *merekam, mengubah* dan *menghapus* data pada deskripsi *use case* (misalnya: deskripsi *use case* "menambah data koleksi" akan menggabungkan prosedur proses *merekam data koleksi, mengubah data koleksi, dan menghapus data koleksi* menjadi satu proses pada item *alur peristiwa normal* maupun *alur peristiwa alternatif*).
  - c. Jika istilah *menambah data* diartikan hanya sebagai *merekam/menyimpan data*, maka perlu dipertimbangkan untuk menambahkan *use case* lain secara tersendiri pada setiap proses memelihara data, yaitu *use case mengubah data* dan *use case menghapus data*. Dengan demikian, perlu membuat tiga buah *use case* (deskripsi *use case*) pada setiap aktivitas pemeliharaan data. Misalnya: pada aktivitas "pemeliharaan data koleksi" akan tercipta tiga buah *use case* dan deskripsi *use case*, yaitu *menambah/merekam data koleksi, mengubah data koleksi, dan menghapus data koleksi*. Demikian juga pada aktivitas *pemeliharaan data anggota* akan tercipta tiga buah *use case* dan deskripsi *use case*, yaitu *menambah/merekam data anggota, mengubah data anggota, dan menghapus data anggota*. Adapun aktivitas peminjaman dan pengembalian, hanya terdapat kegiatan *menambah/merekam data* (tidak diperbolehkan melakukan perubahan atau penghapusan data histori/transaksi). Demikian seterusnya, sehingga jumlah item *use case* dan deskripsi *use case* menjadi sebagai berikut.
- 1) Login.
  - 2) Menambah data koleksi.
  - 3) Mengubah data koleksi.
  - 4) Menghapus data koleksi.
  - 5) Menambah data anggota.
  - 6) Mengubah data anggota.
  - 7) Menghapus data anggota.

- 8) Menampilkan data koleksi.
- 9) Menambah data peminjaman.
- 10) Menambah data pengembalian.
- 11) Mencetak laporan peminjaman.

## 2. Menggambarkan Batas Sistem

Pertama, meletakkan kotak pada diagram *use case* untuk mewakili sistem, dan selanjutnya meletakkan nama sistem di dalam atau di atas kotak. Ini akan membentuk batas (*boundary*) sistem untuk memisahkan *use case* (fungsionalitas sistem) dari *actor* (pengguna eksternal). Pada beberapa diagram *use case*, *boundary system* tidak disertakan.

## 3. Menempatkan *Use Case* pada Diagram

Langkah selanjutnya adalah menambahkan *use case* ke dalam batas sistem (*boundary*), seperti pada Gambar 5.15.

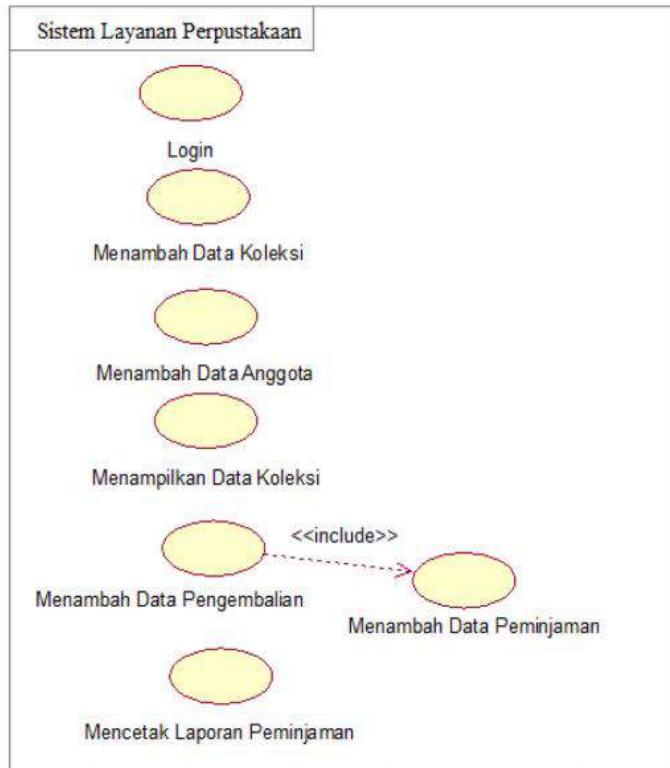


Gambar 5.15  
Menempatkan *Use Case* dalam Diagram

Sedapat mungkin tidak terdapat lebih dari delapan *use case* pada sebuah model. Jika kita mengidentifikasi lebih dari delapan *use case*, kita perlu mengelompokkan beberapa *use case* ke dalam paket (kelompok logis *use case*) untuk membuat diagram lebih mudah dibaca dan mempertahankan model pada tingkat kompleksitas yang wajar (Dennis, 2012). Pada situasi ini, hubungan *use case khusus* (*include* atau *extend*) harus ditambahkan ke model. Ini diimplementasikan dengan mencari *use case* yang mungkin mencakup fungsionalitas umum yang dibutuhkan *use case* lain (yaitu: *include/mencakup* hubungan fungsional) atau *use case* yang menambahkan fungsionalitas ke *use case* yang lain (yaitu: *extend/perluasan* hubungan fungsional).

#### Contoh Kasus

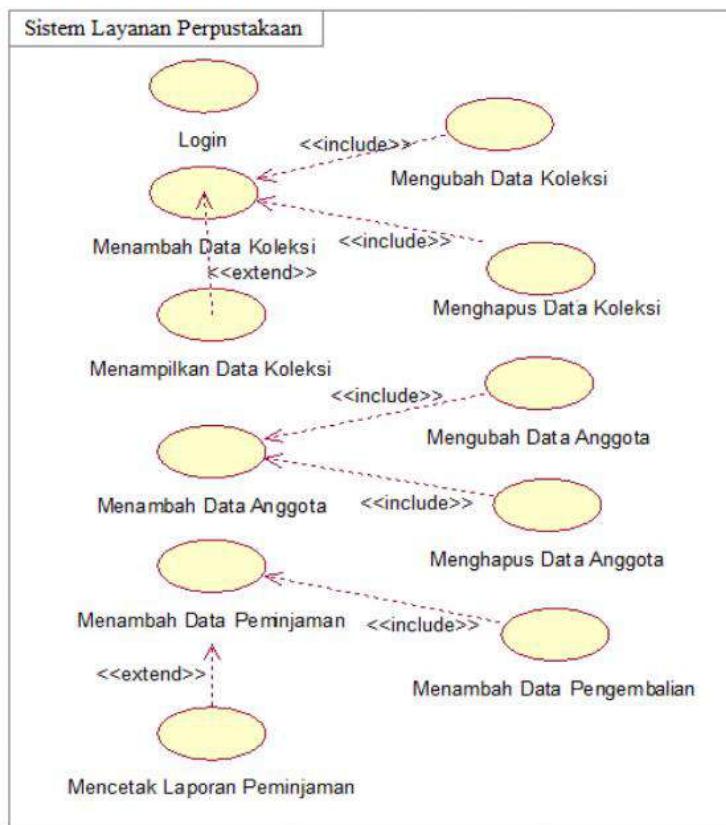
Tujuh item *use case* pada gambar 5.15 dapat disederhanakan menjadi seperti pada gambar 5.16, sedangkan 11 item *use case* pada *Catatan 1-c* dapat disederhanakan menjadi seperti pada Gambar 5.17.



Gambar 5.16  
Penyederhanaan Diagram *Use Case* dengan Sistem *Include*

Gambar 5.16 adalah contoh penyederhanaan tujuh item *use case* menjadi hanya enam *use case/deskripsi use case*, yaitu dengan cara memaketkan *use case* "menambah

data peminjaman” dan *use case* ”menambah data pengembalian” (*use case* ”menambah data pengembalian” termasuk/include ke dalam *use case* ”menambah data peminjaman”). Pada saat membuat deskripsi *use case* ”menambah data peminjaman”, prosedur proses ”menambah data pengembalian” termasuk di dalamnya (menyatu pada *alur peristiwa normal* maupun pada *alur peristiwa alternatif*).



Gambar 5.17  
Penyederhanaan Diagram *Use Case* dengan Sistem *Include* dan *Extend*

Gambar 5.17 adalah contoh penyederhanaan 11 item *use case* (pada Catatan 1-c) menjadi hanya empat *use case*/deskripsi *use case*, yaitu dengan cara:

- memaketkan *use case* ”menambah data koleksi”, *use case* ”menghapus data koleksi”, *use case* ”mengubah data koleksi”, dan *use case* ”menampilkan data koleksi” menjadi satu paket, yaitu:
  - memperluas/extend fungsionalitas *use case* ”menambah data koleksi” dengan memasukkan fungsi *use case* ”menampilkan data koleksi”
  - menyertakan/include fungsi *use case* ”menghapus data koleksi” dan *use case* ”mengubah data koleksi” ke *use case* ”menambah data koleksi”

Pada saat membuat *deskripsi use case* "menambah data koleksi", prosedur proses "menghapus data koleksi", "mengubah data koleksi" dan "menampilkan data koleksi" termasuk di dalamnya.

- b. memaketkan *use case* "menambah data anggota", *use case* "menghapus data anggota", dan *use case* "mengubah data anggota" menjadi satu paket (menyertakan/include *use case* "menghapus data anggota" dan *use case* "mengubah data anggota" ke *use case* "menambah data anggota"). Pada saat membuat *deskripsi use case* "menambah data anggota", prosedur proses "menghapus data anggota" dan "mengubah data anggota" termasuk di dalamnya.
- c. memaketkan *use case* "menambah data peminjaman", *use case* "menambah data pengembalian", dan *use case* "mencetak laporan peminjaman" menjadi satu paket, yaitu:
  - 1) memperluas/extend fungsionalitas *use case* "menambah data peminjaman" dengan memasukkan fungsi *use case* "mencetak laporan peminjaman"
  - 2) menyertakan/include fungsi *use case* "menambah data pengembalian" ke *use case* "menambah data peminjaman".

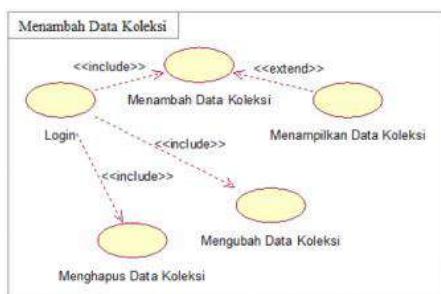
Pada saat membuat *deskripsi use case* "menambah data peminjaman", prosedur proses "menambah data pengembalian" dan "mencetak laporan peminjaman" termasuk di dalamnya.

Cara lain menyembunyikan kompleksitas diagram *use case* adalah memecah sebuah diagram *use case* menjadi beberapa diagram.

#### *Contoh Kasus*

Diagram *use case* Gambar 5.18 dapat dipecah menjadi beberapa diagram *use case* (pada kasus ini misalkan dipecah menjadi tiga diagram *use case*, masing-masing terdiri dari: diagram *use case* "menambah data koleksi", diagram *use case* "menambah data anggota", dan diagram *use case* "memproses data peminjaman", dengan asumsi *use case login* menjadi *include* ke *use case* lainnya):

#### 1) Diagram *use case* "menambah data koleksi"



Gambar 5.18  
Diagram *Use Case* Menambah Data Koleksi

Diagram *use case* "menambah data koleksi" (Gambar 5.18) terdiri atas:

- a) *use case* "menambah data koleksi" (*include* dengan *use case* "login" dan *extend* dengan *use case* "menampilkan data koleksi"). Pada saat membuat deskripsi *use case* "menambah data koleksi", prosedur proses "menampilkan data koleksi" dan prosedur proses "login" termasuk di dalamnya.
  - b) *use case* "mengubah data koleksi" (*include* dengan *use case* "login"). Pada saat membuat deskripsi *use case* "mengubah data koleksi", prosedur proses "login" termasuk di dalamnya.
  - c) *use case* "menghapus data koleksi" (*include* dengan *use case* login). Pada saat membuat deskripsi *use case* "menghapus data koleksi", prosedur proses "login" termasuk di dalamnya.
- 2) Diagram *use case* "menambah data anggota", terdiri atas:
- a) *use case* "menambah data anggota" (*include* dengan *use case* "login"). Pada saat membuat deskripsi *use case* "menambah data anggota", prosedur proses "login" termasuk di dalamnya.
  - b) *use case* "mengubah data anggota" (*include* dengan *use case* "login"). Pada saat membuat deskripsi *use case* "mengubah data anggota", prosedur proses "login" termasuk di dalamnya.
  - c) *use case* "menghapus data anggota" (*include* dengan *use case* "login"). Pada saat membuat deskripsi *use case* "menghapus data anggota", prosedur proses "login" termasuk di dalamnya.
- 3) Diagram *use case* "memproses data peminjaman", terdiri atas:  
*use case* "menambah data peminjaman" (*include* dengan *use case* "login", *include* dengan *use case* "menambah data pengembalian", dan *extend* dengan *use case* "mencetak laporan peminjaman"). Pada saat membuat deskripsi *use case* "menambah data peminjaman", prosedur proses "login", prosedur proses "menambah data pengembalian", serta prosedur proses "mencetak laporan peminjaman" termasuk di dalamnya.

*Catatan 2:*

*Use case* "login" dapat didesain untuk diakses secara mandiri (seperti pada Gambar 5.16), atau didesain untuk dapat diakses secara *include* ke *use case* "menambah data koleksi", "mengubah data koleksi", dan "menghapus data koleksi" (seperti pada Gambar 5.17). Akses secara *include* berarti setiap kali user menjalankan sebuah fungsi (menambah data koleksi/mengubah data koleksi/menghapus data koleksi), harus didahului dengan menjalankan fungsi "login". Akses secara mandiri berarti *user* cukup satu kali menjalankan fungsi "login", selanjutnya dapat mengakses fungsi-fungsi lainnya secara bebas/bergantian tanpa harus kembali menjalankan fungsi "login"

(kecuali jika *user* menjalankan fungsi "logout", maka *user* harus kembali menjalankan fungsi "login").

#### 4. Mengidentifikasi Aktor

Setelah *use case* ditempatkan pada diagram, kita perlu mengidentifikasi aktor. Disarankan untuk melihat kembali sumber dan tujuan (*input* dan *output* utama) sebuah proses yang diidentifikasi pada proses bisnis sistem atau pada *deskripsi use case*.

Seperti diketahui bahwa aktor dapat disamakan dengan entitas luar pada DFD, yaitu orang atau sistem lain yang terkait langsung dengan sistem yang dimodelkan, namun berada di luar sistem yang sedang dimodelkan tersebut. Ini memiliki dua pengertian, yaitu di luar *sistem bisnis* yang sedang dimodelkan atau di luar *sistem aplikasi* yang sedang dimodelkan.

Aktor di luar *sistem bisnis* berarti di luar dari suatu organisasi atau di luar dari suatu bagian organisasi yang dimodelkan.

*Contoh:*

Jika yang sedang dimodelkan adalah *sistem bisnis akademik* sebuah perguruan tinggi, maka seluruh petugas/pegawai yang terlibat secara langsung di dalam pengelolaan sistem akademik menjadi bagian internal sistem (tidak dapat dianggap sebagai aktor/entitas luar sistem). Adapun mahasiswa, dosen, ketua program studi, orang tua/wali siswa, bagian keuangan, bagian penerimaan siswa baru, kasir, petugas administrasi penerimaan siswa baru adalah termasuk aktor/entitas luar.

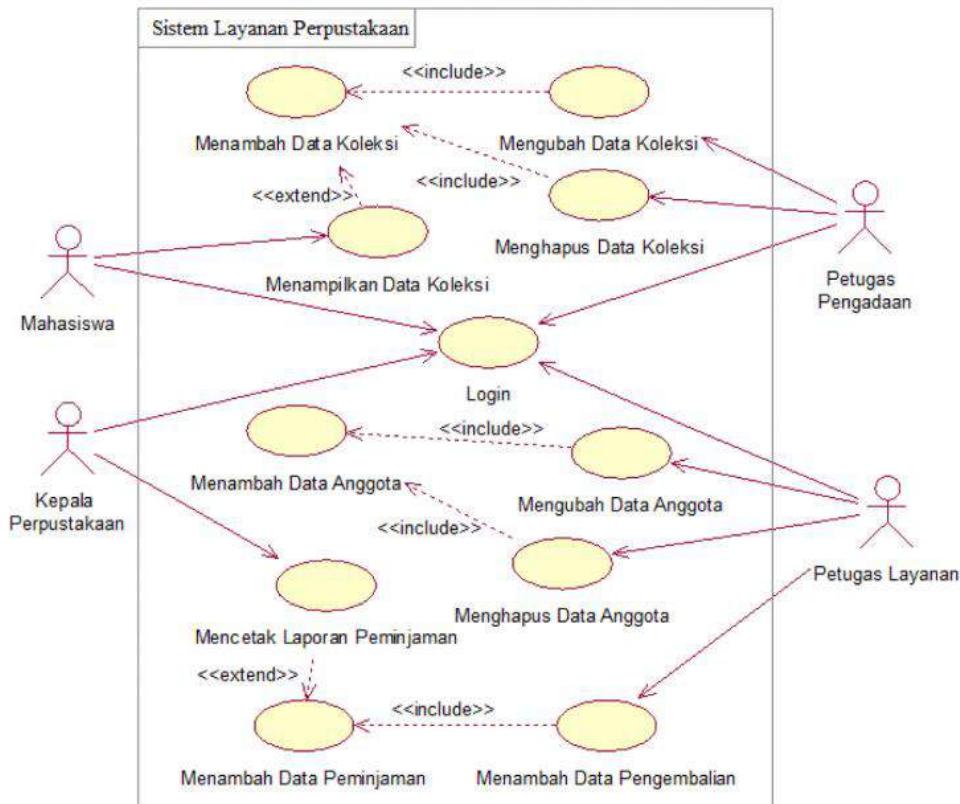
Aktor di luar *sistem aplikasi* berarti siapapun pengguna yang berinteraksi langsung dengan sistem aplikasi, atau sistem lain apapun yang terkoneksi dengan sistem aplikasi yang sedang dimodelkan.

*Contoh:*

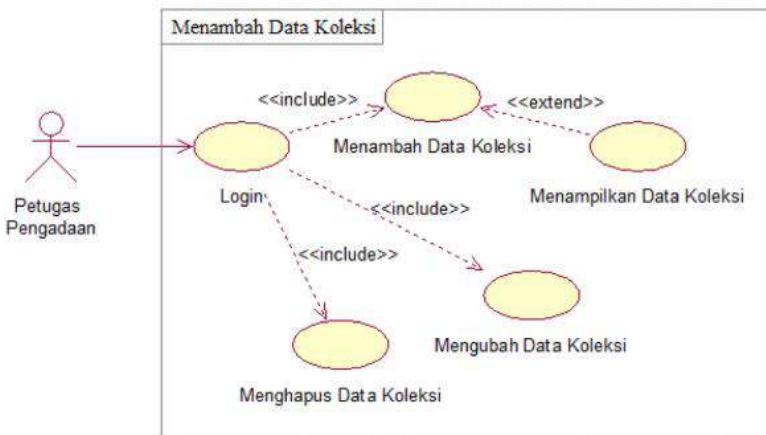
Jika yang dimodelkan adalah *sistem aplikasi akademik* sebuah perguruan tinggi, maka seluruh pengguna yang memiliki hak akses untuk berinteraksi langsung terhadap aplikasi akademik (baik petugas yang berada di dalam lingkungan akademik/pengelola akademik maupun pengguna yang berada di luar sistem akademik) dapat dianggap sebagai aktor/entitas luar. Aplikasi sistem penerimaan mahasiswa baru atau aplikasi sistem keuangan yang terintegrasi dengan aplikasi akademik (sebagai sumber *input* atau sebagai tujuan *output* sistem aplikasi akademik) juga dapat dianggap sebagai aktor. Mahasiswa mungkin tidak dapat dianggap sebagai aktor pada sistem aplikasi akademik yang berbasis desktop, sebab mereka tidak punya hak akses langsung ke sistem aplikasi yang berbasis desktop tersebut. Namun demikian, mahasiswa dapat dianggap sebagai aktor jika memiliki hak akses secara langsung terhadap sistem aplikasi akademik (misalnya: memiliki hak akses untuk registrasi ulang pada setiap awal semester ke dalam sistem aplikasi akademik yang berbasis Web).

## 5. Menambahkan Hubungan Asosiasi

Langkah terakhir adalah menggambar garis yang menghubungkan aktor dengan *use case* dalam suatu interaksi (relasi asosiasi). Tidak ada aturan yang mengharuskan sebuah objek harus ditempatkan dalam urutan tertentu. Oleh karena itu, kita mungkin ingin mengatur ulang simbol-simbol untuk meminimalkan jumlah garis yang saling melintasi, sehingga diagram menjadi membingungkan. Contoh interaksi aktor dengan *use case* seperti pada Gambar 5.11. Contoh lain seperti disajikan pada Gambar 5.19 dan 5.20.



Gambar 5.19  
Hubungan Asosiasi pada Diagram Use Case "Sistem Layanan Perpustakaan"



Gambar 5.20  
Hubungan Asosiasi pada Diagram Use Case "Menambah Data Koleksi"



### Latihan

Untuk memperdalam pemahaman Anda mengenai materi di atas, kerjakanlah latihan berikut!

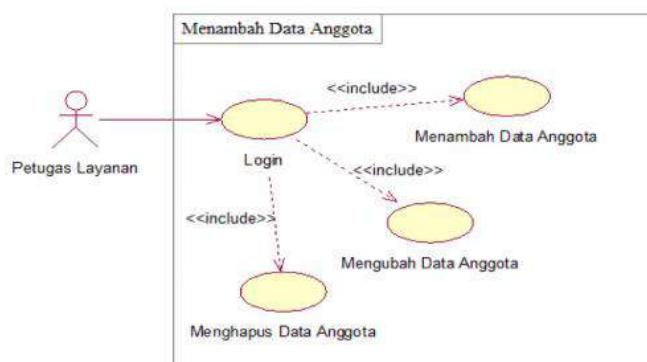
- 1) Jelaskan fungsi diagram *use case*! Bagaimana diagram *use case* serupa dengan DFD (DFD level 0)? Apa bedanya?
- 2) Jelaskan langkah-langkah dalam membuat diagram *use case*!
- 3) Jelaskan fungsi *use case description*! Bagaimana menggunakan *use case description* untuk mengembangkan diagram *use case*?
- 4) Jelaskan alasan yang mendasari perlunya dibuat relasi *assosiasi* antar *use case*!
- 5) Berdasarkan skenario pada **contoh kasus halaman 39**, gambarkan diagram *use case* untuk: "Menambah Data Anggota" (dengan asumsi: *use case* "login" *include* ke *use case* lainnya).

*Petunjuk Jawaban Latihan.*

- 1) Fungsi diagram *use case*, persamaannya dan perbedaannya dengan DFD (DFD Level 0).  
Diagram use case digunakan untuk menjelaskan dan mendokumentasikan fungsi-fungsi utama sistem, yaitu apa yang dapat dilakukan oleh pengguna dan bagaimana sistem merespon tindakan pengguna. Fungsi ini mirip seperti DFD level 0, yang juga menggambarkan proses utama yang dilakukan oleh sistem dan interaksinya dengan entitas luar (pada diagram use case disebut *actor*). Persamaan lainnya adalah fungsi-fungsi utama pada keduanya (diagram *use case* maupun DFD level 0) diidentifikasi dari proses bisnis sistem atau *use case*.

*description*. Perbedaannya, DFD digunakan dalam pemodelan terstruktur, sedangkan diagram *use case* untuk pemodelan berbasis objek.

- 2) Langkah-langkah dalam membuat diagram *use case*:
  - a) mengidentifikasi *use case*, merujuk pada proses bisnis sistem atau *activity diagram* atau *use case description*;
  - b) menggambar batas sistem (opsional);
  - c) menambahkan *use case* ke diagram, dengan memperhatikan hubungan ketergantungan antar *use case* (*extend/include*);
  - d) mengidentifikasi *actor*, dengan memperhatikan konsep *inheritance*;
  - e) menambahkan asosiasi yang sesuai untuk menghubungkan *use case* dengan *actor*.
- 3) Fungsi *use case description*: memberikan penjelasan atau informasi lebih detail tentang *use case* dalam membangun diagram-diagram lain yang mengikutinya. Sebuah *use case description* pada bagian informasi umum berisi informasi mengenai nama *use case*, ID *use case*, prioritas, *actor*, deskripsi singkat, kondisi awal, pemicu, dan kondisi akhir; sedangkan pada bagian aliran peristiwa berisi informasi urutan skenario. Beberapa informasi yang tercantum dalam *use case description*, seperti nama *use case*, deskripsi singkat, dan *actor* inilah yang menjadi komponen sebuah diagram *use case*.
- 4) Untuk mempertahankan diagram *use case* pada tingkat kompleksitas yang wajar sehingga mudah dibaca, sedapat mungkin membatasi jumlah *use case* dalam sebuah model agar tidak terlalu banyak. Penyederhanaan jumlah dapat dilakukan dengan menambahkan relasi *use case* khusus (assosiasi antar *use case*), diimplementasikan dengan mengidentifikasi *use case* yang mungkin mencakup fungsionalitas umum yang dibutuhkan *use case* lain (*include*), atau *use case* yang menambahkan fungsionalitas ke *use case* lain (*extend*).
- 5) Diagram *use case* "menambah data anggota" (dengan asumsi: *use case* "login" *include* ke *use case* lainnya).





## Rangkuman

Diagram *use case* menggambarkan fungsi-fungsi utama suatu sistem dan berbagai jenis pengguna yang berinteraksi dengannya. Diagram terdiri atas *actor*, yang merupakan orang atau sistem lain yang terkoneksi dan mendapatkan manfaat langsung dari sistem, dan menggunakan *case* yang mewakili fungsionalitas sistem. *Actor* dan *use case* dipisahkan oleh batas sistem dan dihubungkan oleh garis yang mewakili asosiasi. Kadang-kadang *actor* merupakan jenis *actor* khusus dari *actor* yang lebih umum (*inheritance*). Demikian pula, *use case* dapat memperluas fungsinya dengan memasukkan perilaku *use case* lainnya (*extend*), atau menyertakan fungsinya pada *use case* lainnya (*include*).

Membangun diagram *use case* terdiri dari lima langkah utama yaitu: analis mengidentifikasi *use case* dari proses bisnis sistem atau *activity diagram*, menggambar batas sistem, menambahkan *use case* ke diagram, mengidentifikasi *actor*, dan akhirnya, menambahkan asosiasi yang sesuai untuk menghubungkan *use case* dengan *actor*.

Setelah dibuat, *use case* sering dapat digunakan untuk mendapatkan kebutuhan fungsional yang lebih rinci untuk sistem baru, sehingga memerlukan penjelasan lebih lanjut, yang sering disebut dengan *use case description*. *Use case description* berisi semua informasi yang diperlukan untuk membangun diagram-diagram lain yang mengikutinya. Mengenai yang mana harus dibuat pertama (deskripsi *use case* atau diagram *use case*) secara teknis, sebenarnya tidak masalah. Keduanya harus dilakukan untuk menggambarkan kebutuhan yang harus dipenuhi oleh sistem informasi.



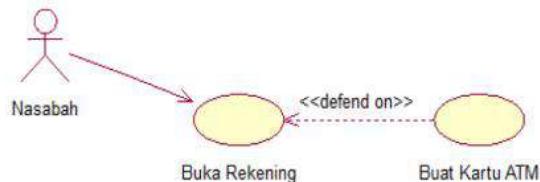
## Tes Formatif 2

Pilihlah satu jawaban yang paling tepat!

- 1) Salah satu komponen notasi dasar yang terdapat di dalam diagram *use case* adalah ....
  - A. *operation name*
  - B. *attribute name*
  - C. *lifeline*
  - D. *association*
  
- 2) Komponen notasi *actor* pada diagram *use case* biasanya menggunakan kata ....
  - A. benda
  - B. kerja
  - C. sifat
  - D. perintah

#### 5.46 Pemodelan Analisis Dengan Pendekatan Berorientasi Objek

- 3) Komponen notasi *use case* pada diagram *use case* biasanya menggunakan kata ....
- A. benda
  - B. kerja deskriptif
  - C. sifat
  - D. perintah
- 4) Berikut adalah jenis relasi yang bisa muncul pada diagram *use case*, *kecuali* ....
- A. *association* antara *use case*
  - B. *association* antara *actor* dengan *use case*
  - C. *generalization/inheritance* antara *actor* dengan *use case*
  - D. *generalization/inheritance* antara *actor* dengan *actor*
- 5) Digambarkan dengan sebuah garis berpanah tertutup (berlubang) pada salah satu ujungnya, yang menunjukkan lebih umum, adalah relasi ....
- A. *association* antar *use case*
  - B. *inheritance* antar *actor*
  - C. *association* antar *actor* dan *use case*
  - D. semua jawaban A, B, dan C salah
- 6) Relasi ketergantungan antar *use case* yang ditunjukkan pada gambar berikut ini adalah ....



- A. relasi *inheritance*
- B. relasi *extend*
- C. relasi *include*
- D. relasi asosiasi

- 7) Relasi ketergantungan antar *use case* yang ditunjukkan pada gambar berikut ini adalah ....



- A. relasi *inheritance*
- B. relasi *extend*

- C. relasi *include*  
 D. relasi asosiasi
- 8) Seorang nasabah akan melakukan transaksi pada sebuah mesin ATM. Setiap kali nasabah menjalankan suatu jenis transaksi tertentu (misalnya: penarikan tunai, cek saldo, atau transaksi lainnya), sistem ATM secara berulang meminta nasabah melakukan login. Relasi antara aktivitas login dan setiap jenis aktivitas transaksi pada sistem mesin ATM tersebut adalah relasi *extend*. Pernyataan ini adalah ....  
 A. Benar  
 B. Salah
- 9) Notasi *boundary* wajib selalu disertakan pada penggambaran diagram *use case*, untuk menandai ruang lingkup sistem. Pernyataan ini adalah ....  
 A. Benar  
 B. Salah
- 10) Ada kalanya diagram *use case* dibuat setelah terlebih dahulu membuat diagram *activity*. Pernyataan ini adalah ....  
 A. Benar  
 B. Salah

Cocokkanlah jawaban Anda dengan Kunci Jawaban Tes Formatif 2 yang terdapat di bagian akhir modul ini. Hitunglah jawaban yang benar. Kemudian, gunakan rumus berikut untuk mengetahui tingkat penguasaan Anda terhadap materi Kegiatan Belajar 2.

$$\text{Tingkat Penguasaan} = \frac{\text{Jumlah Jawaban yang Benar}}{\text{Jumlah Soal}} \times 100$$

Arti tingkat penguasaan

<70%	70% - 79%	80% - 89%	90% - 100%
kurang	cukup	baik	baik sekali

Apabila mencapai tingkat penguasaan 80% atau lebih, Anda dapat meneruskan dengan modul selanjutnya. **Bagus!** Jika masih di bawah 80%, Anda harus mengulangi materi Kegiatan Belajar 2, terutama bagian yang belum dikuasai.

## Kunci Jawaban Tes Formatif

### *Tes Formatif 1*

- 1) C
- 2) B
- 3) A
- 4) B
- 5) C
- 6) D
- 7) D
- 8) D
- 9) B
- 10) B

### *Tes Formatif 2*

- 1) D
- 2) A
- 3) B
- 4) C
- 5) B
- 6) B
- 7) C
- 8) B
- 9) B
- 10) A

## Glosarium

Abstract class	: Class yang tidak memiliki/menghasilkan instance (object)
A-Kind-Of	: Hubungan antara kelas dan superclassnya
Architecture Centric	: Arsitektur yang mendasari sistem yang dikembangkan
Attribut	: Variabel global yang dimiliki oleh sebuah kelas yang tidak melakukan operasi, tetapi kepadanya dilakukan operasi
Behavior	: Hal-hal yang bisa dilakukan oleh object dari suatu class
Class	: Template yang digunakan untuk mendefinisikan dan membuat instance
Concrete class	: Class yang memiliki/menghasilkan instance (object)
Data-Centric	: Dekomposisi masalah berbasis data
Dynamic binding	: Teknik yang menunda mengidentifikasi suatu jenis objek hingga waktu berjalan (run-time)
Encapsulation	: Suatu mekanisme untuk menyembunyikan atau memproteksi suatu proses dari kemungkinan interferensi atau penyalahgunaan dari luar sistem dan sekaligus menyederhanakan penggunaan sistem tersebut
Forward engineering	: Suatu konsep di mana UML dapat menghasilkan kode program
Incremental	: Proses pengembangan sistem berorientasi objek yang menekankan pada pengujian sistem secara terus menerus sepanjang umur proyek
Instance	: Dapat disamakan maknanya dengan object
Iterative	: Pengembangan berorientasi objek yang menekankan pada proses (pengembangan) yang berulang

## 5.50 Pemodelan Analisis Dengan Pendekatan Berorientasi Objek

---

Messages	: Informasi yang dikirim ke objek untuk memicu method.
Method	: Suatu tindakan yang dapat dilakukan oleh object
Object	: Gambaran dari sebuah entitas
Polymorphism	: Proses penafsiran pesan yang serupa secara berbeda
Process-Centric	: Dekomposisi masalah berbasis proses
Reusability	: Penggunaan kembali objek dalam konteks sistem berorientasi objek
Reverse engineering	: Suatu konsep di mana peranti lunak (software) secara otomatis membaca kode program (source code) untuk menghasilkan diagram UML
Static binding	: Teknik mengidentifikasi objek pada saat proses kompilasi
Sub class (child class)	: Class yang berada pada posisi bagian bawah pada diagram hierarki sistem pewarisan objek
Super class (parent class)	: Class yang berada pada posisi atas pada diagram hierarki sistem pewarisan objek
UML (Unified Modeling Language)	: Bahasa yang telah menjadi standar dalam industri untuk visualisasi, merancang, dan mendokumentasikan sistem perangkat lunak
Use Case Diagram	: Diagram yang menggambarkan fungsi-fungsi utama sistem dan berbagai jenis pengguna yang akan berinteraksi dengannya.
Use Case Driven	: Penggunaan use case sebagai alat pemodelan utama untuk mendefinisikan perilaku sistem
Use Case	: Notasi dalam diagram use case yang melambangkan fungsionalitas sistem

## Daftar Pustaka

David, M. K. (2008). *Database processing: dasar-dasar desain dan implementasi*. Jilid 1 edisi 9. Jakarta: Erlangga.

Dennis, A., Wixom, B. H., Roth, R.M. (2012). *Systems analysis & design*. 5th Edition. United States of America: John Wiley & Sons, Inc.

Fowler, M. (2005). *UML distilled*. Edisi 3. Yogyakarta: ANDI.

Nugoro, A. (2005). *Rational rose untuk pemodelan berorientasi objek*. Bandung: Informatika.

Sugiarti, Y. (2013). *Analisis dan perancangan UML*. Yogyakarta: Graha Ilmu.

**MSIM4302**  
**Edisi 1**

**MODUL 06**

# **Pemodelan Proses Bisnis dan Struktural Sistem**

Bahar, S.T., M.Kom.

## Daftar Isi

### Modul 06 6.1

Pemodelan Proses Bisnis dan Struktural Sistem

#### Kegiatan Belajar 1

Pemodelan Proses Bisnis dengan *Activity Diagram*

Latihan

Rangkuman

Tes Formatif 1

6.5

6.14

6.15

6.15

#### Kegiatan Belajar 2

Pemodelan Struktural dengan *Class Diagram* dan *Object Diagram*

Latihan

Rangkuman

Tes Formatif 2

6.18

6.31

6.33

6.34

Kunci Jawaban Tes Formatif

Glosarium

Daftar Pustaka

6.37

6.38

6.39



## Pendahuluan

Model fungsional menggambarkan proses bisnis dan interaksi sistem dengan lingkungannya. Dalam pengembangan sistem berorientasi objek, dua jenis model digunakan untuk menggambarkan fungsionalitas sistem informasi, yaitu: *use case* yang disajikan dalam bentuk *use case diagram*, serta *activity diagram* (diagram aktivitas). *Use case diagram* telah dibahas pada Modul 5, sedangkan *activity diagram* akan menjadi bagian yang dibahas pada modul ini. *Activity diagram* mendukung pemodelan logis dari proses bisnis dan alur kerja (di samping dapat digunakan untuk menggambarkan arus aktivitas dalam *use case*, dan desain terperinci dari suatu *method*). Dengan demikian, pada beberapa situasi tertentu sebelum menggambar *use case diagram*, dapat didahului dengan menggambarkan *activity diagram* untuk menyajikan proses bisnis sistem, sebagai landasan untuk mengidentifikasi fungsi-fungsi logis utama dalam bisnis, yang akan digambarkan dengan *use case diagram*. *Activity diagram* dan *use case diagram* dapat digunakan untuk menggambarkan sistem yang sedang berjalan saat ini dan sistem yang akan/sedang dikembangkan.

Sebagai langkah awal, tim proyek mengumpulkan kebutuhan pengguna (lihat Modul 3). Selanjutnya, dengan menggunakan kebutuhan yang dikumpulkan, tim proyek memodelkan proses bisnis secara keseluruhan menggunakan *activity diagram*. Berikutnya tim proyek dapat menggunakan aktivitas-aktivitas yang diidentifikasi untuk menentukan *use case* yang terjadi dalam bisnis. Mereka kemudian menyiapkan *deskripsi use case* dan *use case diagram* untuk setiap *use case*. Pengguna sistem bekerja sama dengan tim proyek membuat model proses bisnis dalam bentuk *activity diagram* dan *deskripsi use case*, berisi semua informasi yang diperlukan untuk mulai memodelkan sistem. Setelah *activity diagram* dan *deskripsi use case* disiapkan, analis sistem mengubahnya menjadi *use case diagram*, yang menggambarkan pandangan perilaku atau fungsi eksternal dari proses bisnis. Selanjutnya, analis biasanya membuat *class diagram* untuk membuat model struktural domain masalah bisnis.

Model struktural mewakili hal-hal, ide, atau konsep (yang disebut objek) yang terkandung dalam domain masalah, juga memungkinkan representasi hubungan antara objek. Dengan membuat model struktural dari domain masalah, analis menciptakan kosakata yang diperlukan untuk analis dan pengguna untuk berkomunikasi secara efektif dalam pengembangan sistem.

Dalam modul ini, pertama-tama menggambarkan pemodelan proses bisnis menggunakan *activity diagram*. Oleh karena *deskripsi use case* dan *use case diagram* telah dibahas pada Modul 5, maka setelah membahas *activity diagram*, selanjutnya akan dibahas mengenai pemodelan struktural (*class diagram* dan *object diagram*).

#### 6.4 Pemodelan Proses Bisnis dan Struktural Sistem

---

Setelah mempelajari modul ini mahasiswa diharapkan secara lebih rinci mampu:

1. menjelaskan tentang model fungsional, kegunaannya, serta memberikan contoh model fungsional dalam pengembangan Sistem Berorientasi Objek;
2. menjelaskan kegunaan *Activity Diagram*;
3. menjelaskan elemen-elemen dalam *Activity Diagram* beserta kegunaannya;
4. menggambar *Activity Diagram*;
5. menjelaskan tentang model struktural, kegunaannya, serta memberikan contoh model struktural dalam pengembangan Sistem Berorientasi Objek;
6. menjelaskan kegunaan *Class Diagram*;
7. menjelaskan elemen-elemen dalam *Class Diagram* beserta kegunaannya;
8. menjelaskan tentang konsep penyederhanaan *Class Diagram*;
9. menggambar *Class Diagram*;
10. menjelaskan tentang *Object Diagram* dan kegunaannya;
11. menggambar *Object Diagram*.

Kegiatan  
Belajar

1

## Pemodelan Proses Bisnis dengan *Activity Diagram*

Pada modul-modul sebelumnya telah dibahas teknik analisis kebutuhan yang populer, seperti wawancara, JAD, dan observasi. Dengan menggunakan teknik-teknik ini, analis mendefinisikan kebutuhan pengguna, termasuk kebutuhan proses bisnis sistem. Mendefinisikan kebutuhan berarti mendefinisikan apa yang harus dilakukan sistem. Dalam kegiatan belajar ini, dibahas bagaimana informasi mengenai proses bisnis sistem yang dikumpulkan menggunakan teknik-teknik ini disusun dan digambarkan dalam bentuk *activity diagram*. Karena UML telah diterima sebagai notasi standar oleh *Object Management Group* (Nugroho, 2005), hampir semua proyek pengembangan berorientasi objek saat ini menggunakan *activity diagram* dan *use case* untuk mendokumentasikan proses bisnis dan kebutuhan fungsional yang diperoleh semasa fase analisis.

*Activity diagram* pada dasarnya dapat digunakan untuk semua jenis aktivitas pemodelan proses, seperti: logika prosedural, proses bisnis, dan jalur kerja (Fowler, 2005). Dalam beberapa hal, diagram ini memainkan peran mirip sebuah diagram alir (*flowchart*), tetapi perbedaan prinsip antara diagram ini dan notasi diagram alir adalah *activity diagram* mendukung behavior paralel.

*Activity diagram* telah mengalami beberapa perubahan selama perkembangan versi-versi UML. Dalam UML 1, *activity diagram* dianggap sebagai kasus khusus *state diagram*. Hal ini menyebabkan banyak masalah bagi pengguna yang memodelkan jalur kerja, yang mana cocok dikerjakan oleh *activity diagram*. Dalam UML 2, ikatan tersebut dihilangkan.

Sebagai model logis, *activity diagram* menggambarkan aktivitas domain bisnis tanpa menyarankan bagaimana hal itu dilakukan. Model logis kadang-kadang disebut sebagai model domain permasalahan. Membaca *activity diagram* pada prinsipnya, tidak boleh menunjukkan apakah suatu aktivitas terkomputerisasi atau manual; jika sepotong informasi dikumpulkan melalui formulir kertas atau melalui Web; atau jika informasi ditempatkan di lemari arsip atau database besar. Rincian fisik ini didefinisikan selama desain ketika model logis disempurnakan menjadi model fisik. Model-model ini memberikan informasi yang diperlukan untuk akhirnya membangun sistem. Dengan berfokus pada aktivitas logis terlebih dahulu, analis dapat fokus pada bagaimana bisnis harus berjalan tanpa terganggu dengan detail implementasi.

Model proses bisnis menggambarkan berbagai aktivitas yang jika digabungkan bersama, mendukung proses bisnis. Proses bisnis biasanya melintasi departemen fungsional (misalnya penciptaan produk baru akan melibatkan berbagai kegiatan yang akan menggabungkan upaya banyak karyawan di banyak departemen). Selanjutnya, dari perspektif berorientasi objek, mereka melintasi beberapa objek. Dengan demikian, banyak pendekatan pengembangan sistem berorientasi objek sebelumnya cenderung mengabaikan pemodelan proses bisnis. Padahal, bila pendekatan-pendekatan tersebut digunakan dengan benar, mereka adalah alat yang sangat kuat untuk mengkomunikasikan pemahaman analis tentang kebutuhan saat ini dengan pengguna sistem.

Pada kegiatan belajar ini, akan dibahas lebih spesifik penggunaan *activity diagram* dalam konteks pemodelan proses bisnis. Model proses menggambarkan bagaimana sistem bisnis beroperasi. Model ini menggambarkan proses atau kegiatan yang dilakukan dan bagaimana benda (data) bergerak di antara mereka. Sebuah model proses dapat digunakan untuk mendokumentasikan sistem saat ini (yaitu, sistem yang sedang berjalan) atau sistem baru yang sedang dikembangkan (yaitu sistem yang akan dibuat), apakah terkomputerisasi atau tidak.

#### A. ELEMEN ACTIVITY DIAGRAM

*Activity diagram* menggambarkan bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* memiliki beberapa elemen (notasi) dasar seperti disajikan pada Gambar 6.1(a) dan 6.1(b).

##### 1. Action dan Activity

Tindakan (*action*) dan aktivitas (*activity*) adalah sesuatu yang dilakukan dalam suatu urusan bisnis tertentu. Tindakan dan aktivitas dapat mewakili perilaku manual atau terkomputerisasi. Pada beberapa referensi, keduanya digambarkan dengan simbol yang sama (simbol *action*) (Dennis, 2009), namun pada software UML seperti "Rational Rose", keduanya digambarkan dengan simbol yang berbeda, seperti disajikan pada Gambar 6.1(a).

NOTASI	DEFINISI
	<i>Initial Node</i> (node/simpul awal): Menggambarkan awal dari serangkaian tindakan atau kegiatan.
	<i>Action</i> : Digunakan untuk mewakili serangkaian tindakan yang tidak dapat dikomposisi lebih lanjut Diberi label sesuai nama tindakannya

NOTASI	DEFINISI
	<p><i>Activity:</i> Digunakan untuk mewakili serangkaian tindakan yang dapat dikomposisi lebih lanjut Diberi label sesuai nama tindakannya</p>
	<p><i>Object Node:</i> Digunakan untuk mewakili objek/benda Diberi label sesuai nama objek</p>
	<p><i>Control Flow (Aliran Kontrol):</i> Memperlihatkan urutan eksekusi</p>
	<p><i>Object Flow (aliran objek):</i> Memperlihatkan aliran suatu objek dari satu <i>action</i> (atau <i>activity</i>) ke <i>action</i> (atau aksi) lain</p>
	<p><i>Decision Node (node/simpul keputusan):</i> Digunakan untuk mewakili kondisi pengujian untuk memastikan bahwa aliran kontrol atau aliran objek hanya mengarah ke satu jalur keputusan Diberi label dengan kriteria keputusan untuk menuju ke jalur tertentu</p>
	<p><i>Merge Node (node/simpul penggabungan):</i> Digunakan untuk menyatukan kembali jalur keputusan berbeda yang dibuat menggunakan simpul keputusan</p>

Gambar 6.1(a)  
Notasi Dasar *Activity Diagram* (pada UML 2.x)

NOTASI	DEFINISI
	<p><i>Forking (percabangan):</i> Menunjukkan beberapa aliran yang paralel (aliran-aliran konkuren)</p>
	<p><i>Join:</i> Menunjukkan beberapa aliran yang secara bersama-sama masuk pada suatu titik</p>
	<p><i>Final-activity Node:</i> Digunakan untuk menghentikan semua aliran kontrol dan aliran objek dalam suatu kegiatan (<i>activity</i>).</p>

NOTASI	DEFINISI
	<i>Final-flow node:</i> Digunakan untuk menghentikan aliran kontrol atau aliran objek tertentu
	<i>Swimlane:</i> Digunakan untuk mengelompokkan <i>activity</i> berdasarkan <i>actor</i> (pada UML 1)

Gambar 6.1(b)  
Notasi Dasar *Activity Diagram* (Lanjutan)

*Action* dan *activity* harus memiliki nama yang dimulai dengan kata kerja dan diakhiri dengan kata benda (misalnya: *Membuat Janji* atau *Mengatur Pembayaran*). Nama harus singkat, namun mengandung informasi yang cukup sehingga pembaca dapat dengan mudah memahami apa yang mereka lakukan. Perbedaan antara suatu tindakan (*action*) dan suatu kegiatan (*activity*) adalah bahwa suatu *kegiatan* dapat diuraikan lebih lanjut menjadi serangkaian kegiatan dan/atau tindakan, sedangkan suatu *tindakan* mewakili bagian sederhana yang tidak dapat dikomposisikan dari keseluruhan perilaku yang dimodelkan. Dengan demikian dapat dinyatakan bahwa suatu *tindakan* adalah kasus khusus dari suatu *aktivitas/kegiatan*. Biasanya, hanya *aktivitas* yang digunakan untuk proses bisnis atau pemodelan aliran kerja. Selanjutnya, dalam kebanyakan kasus, setiap *aktivitas* dikaitkan dengan *use case* (Dennis, 2009).

Gambar 6.2 memperlihatkan sebuah *activity*, sedangkan Gambar 6.3 memperlihatkan beberapa *action* (Nugroho, 2005).

Memproses Pesanan

Gambar 6.2  
Status *Activity*

$X = X + 1$

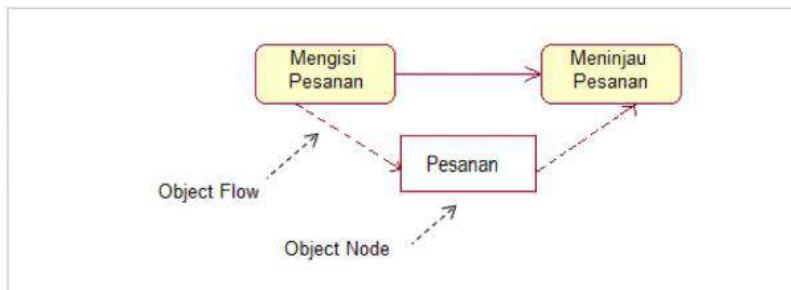
$D = b * b - 4 * a * c$

Gambar 6.3  
Status *Action*

## 2. Object Node

*Action* dan *activity* biasanya memodifikasi atau mengubah objek. *Object node* (node objek) memodelkan objek-objek ini dalam *activity diagram*. *Object node* digambarkan dalam *activity diagram* sebagai persegi panjang (lihat Gambar 6.1). Nama kelas/objek ditulis di dalam persegi panjang. Pada dasarnya, *object node* mewakili aliran informasi dari satu *activity* atau *action* ke *activity* atau *action* lain. Dengan kata lain,

*object node* adalah luaran yang dihasilkan oleh sebuah *activity* atau *action* yang akan menjadi masukan bagi *activity* atau *action* lain. Sistem "Pemesanan" sederhana pada Gambar 6.4 menunjukkan *object node* dengan nama *Pesanan*, mengalir dari aktivitas *Mengisi Pesanan* ke *Meninjau Pesanan*.



Gambar 6.4  
Contoh *Object Node* pada Sistem Pemesanan

### 3. Control Flow dan Object Flow

*Control flow* (aliran kontrol) memodelkan jalur eksekusi melalui proses bisnis. Aliran kontrol digambarkan sebagai garis padat dengan panah di ujungnya yang menunjukkan arah aliran. Aliran kontrol disertakan pada suatu *action* atau *activity* (Gambar 6.4).

*Object flow* (aliran objek) memodelkan aliran objek melalui proses bisnis. Karena *activity* dan *action* memodifikasi atau mengubah objek, *object flow* diperlukan untuk menunjukkan objek aktual yang mengalir masuk dan atau keluar dari suatu *action* atau *activity*. *Object flow* digambarkan sebagai garis putus-putus dengan panah di atasnya yang menunjukkan arah aliran (Gambar 6.4).

### 4. Control Node

Ada tujuh jenis node kontrol (*control node*) dalam diagram aktivitas: node awal (*initial node*), node akhir aktivitas (*final-activity node*), node akhir aliran (*final-flow node*), node keputusan (*decision node*), node penggabungan (*merge node*), node percabangan (*forking*), dan node join (lihat Gambar 6.1).

**Node awal** menggambarkan awal dari serangkaian tindakan atau aktivitas. Node awal ditampilkan sebagai lingkaran kecil yang berisi. **Node akhir aktivitas** digunakan untuk menghentikan proses yang dimodelkan. Setiap kali simpul aktivitas akhir tercapai, semua tindakan dan aktivitas diakhiri dengan segera, terlepas dari apakah mereka selesai. Node akhir aktivitas direpresentasikan sebagai lingkaran yang mengelilingi lingkaran kecil yang berisi. Node kontrol baru yang baru ditambahkan ke *activity diagram* di UML 2.0 adalah **node akhir aliran**. Node akhir aliran mirip dengan node akhir aktivitas, kecuali bahwa ia menghentikan jalur eksekusi tertentu dalam proses bisnis, tetapi memungkinkan jalur yang lain untuk melanjutkan proses. Node akhir aliran ditampilkan sebagai lingkaran kecil dengan tanda "x" di dalamnya.

**Node keputusan** dan **node penggabungan** mendukung pemodelan struktur keputusan dari proses bisnis. **Node keputusan** digunakan untuk mewakili kondisi pengujian aktual untuk menentukan jalur keluar mana dari simpul keputusan yang akan dilalui. Dalam hal ini, setiap jalur yang keluar harus dilabeli dengan suatu kondisi tertentu. Kondisi ini mewakili nilai pengujian untuk jalur tertentu yang akan dieksekusi. **Node penggabungan** digunakan untuk menyatukan kembali beberapa jalur yang saling eksklusif yang telah dipisah berdasarkan keputusan sebelumnya.

**Node percabangan** (*forking*) dan **node join** memungkinkan suatu proses paralel dan konkuren dimodelkan. **Node percabangan** digunakan untuk membagi perilaku proses bisnis menjadi beberapa aliran paralel atau bersamaan. Tidak seperti node keputusan, jalur tidak saling eksklusif (yaitu, kedua jalur dieksekusi bersamaan). **Node join** mirip dengan node penggabungan. Node join hanya menyatukan kembali aliran paralel yang terpisah dalam proses bisnis menjadi aliran tunggal.

##### 5. *Swimlane*

Ada kalanya sangat membantu jika memecah *activity diagram* sedemikian rupa menjadi kolom-kolom untuk membagi tanggung jawab kepada objek atau individu yang melakukan aktivitas. Ini sangat berguna ketika membuat model alur kerja bisnis, yang dicapai melalui penggunaan *swimlane*.

Partisi membagi bagian dalam *activity diagram* berdasarkan aktor atau divisi yang melakukan aktivitas tersebut. Partisi dalam aktivitas biasanya menggunakan notasi *swimlane* yang berupa kotak untuk setiap aktor/divisi dalam aktivitas. Contoh *activity diagram* dengan *swimlane* disajikan pada Gambar 6.5. Jika dibandingkan dengan Gambar 6.4 di mana aktivitas *Mengisi Pesanan* dan *Meninjau Pesanan* tidak mengandung informasi mengenai objek yang bertanggungjawab melakukan aktivitas tersebut, Gambar 6.5 memperlihatkan itu.



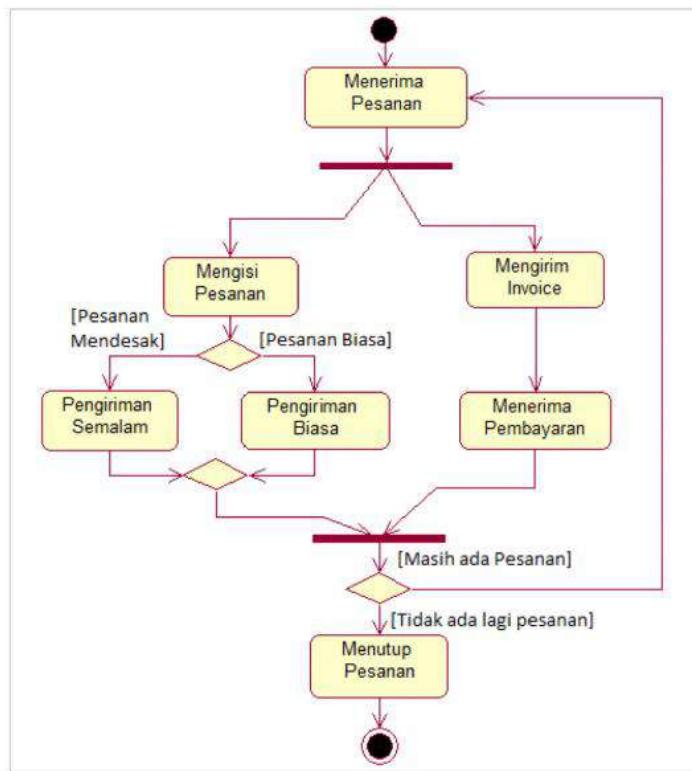
Gambar 6.5  
Contoh *Activity Diagram* menggunakan *Swimlane*

## B. MENGGAMBAR ACTIVITY DIAGRAM

*Activity diagram* digambar/dibaca dari atas ke bawah, memungkinkan percabangan untuk menunjukkan kondisi, keputusan, dan atau memiliki kegiatan paralel.

Contoh:

Gambar 6.6 menunjukkan sebuah contoh sederhana dari sebuah *activity diagram* (Memproses Pesanan).



Gambar 6.6  
Contoh Activity Diagram "Memproses Pesanan" (tanpa menggunakan Swimlane)

Node pada sebuah *activity diagram* disebut sebagai *action* (tindakan), bukan *activity* (aktivitas/kegiatan). Jelasnya, *activity* mengacu pada serangkaian *action*, sehingga diagram tersebut menampilkan sebuah *activity* yang tersusun dari *action* (Dennis, 2009).

*Activity diagram* pada Gambar 6.6 dimulai pada *node awal* dan kemudian melakukan tindakan *menerima pesanan*. Terdapat dua tindakan yang dilakukan setelah tindakan *menerima pesanan*, yaitu tindakan *mengisi pesanan* dan *mengirim invoice* melalui sebuah *node percabangan* yang memiliki satu aliran masuk dan dua aliran keluar. Hal tersebut menunjukkan bahwa *mengisi pesanan*, *mengirim invoice*, dan tindakan setelahnya dilakukan secara paralel. Itu berarti bahwa rangkaian-rangkaian antara tindakan tersebut tidak saling berkaitan. Seseorang dapat *mengisi pesanan*, *mengirim invoice*, *mengantar barang*, dan kemudian *menerima pembayaran*, atau

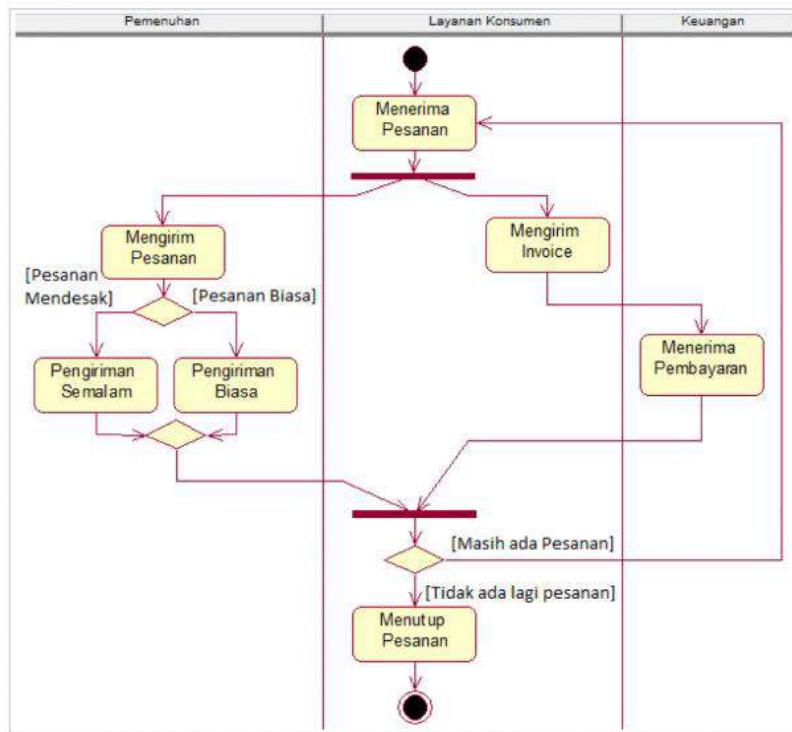
seseorang dapat *mengirim invoice*, *menerima pembayaran*, *mengisi pesanan*, dan kemudian *mengantar barang*; begitulah gambarannya.

Suatu tindakan paralel perlu sinkronisasi. Sebuah pesanan tidak dapat ditutup sebelum diantar dan dibayar. Kita menampilkannya dengan join sebelum kegiatan *menutup pesanan*. Dengan sebuah join, aliran keluar dilakukan hanya jika seluruh aliran masuk telah mencapai join. Dalam kasus Gambar 6.6, tindakan *menutup pesanan* dapat dilakukan hanya jika telah *menerima pembayaran* dan *mengantarkan barang*.

Behavior kondisional digambarkan oleh node *keputusan (decision)* dan node *penggabungan (merge)*. Sebuah keputusan (disebut cabang di UML 1) memiliki sebuah aliran masuk tunggal dan beberapa aliran keluar. Setiap aliran keluar memiliki sebuah kondisi *Boolean* yang diletakkan di dalam tanda kurung kotak. Setiap saat sampai pada sebuah keputusan, hanya dapat mengambil satu aliran keluar. Pada Gambar 6.6, setelah sebuah pesanan telah diisi, muncul sebuah keputusan. Jika mendapat pesanan mendesak, maka dilakukan *pengiriman semalam*. Jika sebaliknya, maka dilakukan *pengiriman biasa*.

Sebuah penggabungan (*merge*) memiliki beberapa aliran *input* dan sebuah *output* tunggal. Sebuah penggabungan menandai berakhirnya behavior kondisional yang dimulai oleh sebuah keputusan.

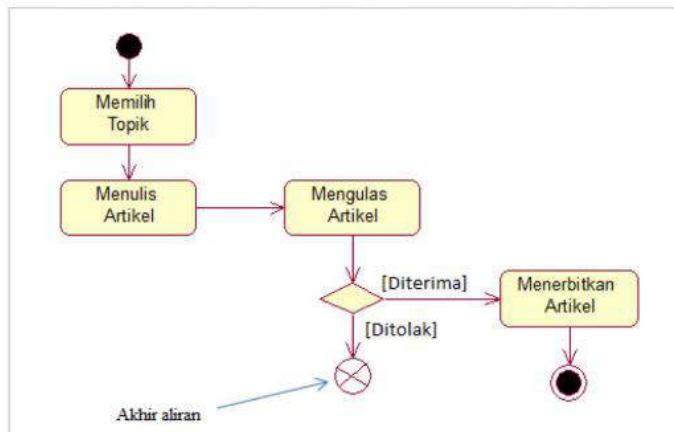
*Activity diagram* memberitahu apa yang terjadi, tetapi diagram ini tidak memberitahu tentang siapa yang melakukan apa. Dalam pemrograman, hal ini berarti diagram ini tidak menyampaikan *class* mana yang bertanggungjawab untuk setiap tindakan. Dalam pemodelan proses bisnis, hal ini tidak menyampaikan bagian mana dari sebuah organisasi yang melakukan tindakan apa. Hal ini mungkin tidak menjadi masalah, sebab acapkali lebih masuk akal untuk berkonsentrasi pada apa yang telah dilakukan daripada siapa melakukan apa dari behavior tersebut. Namun demikian, jika ingin menunjukkan siapa melakukan apa, maka dapat memisahkan *activity diagram* ke dalam partisi-partisi yang menampilkan tindakan mana yang dilakukan oleh sebuah *class* atau organisasi. Gambar 6.7 merupakan sebuah contoh sederhana yang menunjukkan bagaimana tindakan (*action*) terlibat supaya pemrosesan dapat dipisahkan ke dalam berbagai departemen.



Gambar 6.7  
Contoh Activity Diagram "Memproses Pesanan" (Menggunakan *Swimlane*)

Pembuatan partisi pada Gambar 6.7 merupakan pembuatan partisi sederhana satu dimensi. Model ini sering disebut sebagai *swimlane* yang digunakan di UML 1.x, sedangkan pada UML 2.x telah menggunakan jaringan dua dimensi.

Ada kalanya ditemukan aliran-aliran yang berhenti walaupun *activity* secara keseluruhan belum selesai. Sebuah *akhir aliran* menunjukkan akhir sebuah aliran tertentu, tanpa menghentikan seluruh *activity*.



Gambar 6.8  
Akhir Aliran dalam Activity Diagram

Gambar 6.8 menunjukkan aliran yang berhenti (ketika sebuah artikel ditolak). Jika sebuah artikel ditolak, pesan dihentikan oleh *akhir aliran*. Tidak seperti sebuah *akhir activity*, sisa *activity* yang lain masih dapat berlanjut.



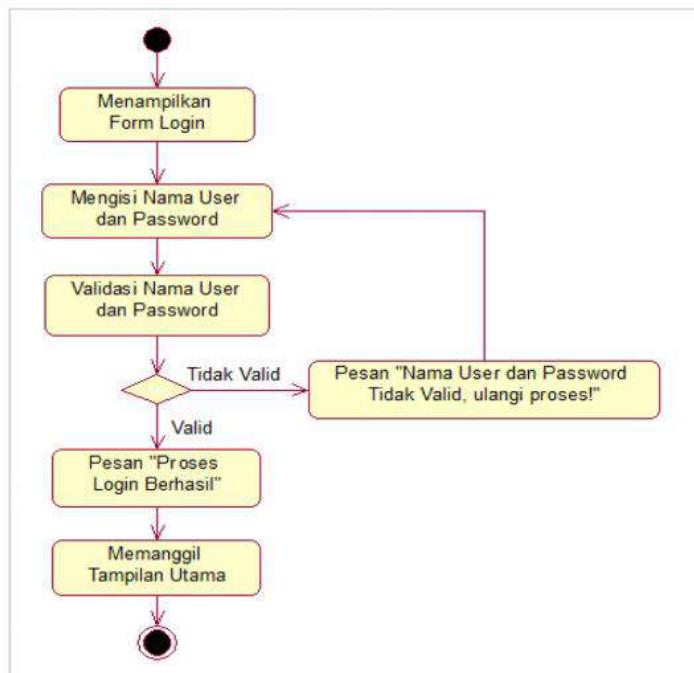
### Latihan

Untuk memperdalam pemahaman Anda mengenai materi di atas, kerjakanlah latihan berikut!

- 1) Jelaskan kegunaan dari *activity diagram*!
- 2) Jelaskan perbedaan antara *activity* dan *action*!
- 3) Gambarkan *activity diagram* berdasarkan deskripsi *use case* pada Gambar 5.1 (Modul 5)!

#### Petunjuk Jawaban Latihan

- 1) *Activity diagram* digunakan untuk penggambaran semua jenis model proses, seperti: pemodelan proses logika prosedural, pemodelan proses bisnis, dan pemodelan jalur kerja.
- 2) Suatu *activity* dapat diurai lebih lanjut menjadi serangkaian kegiatan dan atau tindakan, sedangkan suatu *action* adalah suatu tindakan sederhana yang mewakili suatu bagian, yang tidak dapat dikomposisikan dari keseluruhan perilaku yang dimodelkan. Jadi, suatu *action* adalah kasus khusus dari suatu *activity*.
- 3)



*Activity diagram* di atas dibuat sesuai dengan alur peristiwa yang disajikan pada Deskripsi Use Case. Gambar tersebut dapat mengalami perubahan bentuk, tergantung konsep alur peristiwa yang disajikan pada Deskripsi Use Case.



## Rangkuman

Dari sudut pandang pengembangan sistem berorientasi objek, pemodelan proses bisnis telah terbukti bermanfaat. Salah satu kelemahan dari pemodelan proses bisnis adalah kecenderungan memfokuskan proses pengembangan sistem ke arah dekomposisi fungsional. Namun, jika digunakan dengan hati-hati, ini dapat meningkatkan pengembangan sistem berorientasi objek. UML (khususnya UML 2.0) mendukung pemodelan proses menggunakan *activity diagram*. *Activity diagram* terdiri dari aktivitas atau tindakan, objek, aliran kontrol, aliran objek, dan satu set node kontrol yang berbeda (awal, aktivitas akhir, aliran akhir, keputusan, penggabungan, percabangan, dan penggabungan). Selanjutnya, *swimlane* dapat digunakan untuk meningkatkan keterbacaan diagram. Diagram aktivitas sangat berguna untuk membantu analis dalam mengidentifikasi *use case* yang relevan untuk sistem informasi yang dikembangkan.

Kekuatan *activity diagram* terletak pada kemampuannya mendukung lingkungan atau situasi yang paralel. Hal ini yang menyebabkan *activity diagram* menjadi peranti yang bagus untuk menggambarkan aliran kerja dan pemodelan proses. Konsep percabangan dan join memberikan keuntungan ketika menggambarkan algoritma paralel untuk program yang bersamaan.



## Tes Formatif 1

Pilihlah satu jawaban yang paling tepat!

- 1) Versi UML yang mengkategorikan *activity diagram* sebagai kasus khusus *state diagram* adalah....
  - A. UML Versi 1.0
  - B. UML Versi 2.0
  - C. UML Versi 2.1
  - D. Pilihan jawaban B dan C benar
  
- 2) Object dalam *activity diagram* yang merepresentasikan benda, yang biasanya dicantumkan sebagai hasil dari *action* atau yang akan dipakai oleh *action* adalah....
  - A. node awal (*initial node*)
  - B. node aktivitas akhir (*final-activity node*)
  - C. node objek (*object node*)
  - D. jawaban A dan B benar

- 3) Node yang digunakan untuk memastikan bahwa aliran kontrol atau aliran objek hanya mengarah ke satu jalur keputusan adalah....
  - A. node percabangan (*fork*)
  - B. node keputusan (*decision*)
  - C. node penggabungan (*merge*)
  - D. node join
- 4) Untuk menunjukkan objek aktual yang mengalir masuk dan keluar dari suatu *action* atau *activity* digunakan....
  - A. aliran kontrol (*control flow*)
  - B. aliran objek (*object flow*)
  - C. node kontrol (*control node*)
  - D. Semua jawaban A, B, dan C salah
- 5) Node yang digunakan untuk menghentikan aliran kontrol tertentu adalah....
  - A. node awal (*initial node*)
  - B. node aktivitas akhir (*final-activity node*)
  - C. nodel aliran akhir (*final-flow node*)
  - D. jawaban B dan C benar
- 6) Simbol yang digunakan untuk mengelompokkan *action* berdasarkan *actor*/pelakunya (khususnya pada UML versi 1.xx) adalah....
  - A. *class*
  - B. node penggabungan (*merge*)
  - C. node join
  - D. *swimlane*
- 7) Simbol berikut ini merupakan simbol untuk menggambarkan ... pada *activity diagram*:  

  - A. node awal (*initial node*)
  - B. nodel aliran akhir (*final-flow node*)
  - C. node aktivitas akhir (*final-activity node*)
  - D. Semua jawaban A, B, dan C benar
- 8) *Activity diagram* dapat digunakan untuk memodelkan event-event yang terjadi dalam suatu use case. Pernyataan tersebut:
  - A. Benar
  - B. Salah

- 9) *Activity diagram* memiliki kemiripan dengan diagram alir (*flowchart*). Namun demikian, perbedaan prinsip antara *activity diagram* dan diagram alir adalah diagram alir mendukung behavior paralel. Pernyataan tersebut:
- Benar
  - Salah
- 10) *Swimlane* adalah notasi yang harus disertakan pada setiap penggambaran *activity diagram*.
- Benar
  - Salah

Cocokkanlah jawaban Anda dengan Kunci Jawaban Tes Formatif 1 yang terdapat di bagian akhir modul ini. Hitunglah jawaban yang benar. Kemudian, gunakan rumus berikut untuk mengetahui tingkat penguasaan Anda terhadap materi Kegiatan Belajar 1.

Tingkat Penguasaan =

$$\frac{\text{Jumlah Jawaban yang Benar}}{\text{Jumlah Soal}} \times 100$$

Arti tingkat penguasaan

<70%

70% - 79%

80% - 89%

90% - 100%

kurang

cukup

baik

baik sekali

Apabila mencapai tingkat penguasaan 80% atau lebih, Anda dapat meneruskan dengan Kegiatan Belajar 2. **Bagus!** Jika masih di bawah 80%, Anda harus mengulangi materi Kegiatan Belajar 1, terutama bagian yang belum dikuasai.

## Pemodelan Struktural dengan *Class Diagram* dan *Object Diagram*

Semasa fase analisis, analis membuat model fungsional untuk mewakili bagaimana sistem bisnis akan berperilaku. Pada saat yang sama, analis perlu memahami informasi yang digunakan dan dibuat oleh sistem bisnis (misalnya informasi pelanggan, informasi pesanan).

Model struktural adalah model konseptual yang secara formal merepresentasikan objek yang digunakan dan dibuat oleh sistem bisnis. Model ini menggambarkan orang, tempat, atau hal-hal mengenai informasi apa yang ditangkap dan bagaimana mereka saling berhubungan. Model struktural menggambarkan struktur data yang mendukung proses bisnis dalam suatu organisasi. Semasa fase analisis, model struktural menyajikan organisasi logis dari data tanpa memperlihatkan bagaimana data disimpan, dibuat, atau dimanipulasi, sehingga analis dapat fokus pada prosedur bisnis tanpa terganggu oleh detail teknisnya. Kemudian semasa fase desain, model struktural diperbarui untuk mencerminkan dengan tepat bagaimana data akan disimpan dalam database dan file.

Hal penting lainnya untuk diingat adalah bahwa pada tahap pemodelan analisis, model struktural tidak mewakili komponen perangkat lunak atau kelas dalam bahasa pemrograman berorientasi objek, meskipun model struktural memang mengandung unsur kelas, atribut, operasi, dan hubungan antara kelas. Penyempurnaan kelas-kelas awal ini di masa mendatang (fase desain) akan menjadi objek pada tingkat pemrograman.

Bagian ini akan fokus pada pembuatan model struktural objek menggunakan *class diagram* dan *object diagram*. Tahap awal menjelaskan tentang *class diagram*, selanjutnya menjelaskan tahapan-tahapan pembuatan *class diagram*. Pada bagian akhir menjelaskan tentang *object diagram*. Dengan menggunakan teknik ini, dimungkinkan untuk menunjukkan semua objek dalam sistem bisnis.

### A. CLASS DIAGRAM

*Class diagram* (diagram kelas) tidak hanya telah digunakan secara luas, tetapi juga memiliki banyak konsep pemodelan, dimulai dari elemen-elemen dasar hingga konsep-konsep tingkat lanjut. Modul ini akan membahas konsep dasar dalam membuat *class diagram*. Konsep-konsep tersebut merupakan hal pertama yang harus dipahami

dan dikenali, karena konsep-konsep tersebut akan mencakup lebih dari separuh usaha dalam membangun *class diagram*.

*Class diagram* mendeskripsikan jenis-jenis objek dalam sistem dan berbagai macam hubungan statis yang terdapat di antara objek-objek yang ada. *Class diagram* juga menunjukkan properti dan operasi sebuah kelas (*class*) dan batasan-batasan yang terdapat dalam hubungan-hubungan objek tersebut (Fowler, 2005). UML menggunakan istilah fitur sebagai istilah umum yang meliputi properti dan operasi sebuah kelas.

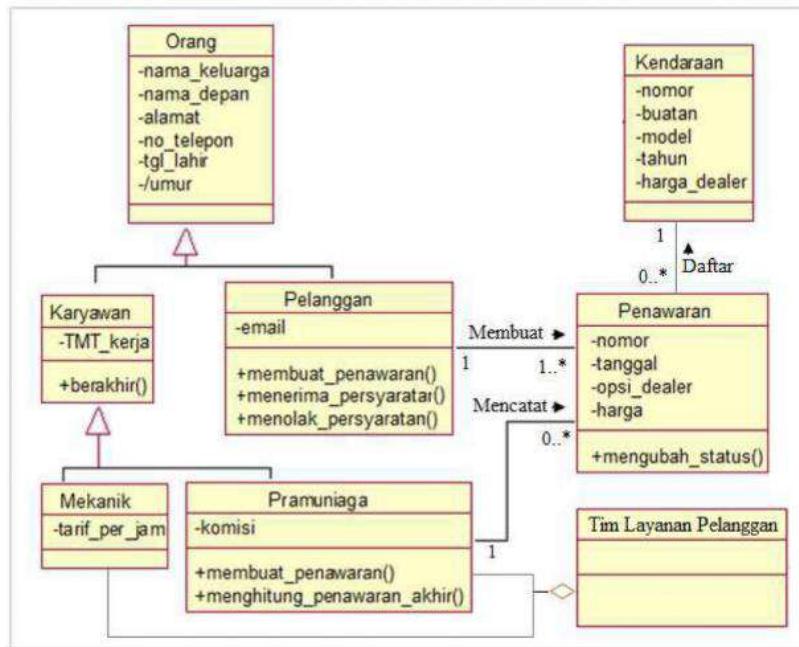
*Class diagram* sangat mirip dengan ERD (*entity relationship diagram*). Namun, *class diagram* menggambarkan kelas, yang mencakup atribut, perilaku, dan status, sementara entitas dalam ERD hanya menyertakan atribut (tidak terdapat operasi/*method*). Ruang lingkup *class diagram* sama seperti ERD, yaitu mencakup seluruh sistem. Pembahasan pertama menyajikan sintaks diagram kelas, selanjutnya menyajikan cara *class diagram* digambarkan.

### 1. Elemen-elemen *Class Diagram*

Gambar 6.9 menunjukkan contoh *class diagram* yang dibuat untuk menggambarkan kelas dan hubungan yang diperlukan untuk sebagian *use case* pada "sistem penjualan kendaraan".

#### a. Class

Blok bangunan utama dari *class diagram* adalah kelas (*class*), yang menyimpan dan mengelola informasi dalam sistem. Semasa fase analisis, kelas merujuk pada orang-orang, tempat, peristiwa, dan hal-hal yang akan diperoleh informasinya. Kemudian, semasa fase desain dan implementasi, kelas dapat merujuk pada komponen spesifik seperti jendela program, form, dan objek lain yang digunakan untuk membangun sistem. Setiap kelas digambar dengan menggunakan tiga bagian persegi panjang dengan nama kelas di bagian atas, atribut di tengah, dan metode (juga disebut operasi) di bagian bawah. Kita dapat mengidentifikasi bahwa Orang, Karyawan, Mekanik Toko, Tenaga Penjual, Pelanggan, Penawaran, dan Kendaraan adalah kelas-kelas dalam Gambar 6.9. Atribut suatu kelas dan nilainya mendefinisikan keadaan masing-masing objek yang dibuat dari kelas, dan perilaku diwakili oleh metode (*method*).



Gambar 6.9  
Contoh Class Diagram "Sistem Penjualan Kendaraan"

#### b Attribute

*Attribute* (atribut) adalah properti kelas yang ingin diperoleh informasinya. Perhatikan bahwa kelas *Orang* pada Gambar 6.9 berisi atribut nama belakang, nama depan, alamat, telepon. Kadang-kadang, kita mungkin ingin menyimpan atribut turunan, yang merupakan atribut yang dapat dihitung atau berasal dari atribut lain dan karenanya tidak disimpan. Atribut turunan yang terkait ditunjukkan dengan menempatkan garis miring (/) di depan nama atribut. Perhatikan bagaimana kelas *Orang* berisi atribut turunan yang disebut "/ Umur" yang dapat diturunkan dengan mengurangi "tanggal lahir" orang tersebut dari tanggal saat ini. Dimungkinkan juga untuk memperlihatkan visibilitas atribut pada diagram. Visibilitas berkaitan dengan tingkat penyembunyian informasi yang diberlakukan untuk atribut. Visibilitas atribut dapat berupa *public* (+), *protected* (#), atau *private* (-).

Elemen	Simbol			
<b>Class (kelas):</b> <ul style="list-style-type: none"> <li>- Merepresentasikan jenis orang, tempat, atau hal yang harus ditangkap dan disimpan oleh sistem</li> <li>- Memiliki nama yang dicetak tebal dan berada di tengah kompartemen atas</li> <li>- Memiliki daftar atribut di kompartemen tengahnya</li> <li>- Memiliki daftar operasi di kompartemen bawahnya</li> </ul>	<table border="1"> <tr> <td>Nama kelas</td></tr> <tr> <td>Nama Atribut / Nama atribut turunan</td></tr> <tr> <td>Nama Operasi()</td></tr> </table>	Nama kelas	Nama Atribut / Nama atribut turunan	Nama Operasi()
Nama kelas				
Nama Atribut / Nama atribut turunan				
Nama Operasi()				

<b>Attribute (atribut):</b>	<ul style="list-style-type: none"> <li>- Merepresentasikan properti yang mendeskripsikan status suatu objek</li> <li>- Dapat diturunkan dari atribut lain, ditunjukkan dengan menempatkan garis miring sebelum nama atribut</li> </ul>	Nama atribut / Nama atribut turunan
<b>Method (metode/operasi):</b>	<ul style="list-style-type: none"> <li>- Merepresentasikan tindakan atau fungsi yang dapat dilakukan oleh kelas</li> <li>- Dapat diklasifikasikan sebagai operasi konstruktur, kueri, atau pembaruan</li> <li>- Berisi tanda kurung yang mungkin berisi parameter atau informasi khusus yang diperlukan untuk menjalankan operasi</li> </ul>	Nama operasi ()
<b>Association (asosiasi):</b>	<ul style="list-style-type: none"> <li>- Merepresentasikan hubungan antara beberapa kelas, atau kelas dan dirinya sendiri</li> <li>- Dapat dilabeli dengan frasa kata kerja atau nama peran, atau apa saja yang lebih mewakili hubungan</li> <li>- Berisi simbol multiplisitas, yang mewakili frekuensi minimum dan maksimum <i>instance</i> kelas dapat dikaitkan dengan <i>instance</i> kelas terkait</li> </ul>	<u>1..* frase kata kerja 0..1</u>

**Gambar 6.10**  
Elemen-elemen Class Diagram

Atribut publik adalah atribut yang tidak disembunyikan dari objek lain. Dengan demikian, objek lain dapat mengubah nilainya. Atribut yang *protected* (dilindungi) adalah atribut yang disembunyikan dari semua kelas lain kecuali *subclass* langsungnya. Atribut privat adalah atribut yang disembunyikan dari semua kelas lainnya. Visibilitas default untuk atribut biasanya bersifat publik.

#### c. Method

Operasi adalah tindakan atau fungsi yang dapat dilakukan oleh suatu kelas. Fungsi-fungsi yang tersedia untuk semua kelas (misalnya, membuat instance baru, mengembalikan nilai untuk atribut tertentu, menetapkan nilai untuk atribut tertentu, atau menghapus sebuah instance) tidak secara eksplisit ditampilkan dalam persegi panjang kelas. Sebagai gantinya, hanya operasi-operasi yang unik untuk kelas yang dimasukkan, seperti operasi "berakhir" dan "membuat penawaran" pada Gambar 6.9. Perhatikan bahwa kedua operasi diikuti oleh tanda kurung. Operasi harus ditunjukkan dengan tanda kurung yang kosong, atau diisi dengan beberapa nilai yang mewakili parameter yang diperlukan operasi untuknya. Seperti halnya atribut, visibilitas suatu operasi dapat

ditetapkan sebagai *public* (+), *protected* (#), atau *private* (-). Visibilitas default untuk operasi biasanya bersifat publik.

Ada tiga jenis operasi yang dapat terkandung dalam kelas: *constructor*, *query*, dan *update*. Operasi *constructor* membuat instance baru dari kelas. Misalnya, kelas kendaraan mungkin memiliki operasi yang disebut *insert ()* yang membuat turunan kendaraan baru. Karena operasi yang tersedia untuk semua kelas (misalnya membuat instance baru) tidak ditampilkan pada diagram kelas, kita tidak akan melihat metode konstruktur pada Gambar 6.9.

Operasi *query* membuat informasi tentang keadaan suatu objek tersedia untuk objek lain, tetapi itu tidak akan mengubah objek dengan cara apapun. Misalnya, operasi "menghitung penawaran akhir ()" yang menentukan kapan tenaga penjual terakhir menulis penawaran akan mengakibatkan objek diakses oleh sistem, tetapi tidak akan membuat perubahan apapun terhadap informasinya. Jika operasi *query* hanya meminta informasi dari atribut di kelas (misalnya, nama pelanggan, alamat, atau telepon), maka itu tidak ditampilkan pada diagram, karena dianggap bahwa semua objek memiliki operasi yang menghasilkan nilai atribut mereka.

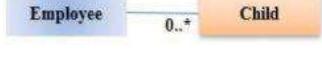
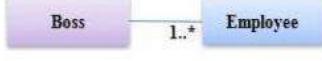
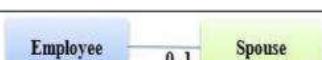
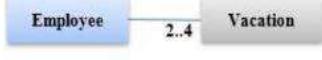
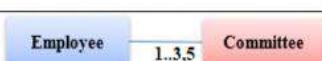
Operasi *update* akan mengubah nilai beberapa atau semua atribut objek, yang dapat mengakibatkan perubahan pada status objek. Misalnya pertimbangan untuk mengubah status penawaran dari "dibuka" menjadi "ditutup" dengan operasi yang disebut "ubah status()", atau mengaitkan pelanggan dengan penawaran tertentu dengan "membuat penawaran".

#### d. Association

Tujuan utama *class diagram* adalah untuk menunjukkan asosiasi atau hubungan/relasi, yang dimiliki suatu kelas terhadap kelas yang lain. Ini digambarkan pada diagram dengan garis yang ditarik antar kelas. Asosiasi ini sangat mirip dengan relasi yang ditemukan di ERD. Asosiasi dikelola oleh *referensi*, yang mirip dengan pointer dan dikelola secara internal oleh sistem (tidak seperti dalam model relasional di mana relasi dikelola oleh kunci asing dan kunci primer).

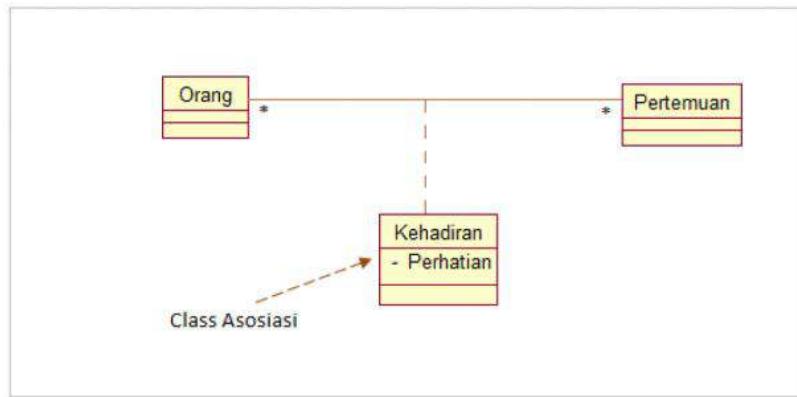
Ketika beberapa kelas berbagi asosiasi (atau kelas berbagi asosiasi dengan dirinya sendiri), sebuah garis dibuat dan diberi label dengan nama asosiasi atau peran yang dimainkan kelas dalam asosiasi tersebut. Sebagai contoh, pada Gambar 6-2-1, dua kelas *pelanggan* dan *penawaran* dikaitkan satu sama lain setiap kali pelanggan mengajukan penawaran. Dengan demikian, garis berlabel *membuat/mengajukan* menghubungkan *pelanggan* dan *penawaran*, yang menunjukkan dengan tepat bagaimana kedua kelas tersebut dikaitkan satu sama lain. Juga, perhatikan bahwa ada segitiga padat kecil di samping nama asosiasi. Segitiga menunjukkan arah untuk dikaitkan dengan nama asosiasi. Pada Gambar 6.9, asosiasi *mengajukan* memiliki segitiga, yang menunjukkan bahwa asosiasi tersebut harus dibaca *pelanggan* mengajukan *penawaran*. Dicantumkannya segitiga dimaksudkan hanya untuk meningkatkan keterbacaan diagram.

Asosiasi juga memiliki *multiplicity* (multiplisitas), yang menunjukkan bagaimana instance objek dapat dikaitkan dengan instance lainnya. Angka ditempatkan pada jalur asosiasi untuk menunjukkan jumlah minimum dan maksimum yang dapat dihubungkan melalui asosiasi dalam format angka minimum angka maksimum, seperti pada Gambar 6.11 (Dennis, 2012). Ini identik dengan modalitas dan kardinalitas suatu relasi pada ERD. Angka-angka menentukan asosiasi dari kelas di ujung garis asosiasi sampai akhir dengan nomor. Sebagai contoh, pada Gambar 6.9, ada "1 .." pada ujung *penawaran* dari asosiasi *pelanggan membuat penawaran*. Ini berarti bahwa sebuah pelanggan dapat dikaitkan dengan satu melalui banyak penawaran berbeda. Pada akhir pelanggan dari asosiasi yang sama ini ada "1", yang berarti bahwa suatu penawaran harus dikaitkan dengan satu dan hanya satu (1) pelanggan.

Instance	Perwakilan Instance	Contoh Diagram	Keterangan Diagram
<i>Exactly one</i>	1		Sebuah <i>Departemen</i> memiliki satu dan hanya satu <i>Atasan</i>
<i>Zero or more</i>	0..*		Seorang <i>Karyawan</i> dapat tidak memiliki atau memiliki banyak <i>Anak</i>
<i>One or more</i>	1..*		Seorang <i>Atasan</i> bertanggung jawab terhadap satu atau lebih <i>Karyawan</i>
<i>Zero or one</i>	0..1		Seorang <i>Karyawan</i> dapat tidak menikah atau menikah hanya dengan satu <i>Pasangan</i>
<i>Specified range</i>	2..4		Seorang karyawan dapat mengambil antara dua sampai empat kali liburan setiap tahun
<i>Multiple, disjoint ranges</i>	1..3, 5		Seorang <i>Karyawan</i> dapat menjadi anggota dari satu sampai tiga atau lima <i>Kepanitiaan</i>

Gambar 6.11  
Multiplisitas

Ada kalanya asosiasi itu sendiri memiliki properti terkait, terutama ketika kelasnya berbagi hubungan banyak-ke-banyak. Dalam kasus ini, sebuah kelas dibentuk disebut *association class* yang memiliki atribut dan metode sendiri. Ini sangat mirip dengan entitas persimpangan yang ditempatkan pada ERD. Itu ditampilkan sebagai persegi panjang yang dihubungkan oleh garis putus-putus ke jalur asosiasi. Contoh pada Gambar 6.12 menunjukkan *association class* (Fowler, 2005).



Gambar 6.12  
Contoh Association Class

Gambar 6.12 memperlihatkan seseorang dapat mengikuti banyak pertemuan. Misalkan kita perlu menyimpan informasi tentang seberapa sadar orang tersebut dalam mengikuti pertemuan (diindikasikan dengan keaktifannya selama mengikuti pertemuan), maka dilakukan penambahan atribut *perhatian* pada *association class* *Kehadiran*.

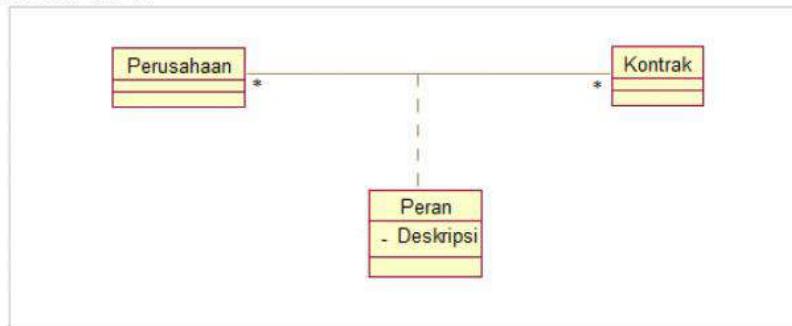
Gambar 6.13 menunjukkan cara lain menyampaikan informasi yang terkandung pada Gambar 6.12 dengan membuat *Kehadiran* menjadi sebuah kelas utuh. Perhatikan bagaimana *multiplicity* berpindah!



Gambar 6.13  
Mengubah Association Class Menjadi Kelas Utuh

Gambar 6.13 memperlihatkan tiga buah kelas, dengan dua buah asosiasi: Seseorang *didata* kehadirannya dalam suatu Pertemuan. Perlu ditegaskan lagi bahwa tidak semua *class* dapat dijadikan *association class*. Hanya *class* yang memiliki

*multiplicity* bernilai banyak yang dapat dijadikan *association class*. Perhatikan contoh pada Gambar 6.14!



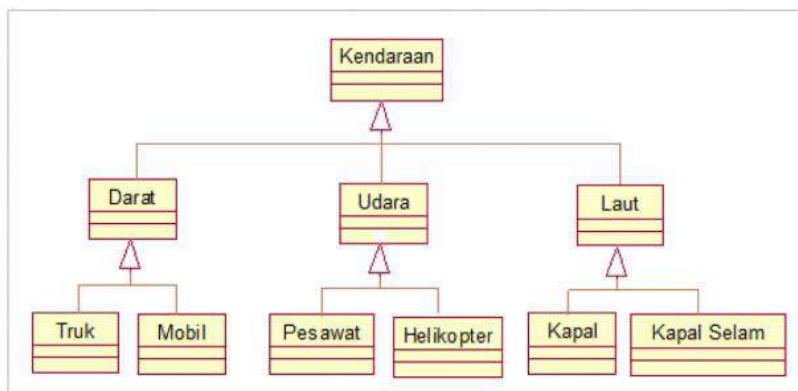
Gambar 6.14  
Keluwasan Association Class (Peran Tidak Dapat Menjadi Association Class)

Pada Gambar 6.14, misalkan tidak memungkinkan sebuah *Perusahaan* untuk memiliki lebih dari satu *Peran* dalam sebuah *Kontrak* tunggal, maka *Peran* tidak dapat dijadikan *association class*, melainkan perlu diubah menjadi sebuah *class* utuh.

Seringkali, kelas terkait melalui asosiasi "normal", tetapi ada dua kasus khusus dari asosiasi yang muncul cukup sering: *generalisasi* dan *agregasi*.

## 2. Generalization dan Aggregation

*Generalization* (generalisasi) menunjukkan bahwa satu kelas (*subclass*) mewarisi kelas lain (*superclass*), yang berarti bahwa properti dan operasi *superclass* juga berlaku untuk objek-objek dari *subclass*. Jalur generalisasi ditunjukkan dengan garis padat dari *subclass* ke *superclass* dan panah berongga menunjuk ke *superclass*. Sebagai contoh, pada Gambar 6.9 *mekanik toko* dan *tenaga penjualan/pramuniaga* adalah jenis *karyawan*, dan *karyawan* serta *pelanggan* adalah jenis *orang*. Contoh lain sebuah asosiasi generalisasi disajikan pada Gambar 6.15.



Gambar 6.15  
Contoh Asosiasi Generalisasi

Agregasi digunakan ketika kelas sebenarnya terdiri dari kelas lain. Misalnya, konsep tentang dealer kendaraan yang telah memutuskan untuk membuat *Tim Layanan Pelanggan* yang mencakup *Mekanik Toko* dan *Tenaga Penjualan*. Ketika pelanggan mengunjungi dealer, mereka diarahkan ke *Tim Layanan Pelanggan* yang memperhatikan kebutuhan mereka selama kunjungan mereka. Gambar 6.9 menunjukkan bagaimana hubungan ini dilambangkan pada diagram kelas. Sebuah simbol "berlian" ditempatkan paling dekat dengan kelas yang mewakili agregasi (*Tim Layanan Pelanggan*), dan garis ditarik untuk menghubungkan kelas yang berfungsi sebagai bagian-bagiannya (*Mekanik Toko* dan *Tenaga Penjualan/Pramuniaga*). Asosiasi agregasi biasanya diidentifikasi ketika kita perlu menggunakan kata-kata seperti "adalah bagian dari" atau "terdiri dari" untuk menggambarkan hubungan. Contoh lain asosiasi agregasi disajikan pada Gambar 6.16.



Gambar 6.16  
Contoh Asosiasi Agregasi

### 3. Penyederhanaan *Class Diagram*

Ketika *class diagram* digambar dengan menyertakan semua kelas dan asosiasi untuk sistem dunia nyata, *class diagram* bisa menjadi sangat kompleks. Ketika ini terjadi, kadang-kadang perlu untuk melakukan penyederhanaan (*simplifying*) diagram dengan menggunakan tampilan untuk membatasi jumlah informasi yang ditampilkan (Dennis, 2012). Tampilan hanya memuat himpunan bagian dari informasi yang terkandung dalam seluruh model. Seperti pada tampilan *use case* yang hanya menunjukkan kelas dan hubungan yang diperlukan untuk *use case* tertentu, pada *class diagram*, penyajian cukup dapat menunjukkan jenis hubungan tertentu (seperti agregasi, asosiasi, atau generalisasi) dan hanya yang terkait dengan kelas tertentu (seperti nama *class*, atribut, dan/atau metode). Menampilkan hanya ketiga jenis tersebut dapat membatasi informasi yang ditampilkan.

Pendekatan kedua untuk menyederhanakan *class diagram* adalah melalui penggunaan paket/*package* (misalnya dengan mendesain kelompok kelas yang logis). Untuk membuat diagram lebih mudah dibaca dan menjaga model pada tingkat kompleksitas yang masih logis, kita dapat mengelompokkan kelas terkait menjadi satu paket. *Package diagram* (diagram paket) adalah konstruksi umum yang menjadi bagian

yang dapat diterapkan ke salah satu elemen dalam model UML/*structure diagram* (tidak dibahas di dalam buku ini).

#### 4. Membuat *Class Diagram*

Membuat *class diagram* adalah proses berulang di mana analis memulai dengan menggambar versi kasar diagram, kemudian memperbaikinya seiring waktu. Berikut akan disajikan proses pembuatan *class diagram* melalui satu iterasi pembuatan, namun contoh diagram yang disajikan ini mungkin akan berubah secara dramatis ketika analis berkomunikasi dengan pengguna riil sistem untuk menyempurnakan pemahaman mereka tentang sistem ini.

##### a. Mengidentifikasi Kelas

Langkah-langkah untuk membuat *class diagram* sangat mirip dengan langkah-langkah untuk membuat ERD. Pertama, perlu mengidentifikasi kelas apa yang harus ditempatkan pada diagram (seperti ERD, *class diagram* menyajikan kelas yang diperlukan untuk sistem secara keseluruhan). Namun, untuk tujuan demonstrasi, bagian ini hanya menyajikan contoh *class diagram* untuk satu *use case*: "memilih lagu".

Banyak pendekatan berbeda telah disarankan untuk membantu analis dalam mengidentifikasi sekumpulan kandidat kelas untuk diagram kelas. Pendekatan yang paling umum adalah analisis tekstual dan analisis teks dalam *use case*. Analis mulai dengan meninjau *use case* dan *use case diagram*. Deskripsi teks dalam *use case* diperiksa untuk mengidentifikasi objek potensial, atribut, metode, dan asosiasi. *Kata benda* dalam *use case* mengisyaratkan kemungkinan *kelas*, sementara *kata kerja* mengisyaratkan kemungkinan *operasi* atau *asosiasi*. Gambar 6.17 menyajikan ringkasan pedoman untuk mengidentifikasi kandidat kelas (Dennis, 2012).

Membuat *class diagram* juga dapat dilakukan berdasarkan kartu CRC (*Class-Responsibility-Collaboration*). Ini sama dengan membuat *use-case diagram* dari *deskripsi use case*. Informasi yang terkandung pada kartu CRC ditransfer ke diagram kelas (lihat Dennis, 2009, p. 222).

##### b. Mengidentifikasi Atribut dan Operasi

Langkah selanjutnya adalah menentukan jenis informasi yang ingin ditangkap pada setiap kelas. *Use case description* memberikan wawasan tentang jenis informasi yang perlu ditangkap (di bagian bawah berlabel "Detail informasi untuk setiap langkah"). Terkadang pada fase pengumpulan kebutuhan juga dibutuhkan informasi tambahan. Pada titik ini, analis membuat daftar beberapa informasi yang mungkin ingin ditangkap untuk keperluan setiap kelas.

Misalkan dari hasil analisis tekstual terhadap *use case diagram* dan *use case description*, serta menggunakan petunjuk pada Gambar 6.17, diperoleh kandidat kelas beserta jenis informasi yang ingin ditangkap pada setiap kelas untuk *class diagram* "Memilih Lagu" berikut ini (Gambar 6.18). Pada titik ini, selanjutnya kita dapat

mempertimbangkan *operasi* apa yang perlu dimiliki setiap kelas (perlu diingat bahwa semua kelas dapat melakukan operasi dasar seperti "memasukkan pelanggan baru", "memasukkan lagu baru", atau operasi yang lainnya). Gambar 6.18 menunjukkan "model awal" atribut untuk diagram kelas. Kita juga dapat mempertimbangkan kemungkinan adanya atribut dan atau operasi lain yang bisa disertakan dalam kelas-kelas ini.

Pedoman	Contoh
<b>Kata benda:</b> menyiratkan objek atau kelas. <ul style="list-style-type: none"> <li>- Kata benda umum (<i>common noun</i>) atau <i>improper noun</i> menyiratkan sebuah kelas dari objek.</li> <li>- Kata benda atau referensi langsung, menyiratkan sebuah instance dari sebuah kelas.</li> <li>- Kata benda kolektif menyiratkan kelas objek yang terdiri dari grup instance kelas lain.</li> </ul>	<ul style="list-style-type: none"> <li>- "Seorang karyawan melayani pelanggan" menyiratkan dua kelas objek, <i>karyawan</i> dan <i>pelanggan</i>.</li> <li>- "John membahas masalah yang diangkat oleh Susan" menyiratkan dua contoh objek, <i>John</i> dan <i>Susan</i>.</li> <li>- "Daftar siswa tidak terverifikasi" mengandung arti bahwa <i>daftar siswa</i> adalah suatu <i>benda</i> yang mempunyai <i>atribut</i> dan metodenya sendiri.</li> </ul>
<b>Kata kerja:</b> menyiratkan asosiasi atau operasi. <ul style="list-style-type: none"> <li>- Kata kerja doing menyiratkan sebuah operasi.</li> <li>- Kata kerja being menyiratkan asosiasi klasifikasi antara suatu objek dan kelasnya</li> <li>- Kata kerja memiliki menyiratkan hubungan agregasi atau asosiasi</li> <li>- Kata kerja transitif menyiratkan operasi.</li> <li>- Sebuah predikat atau frasa kata kerja deskriptif menyiratkan operasi.</li> </ul>	<ul style="list-style-type: none"> <li>- "Doni mengajukan pesanan pembelian" menyiratkan operasi "file"</li> <li>- "Joe adalah seekor anjing" menyiratkan bahwa Joe adalah <i>contoh</i> dari <i>kelas anjing</i>.</li> <li>- "Mobil memiliki mesin" menyiratkan asosiasi agregasi antara <i>mobil</i> dan <i>mesin</i>.</li> <li>- "Frank mengirim Donna pesanan" menyiratkan bahwa <i>Frank</i> dan <i>Donna</i> adalah contoh dari <i>beberapa kelas</i> yang memiliki operasi terkait dengan pengiriman pesanan</li> <li>- "Jika dua karyawan berada di departemen yang berbeda, maka ..." menyiratkan <i>operasi</i> untuk menguji apakah karyawan berada di departemen yang berbeda atau tidak.</li> </ul>
<b>Kata sifat:</b> menyiratkan atribut kelas <ul style="list-style-type: none"> <li>- Kata sifat menyiratkan atribut dari suatu objek.</li> </ul>	

Pedoman	Contoh
	- "Semua pelanggan berusia 55 tahun sekarang berhak atas diskon senior" menyiratkan bahwa <i>usia</i> adalah <i>atribut</i> .
<b>Kata keterangan:</b> menyiratkan atribut asosiasi atau operasi - Kata keterangan menyiratkan atribut asosiasi atau atribut operasi.	- "John mengemudi dengan sangat cepat" menyiratkan <i>atribut kecepatan</i> yang terkait dengan <i>operasi mengemudi</i> .

**Gambar 6.17**  
Pedoman Analisis Tekstual untuk Membuat *Class Diagram*

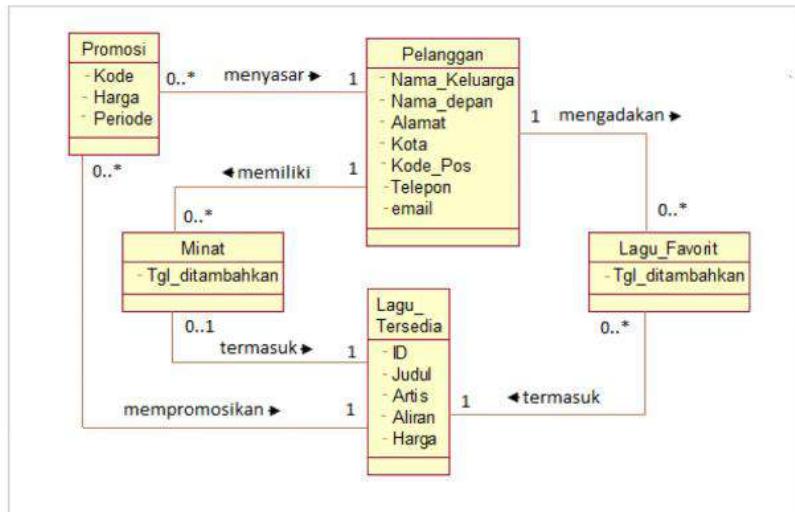


**Gambar 6.18**  
Kandidat Kelas dan Atribut Awal *Class Diagram* "Memilih Lagu"

c. *Menggambarkan Asosiasi Antar Kelas*

Asosiasi ditambahkan ke *class diagram* dengan menggambar garis asosiasi. Telusuri setiap kelas dan tentukan kelas lain yang terkait, nama asosiasi (atau peran yang dimainkannya), dan jumlah instance yang dapat berpartisipasi dalam asosiasi tersebut. Misalnya, kelas *promosi* dikaitkan dengan kelas *pelanggan* karena *promosi* menyasar/menargetkan *pelanggan*. *Pelanggan* dapat disasar tanpa melakukan *promosi* (nol promosi) atau dengan banyak *promosi* (multiplisitas 0 .. \*), dan *promosi* dapat dikaitkan dengan satu dan hanya satu *pelanggan* (multiplisitas = 1). Selanjutnya, merumuskan asosiasi untuk *Minat*, *Lagu Favorit*, dan *Lagu Tersedia*. Gambar 6.19 menunjukkan contoh diagram yang dapat dihasilkan. Pada Gambar 6.19, kita dapat menyertakan hubungan antara *Pelanggan* dan *Lagu Favorit*, *Pelanggan* dan *Minat*, *Lagu Favorit* dan *Lagu yang Tersedia*, *Minat* dan *Lagu yang Tersedia*, serta *Promosi* dan *Lagu yang Tersedia*. Kita juga dapat memeriksa kemungkinan peluang untuk menggunakan asosiasi agregasi atau generalisasi (misalnya pada kasus keterkaitan

antara *Lagu Favorit* dan *Lagu yang Tersedia*, atau antara *Minat* dan *Lagu yang Tersedia*.



Gambar 6.19

Kelas-kelas yang Dihubungkan dengan Asosiasi pada *Class Diagram* “Memilih Lagu”

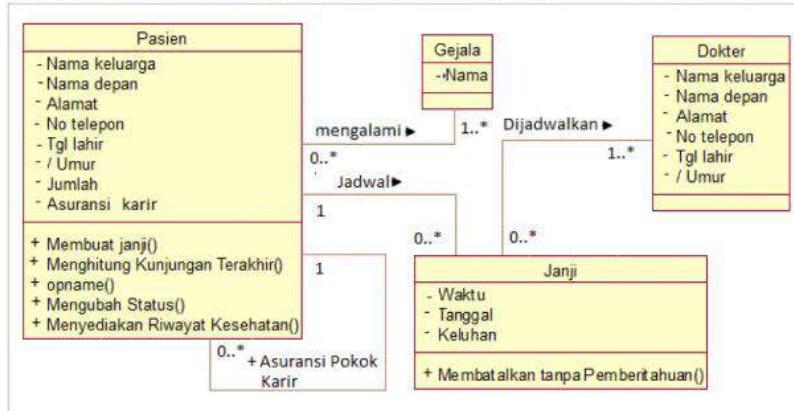
## B. OBJECT DIAGRAM

Meskipun *class diagram* dinyatakan telah memenuhi untuk mendokumentasikan struktur kelas, ada kalanya jenis kedua dari *structural diagram*, yang disebut *object diagram* (diagram objek), dapat bermanfaat. *Object diagram* pada dasarnya adalah instantiasi dari semua atau bagian dari diagram kelas. Instansiasi berarti membuat instance kelas dengan satu set nilai atribut yang sesuai (Dennis, 2009).

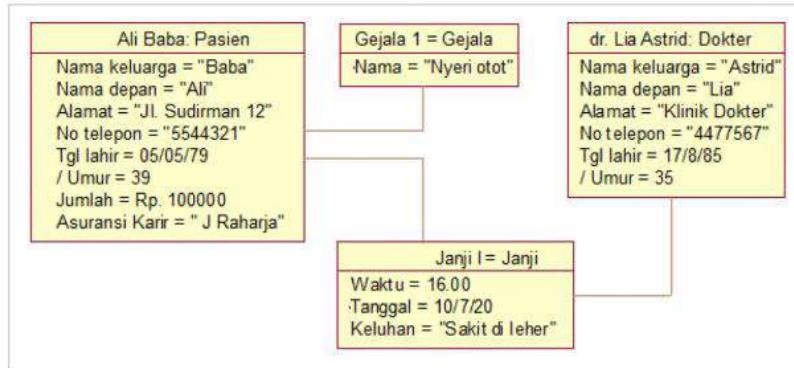
*Object diagram* dapat digunakan untuk menunjukkan sebuah konfigurasi contoh dari objek (Fowler, 2005). *Object diagram* dapat sangat berguna ketika mencoba mengungkap detail kelas. *Object diagram* juga sangat berguna sebagai alat bantu analisis pada saat terdapat kerumitan dalam koneksi yang mungkin terjadi antar objek.

Secara umum, lebih mudah untuk menganalisis sesuatu dengan menyajikan objek konkret (instance) daripada objek abstrak (kelas). Sebagai contoh pada Gambar 6.20. Bagian (a) dari Gambar 6.20 merupakan penggalan kecil sebuah *class diagram* “Sistem Janji Temu Dokter” secara keseluruhan. Bagian (b) adalah *object diagram* hasil transformasi dari *class diagram* itu. Dengan meninjau contoh aktual yang terlibat: Ali Baba, Gejala1, dan dr. Lia Astrid, kita dapat menemukan atribut tambahan yang mungkin relevan, hubungan, dan operasi atau mungkin atribut yang salah tempat. Misalnya, *Janji Temu* memuat atribut *Keluhan*. Setelah pemeriksaan lebih dekat, atribut *Keluhan* mungkin lebih baik dimodelkan sebagai bagian yang berhubungan dengan kelas *Gejala*. Saat ini, kelas *Gejala* dikaitkan dengan kelas *Pasien*. Setelah meninjau

*object diagram*, ini tampaknya salah. Dengan demikian, kita harus memodifikasi diagram kelas untuk mencerminkan pemahaman baru tentang masalah ini.



*Class Diagram*



*Object Diagram*

Gambar 6.20  
Transformasi Class Diagram Menjadi Object Diagram



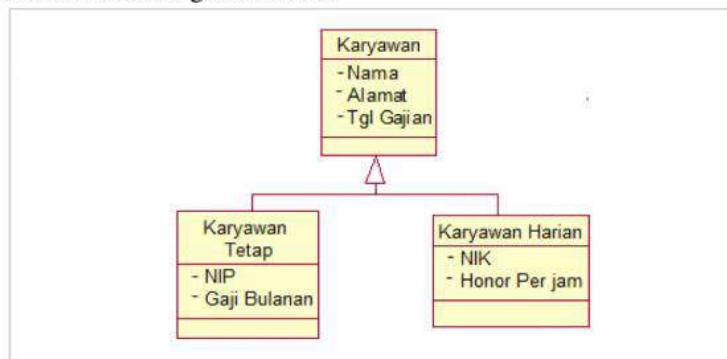
### Latihan

Untuk memperdalam pemahaman Anda mengenai materi di atas, kerjakanlah latihan berikut!

- 1) Jelaskan model struktural, serta kegunaannya!
- 2) Jelaskan kegunaan *class diagram*!
- 3) Apa yang dimaksud dengan *class asosiasi*, apa syaratnya?
- 4) Gambarkan sebuah contoh asosiasi generalisasi, lengkapi dengan atribut seperlunya!
- 5) Jelaskan kegunaan *object diagram*!

*Petunjuk Jawaban Latihan*

- 1) Model struktural adalah model konseptual yang secara formal merepresentasikan berbagai jenis objek yang terdapat di dalam sistem bisnis.  
Kegunaannya:
  - a) Untuk menggambarkan orang, tempat, atau hal-hal mengenai informasi yang ditangkap dan bagaimana mereka saling berhubungan.
  - b) Untuk menggambarkan struktur data yang mendukung proses bisnis dalam suatu organisasi. Pada fase analisis, model struktural menyajikan organisasi logis dari data tanpa memperlihatkan bagaimana data disimpan, dibuat, atau dimanipulasi, sehingga analis dapat fokus pada prosedur bisnis, tanpa terganggu oleh detail teknisnya. Pada fase desain, model struktural diperbarui untuk mencerminkan dengan tepat bagaimana data akan disimpan dalam database dan file.
- 2) *Class diagram* mendeskripsikan jenis-jenis objek dalam sistem dan berbagai macam hubungan statis yang terdapat di antara objek-objek yang ada. *Class diagram* juga menunjukkan properti dan operasi sebuah kelas, serta batasan-batasan yang terdapat dalam hubungan-hubungan objek tersebut.
- 3) Asosiasi yang mengandung informasi, yang dalam hal ini memiliki atribut dan metode sendiri, disebut class asosiasi. Persyaratan terbentuknya class asosiasi adalah kelas-kelas yang saling berhubungan memiliki jenis hubungan banyak-ke-banyak
- 4) Contoh asosiasi generalisasi:



Asosiasi generalisasi menunjukkan bahwa satu kelas (subclass) mewarisi kelas lain (superclass), yang berarti bahwa properti dan operasi superclass juga berlaku untuk objek-objek dari subclass. Secara eksplisit dapat dikatakan bahwa semua atribut pada class *Karyawan* (*Nama*, *Alamat*, *Tgl Gajian*) juga merupakan atribut pada class *Karyawan Tetap* dan class *Karyawan Harian*, sehingga class

*Karyawan Tetap* dan class *Karyawan Harian* harus memiliki tidak hanya nilai atribut pribadinya, melainkan juga nilai-nilai atribut yang merupakan atribut warisan dari class *Karyawan*.

- 5) *Object diagram* digunakan sebagai alat bantu analisis untuk mendukung *class diagram* dalam mendokumentasikan struktur kelas, terutama pada saat terdapat kerumitan dalam koneksi yang mungkin terjadi antar objek dalam kelas. Dalam hal ini, *object diagram* digunakan untuk menyajikan sebuah konfigurasi contoh dari objek secara konkret. Daripada membayangkan sesuatu secara abstrak (kelas), lebih mudah menganalisis suatu objek yang tampak secara konkret (instance).



## Rangkuman

Model struktural menggambarkan struktur data yang mendasari sistem berorientasi objek. Model struktural memberikan pandangan statis internal dari sistem yang berkembang (misalnya bagaimana objek disusun dalam sistem). Pada titik ini dalam pengembangan sistem, model struktural hanya mewakili model logis dari domain masalah yang mendasarinya. Salah satu tujuan utama dari model struktural adalah untuk menciptakan kosakata yang memungkinkan pengguna dan pengembang untuk berkomunikasi secara efektif tentang masalah yang sedang diselidiki. Model struktural biasanya direpresentasikan oleh *class diagram*, yang dalam beberapa kasus disempurnakan menggunakan *object diagram*.

*Class diagram* menunjukkan kelas dan hubungan (asosiasi) antar kelas yang tetap konstan dalam sistem, seiring waktu. Blok bangunan utama *class diagram* adalah kelas, yang menyimpan dan mengelola informasi dalam sistem. Kelas memiliki atribut yang menangkap informasi tentang kelas dan tentang operasi, yang merupakan tindakan yang dapat dilakukan kelas. Ada tiga jenis operasi: konstruktor, kueri, dan pembaruan. Kelas terkait satu sama lain melalui asosiasi, yang memiliki nama dan multiplisitas yang menunjukkan contoh minimum dan maksimum yang berpartisipasi dalam hubungan. Ada kalanya hubungan itu sendiri mengandung informasi, yang disebut *class* asosiasi. Terdapat tiga tipe dasar hubungan yang biasanya digambarkan pada model struktural (agregasi, generalisasi, dan asosiasi) yang terkandung dalam diagram. Dua asosiasi khusus, agregasi dan generalisasi, digunakan ketika kelas terdiri dari kelas lain atau ketika satu subclass mewarisi sifat dan perilaku dari superclass, masing-masing.

*Class diagram* dibuat dengan mengidentifikasi kelas pertama, bersama dengan atribut dan operasi mereka. Kemudian hubungan ditarik di antara kelas untuk menunjukkan asosiasi. Selanjutnya, notasi khusus digunakan untuk menggambarkan asosiasi agregasi dan generalisasi.

Dalam sistem dunia nyata yang dapat memiliki begitu banyak kelas, *class diagram* dapat menjadi terlalu rumit digambarkan. Untuk penyederhanaan diagram, mekanisme pembatasan tampilan dapat digunakan. Tampilan membatasi jumlah informasi yang digambarkan pada diagram. Beberapa pandangan yang bermanfaat adalah: menyembunyikan semua informasi tentang kelas kecuali namanya dan

hubungannya; hanya menunjukkan kelas yang terkait dengan *use case* tertentu; membatasi hubungan hanya pada satu tipe spesifik (agregasi, generalisasi, dan asosiasi); dan tidak menggambarkan model untuk semua hal, melainkan hanya berkonsentrasi pada hal kunci.

Kesulitan menggambarkan *class diagram* karena *class diagram* sangatlah "kaya", sehingga orang-orang sering kewalahan untuk menggunakan semua notasi dan aturan penggambaran. Oleh karena itu disarankan untuk tidak mencoba menggunakan seluruh notasi yang tersedia, melainkan menggunakan notasi-notasi yang sederhana: *class*, asosiasi, atribut, generalisasi dan batasan-batasan.

Ketika mencoba mengungkap informasi tambahan tentang detail kelas, akan berguna untuk menggambarkan instance kelas tertentu daripada kelas itu sendiri. *Object diagram* digunakan untuk menggambarkan satu set objek yang mewakili semua contoh atau bagian dari *class diagram*.



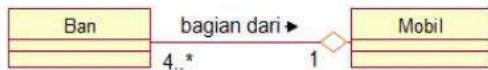
### Tes Formatif 2

---

Pilihlah satu jawaban yang paling tepat!

- 1) Tools UML (versi 2.x) yang dapat digunakan pada pemodelan struktural sistem adalah ....
  - A. *Use Case Diagram*
  - B. *Activity Diagram*
  - C. *Use Case Description* dan *Class Diagram*
  - D. *Class Diagram* dan *Object Diagram*
  
- 2) Elemen-elemen dasar dari *Class Diagram*, *kecuali* ....
  - A. *Class*
  - B. *Actor*
  - C. *Attribute*
  - D. *Method*
  
- 3) Berikut merupakan jenis-jenis hubungan/relasi antar kelas pada *class diagram*, *kecuali* ....
  - A. Asosiasi (normal)
  - B. Agregasi
  - C. *Private*
  - D. Pewarisan (generalisasi)
  
- 4) Berikut adalah jenis operasi/*method* yang terkandung dalam kelas ....
  - A. *Constructor*
  - B. Operasi khusus dan *query*
  - C. *Update*
  - D. jawaban A dan C benar

- 5) Jenis operasi/*method* yang terkandung dalam kelas, yang dapat melakukan fungsi penyediaan informasi mengenai keadaan suatu objek untuk objek lain adalah ....
- Query*
  - Operasi khusus
  - Constructor*
  - Update*
- 6) Tanda (-) yang terletak di awal penulisan *attribut* atau *method* pada kelas, biasanya menunjukkan ....
- Visibilitas *Public*
  - Visibilitas *Protected*
  - Visibilitas *Private*
  - Tidak memiliki makna apapun
- 7) Visibilitas default untuk atribut pada kelas, pada umumnya bersifat ....
- Public*
  - Private*
  - Protected*
  - Atribut pada kelas tidak memiliki visibilitas default
- 8) Gambar di bawah ini menunjukkan relasi ....



- Asosiasi (Normal)
  - Generalisasi
  - Agregasi
  - Subclass*
- 9) Pada pemodelan berorientasi objek, *class diagram* dapat digunakan untuk menggantikan fungsi ERD/Diagram Relasi Database, untuk menggambarkan Struktur Database Sistem. Pernyataan tersebut ....
- Benar
  - Salah
- 10) *Class diagram* dinyatakan telah memenuhi untuk mendokumentasikan struktur kelas, tanpa perlu didukung oleh tool UML lainnya. Pernyataan tersebut ....
- Benar
  - Salah

Cocokkanlah jawaban Anda dengan Kunci Jawaban Tes Formatif 2 yang terdapat di bagian akhir modul ini. Hitunglah jawaban yang benar. Kemudian, gunakan rumus berikut untuk mengetahui tingkat penguasaan Anda terhadap materi Kegiatan Belajar 2.

$$\text{Tingkat Penguasaan} = \frac{\text{Jumlah Jawaban yang Benar}}{\text{Jumlah Soal}} \times 100$$

Arti tingkat penguasaan

<70%	70% - 79%	80% - 89%	90% - 100%
kurang	cukup	baik	baik sekali

Apabila mencapai tingkat penguasaan 80% atau lebih, Anda dapat meneruskan dengan modul selanjutnya. **Bagus!** Jika masih di bawah 80%, Anda harus mengulangi materi Kegiatan Belajar 2, terutama bagian yang belum dikuasai.

## Kunci Jawaban Tes Formatif

### *Tes Formatif 1*

- 1) A
- 2) C
- 3) B
- 4) B
- 5) C
- 6) D
- 7) B
- 8) A
- 9) B
- 10) B

### *Tes Formatif 2*

- 1) D
- 2) B
- 3) C
- 4) D
- 5) A
- 6) C
- 7) A
- 8) C
- 9) A
- 10) B

## Glosarium

Asosiasi	: Suatu hubungan/relasi yang mempunyai makna di antara sejumlah objek, yang digambarkan dengan sebuah garis penghubung di antara objeknya
Behavior	: Kumpulan dari sesuatu yang dapat dilakukan oleh objek dan terkait dengan fungsi-fungsi yang bertindak pada data objek (atau atribut). Pada siklus berorientasi objek, perilaku (behavior) objek merujuk kepada metode, operasi, atau fungsi
Fase	: Tahapan utama dalam analisis dan perancangan sistem informasi (misalnya: perencanaan, analisis sistem, perancangan sistem, implementasi sistem, dan seterusnya)
Fitur	: Properti dan operasi pada sebuah kelas
Instance	: Objek yang riil
Multiplicity	: Jumlah kejadian minimum dan maksimum dari satu objek/kelas untuk suatu kejadian tunggal dari objek atau kelas terkait
Node	: Titik yang menghubungkan suatu objek dalam penggambaran activity diagram
Notasi	: Seperangkat lambang (tanda) yang menggambarkan suatu objek dalam Unified Modeling Language (UML)
OMG (Object Management Group)	: Konsorsium yang berusaha membuat standardisasi metode pengembangan perangkat lunak berorientasi objek
Visibility	: Level akses yang dimiliki oleh sebuah objek eksternal pada sebuah atribut atau method suatu kelas

## Daftar Pustaka

Dennis, A., Wixom, B. H., & Tegarden, D. (2009). *Systems analysis & design with uml version 2.0*, Third edition. United States of America: John Wiley & Sons, Inc.

Dennis, A., Wixom B. H., & Roth R.M. (2012). *Systems analysis & design*, 5th Edition. United States of America: John Wiley & Sons, Inc.

Fowler, M. (2005). *UML distilled*. Edisi 3. Yogyakarta: ANDI.

Nugroho, A. (2005). *Rational rose untuk pemodelan berorientasi objek*. Bandung: Informatika.

**MSIM4302**  
**Edisi 1**

**MODUL 07**

# **Pemodelan Perilaku (*Behavioral*) Objek**

Bahar, S.T., M.Kom.

## Daftar Isi

### Modul 07

7.1

Pemodelan Perilaku (*Behavioral*)  
Objek

**Kegiatan Belajar 1**  
Diagram Interaksi  
(*Sequence Diagram* dan  
*Communication Diagram*)

7.5

**Latihan**  
**Rangkuman**  
**Tes Formatif 1**

7.20

7.22

7.23

**Kegiatan Belajar 2**  
*Behavioral State Machines*  
*Diagram*

7.26

**Latihan**  
**Rangkuman**  
**Tes Formatif 2**

7.33

7.35

7.35

**Kunci Jawaban Tes Formatif**

7.38

**Glosarium**

7.39

**Daftar Pustaka**

7.41



## Pendahuluan

**A**nalis sistem menggunakan model fungsional untuk menggambarkan pandangan perilaku eksternal dari suatu sistem informasi, sedangkan model struktural untuk menggambarkan pandangan statis dari suatu sistem informasi. Hal ini telah dibahas pada modul-modul sebelumnya. Dalam modul ini, akan dibahas bagaimana analis menggunakan model perilaku (*behavioral model*) untuk merepresentasikan perilaku internal atau tampilan dinamis dari suatu sistem informasi.

Model perilaku menggambarkan aspek dinamis internal dari suatu sistem informasi yang mendukung proses bisnis dalam suatu organisasi. Selama analisis, model perilaku mendeskripsikan logika internal dari proses tanpa menentukan bagaimana proses akan diimplementasikan. Kemudian, dalam tahap desain dan implementasi, desain rinci dari operasi yang terdapat dalam objek ditentukan sepenuhnya.

Ada dua jenis model perilaku. Pertama, adalah model perilaku yang digunakan untuk merepresentasikan detail yang mendasari proses bisnis yang digambarkan oleh model *use case*. Dalam UML (*Unified Modeling Language*), *interaction diagram* (*sequence diagram* dan *communication diagram*) digunakan untuk jenis model perilaku ini. Kedua, adalah model perilaku yang digunakan untuk merepresentasikan perubahan yang terjadi pada data yang mendasarinya. UML menggunakan *behavioral state machines* untuk ini (Dennis, 2009).

Selama fase analisis, analis menggunakan model perilaku untuk menangkap pemahaman dasar tentang aspek dinamis dari proses bisnis yang mendasarinya. Secara tradisional, model perilaku telah digunakan terutama selama fase desain, di mana analis menyempurnakan model perilaku untuk memasukkan rincian implementasi. Modul ini berfokus pada pandangan dinamis dari sistem yang dikembangkan dan bukan pada bagaimana aspek dinamis sistem akan diterapkan.

Ketika seorang analis mencoba untuk memahami domain aplikasi yang mendasari suatu masalah, dia harus mempertimbangkan aspek struktural dan perilaku dari masalah tersebut. Tidak seperti pendekatan lain untuk pengembangan sistem informasi, pendekatan berorientasi objek mencoba untuk melihat domain aplikasi yang mendasari secara holistik. Dengan melihat domain masalah sebagai sekumpulan *use case* yang didukung oleh sekumpulan objek yang berkolaborasi, pendekatan berorientasi objek memungkinkan seorang analis untuk meminimalkan celah semantik antara kumpulan objek dunia nyata dan model berorientasi objek yang dikembangkan dari domain masalah. Namun demikian, seperti yang telah dijelaskan sebelumnya, dunia nyata cenderung tidak beraturan, yang membuat pemodelan domain aplikasi secara praktis tidak mungkin dilakukan dalam perangkat lunak. Ini karena perangkat lunak harus beraturan dan logis dalam operasinya.

#### 7.4 Pemodelan Perilaku (*Behavioral*) Objek

Salah satu tujuan dari model perilaku adalah untuk menunjukkan bagaimana objek yang mendasari domain masalah akan bekerja sama secara berkolaborasi mendukung setiap *use case*. Model struktural mewakili objek dan hubungan di antara mereka, sedangkan model perilaku menggambarkan tampilan internal proses bisnis yang dijelaskan oleh *use case*. Proses tersebut dapat ditunjukkan oleh interaksi yang terjadi antara objek yang berkolaborasi untuk mendukung *use case* melalui penggunaan diagram interaksi (*sequence diagram* dan *communication diagram*). Dimungkinkan juga untuk menunjukkan efek yang dimiliki kumpulan *use case* yang membentuk sistem pada objek dalam sistem melalui *behavioral state machines diagram*.

Membuat model perilaku adalah proses berulang yang tidak hanya mengulangi model perilaku individu, seperti diagram interaksi (*sequence diagram* dan *communication diagram*) dan *behavioral state machines diagram*, tetapi juga melalui model fungsional dan struktural. Saat model perilaku dibuat, tidak jarang melakukan perubahan pada model fungsional dan struktural.

Pembahasan dalam modul ini difokuskan pada pembuatan model perilaku dari proses bisnis yang mendasarinya. Dengan menggunakan *interaction diagram* (*sequence diagram* dan *communication diagram*) dan *behavioral state machines diagram*, dimungkinkan untuk memberikan gambaran lengkap tentang aspek dinamis dari sistem informasi bisnis yang dikembangkan. Pertama, menjelaskan model perilaku dan komponennya, selanjutnya menjelaskan masing-masing diagram perilaku, bagaimana diagram dibuat, dan bagaimana diagram perilaku terkait dengan model fungsional dan model struktural.

Setelah mempelajari modul ini mahasiswa diharapkan secara lebih rinci mampu:

1. menjelaskan jenis-jenis model perilaku objek beserta kegunaannya;
2. menjelaskan aturan dan pedoman membuat *sequence diagram*, *communication diagram*, dan *behavioral state machine diagram*;
3. menjelaskan elemen-elemen serta proses pembuatan *sequence diagram*, *communication diagram*, dan *behavioral state machine diagram*;
4. membuat *sequence diagram*, *communication diagram*, dan *behavioral state machine diagram*;
5. menjelaskan hubungan antara model perilaku, model struktural, dan model fungsional.

# Diagram Interaksi (*Sequence Diagram* dan *Communication Diagram*)

**S**alah satu perbedaan antara diagram kelas (*class diagram*) dan diagram interaksi (*interaction diagram*) adalah fokus pemodelan *class diagram* berada pada tingkat kelas, sedangkan *interaction diagram* pada tingkat objek. Pada kegiatan belajar ini, pertama-tama akan diuraikan kembali secara singkat gambaran tentang objek, operasi, dan pesan, selanjutnya membahas dua diagram berbeda (*sequence diagram* dan *communication diagram*) yang dapat digunakan untuk memodelkan interaksi yang terjadi antara objek dalam sistem informasi.

## A. OBJEK, OPERASI, DAN PESAN

Objek adalah instansiasi sebuah kelas, yaitu: orang, tempat, peristiwa, atau hal yang sebenarnya ingin kita tangkap informasinya (Dennis, 2009). Jika kita sedang membuat Sistem Janji Temu di klinik dokter, yang termasuk kelas adalah: *dokter*, *pasien*, dan *janji temu*. Pasien tertentu, seperti "Ridho", "Mirza", dan "Tuty", dianggap sebagai objek (contoh kelas *pasien*).

Setiap objek memiliki atribut yang mendeskripsikan informasi tentang objek tersebut, seperti: *nama pasien*, *tanggal lahir*, *alamat*, dan *nomor telepon*. Setiap objek juga memiliki perilaku. Pada titik ini, perilaku dijelaskan oleh operasi (*operation*). Operasi adalah sebuah tindakan yang dapat dilakukan oleh sebuah objek. Misalnya, objek *janji temu* mungkin dapat "menjadwalkan janji temu baru", "menghapus janji temu", dan "menemukan janji temu berikutnya" yang tersedia. Nantinya, selama tahap pengembangan sistem yang dikembangkan, perilaku akan diimplementasikan sebagai *method*.

Setiap objek juga dapat mengirim dan menerima pesan (*message*). Pesan adalah informasi yang dikirim ke objek untuk memberi tahu objek agar menjalankan salah satu perilakunya. Pada dasarnya *message* adalah suatu fungsi (*function*) atau pemanggilan prosedur dari satu objek ke objek lain, misalnya jika terdapat "pasien baru" di klinik dokter, sistem akan mengirimkan pesan "penyisipan" ke aplikasi. Objek *pasien* akan menerima instruksi (pesan) dan melakukan apa yang perlu dilakukan untuk memasukkan "pasien baru" ke dalam sistem (perilaku/*behavioral*).

## B. SEQUENCE DIAGRAM

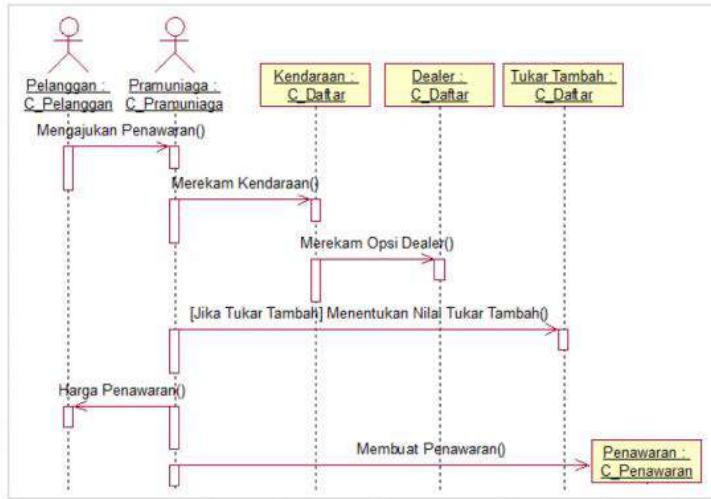
*Sequence diagram* menggambarkan objek yang berpartisipasi dalam *use case* dan pesan yang melewati objek-objek ini dari waktu ke waktu untuk satu *use case* (Fowler, 2005; Dennis, 2012). *Sequence diagram* menunjukkan urutan pesan eksplisit yang diteruskan antara suatu objek dengan objek yang lainnya dalam suatu interaksi yang ditentukan. Karena *sequence diagram* menekankan pengurutan berbasis waktu dari suatu aktivitas yang terjadi di antara sekumpulan objek, diagram tersebut sangat membantu untuk memahami spesifikasi waktu nyata dan *use case* yang kompleks.

*Sequence diagram* dapat berupa *sequence diagram* umum yang menunjukkan semua kemungkinan skenario untuk *use case*, atau analis hanya mengembangkan satu set *sequence diagram* pada lingkup *instance*, yang masing-masing menggambarkan skenario tunggal dalam *use case*. Jika kita tertarik untuk memahami aliran kendali suatu skenario berdasarkan waktu kejadian, kita dapat menggunakan *sequence diagram* untuk menggambarkan informasi ini. Diagram dapat digunakan di seluruh tahap analisis dan desain untuk memodelkan proses sistem; dan digunakan sangat spesifik pada tahap implementasi, yang sering kali menyertakan objek database atau komponen GUI (*Graphical User Interface*) tertentu sebagai kelas (Dennis, 2012). Pembahasan berikut ini pertama-tama menyajikan sintaks dari *sequence diagram* dan kemudian menyajikan cara menggambarnya.

### 1. Element-elemen *Sequence Diagram*

Gambar 7.1 menyajikan contoh *sequence diagram* yang menggambarkan *objek* dan *pesan* untuk *use case* "membuat penawaran", yang menjelaskan proses di mana pelanggan membuat penawaran baru untuk sistem "Sewa Kendaraan".

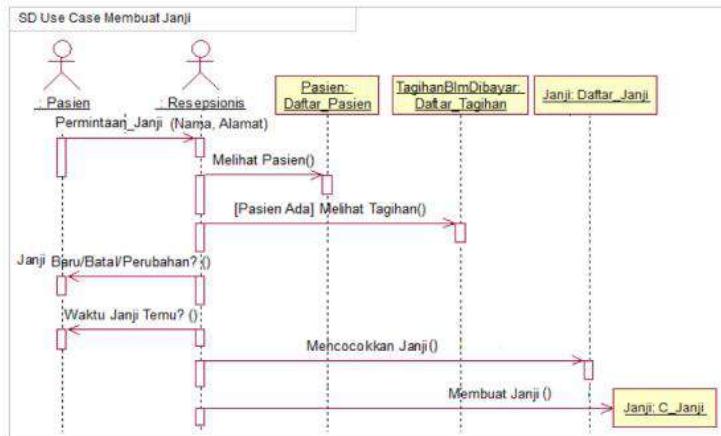
Aktor dan objek yang berpartisipasi dalam *use case* ditempatkan di bagian atas diagram, yang digambarkan dengan simbol *aktor* atau persegi panjang tanpa label (lihat Gambar 7.2.) Perhatikan bahwa objek pada Gambar 7.1 adalah *Pelanggan*, *Pramuniaga*, *Kendaraan*, *Dealer*, *Nilai Tukar Tambah*, dan *Penawaran*. Objek-objek yang ada tidak harus ditempatkan dalam urutan tertentu, meskipun sangat bagus jika mengurnanya sesuai dengan urutan logis proses, seperti urutan partisipasi mereka dalam aktivitas. Untuk setiap objek, nama kelas tempat mereka berada diberikan setelah nama objek (misalnya, *Kendaraan*: *C\_Daftar*, berarti bahwa *Kendaraan* adalah *instance* dari kelas *C\_Daftar* yang berisi objek kendaraan individu).



Gambar 7.1  
Contoh Sequence Diagram untuk Use Case Sistem Bisnis  
"Penawaran Sewa Kendaraan"

Contoh *sequence diagram* lainnya disajikan pada Gambar 7.2, yang menggambarkan objek dan pesan untuk *use case* "Membuat Janji", yang menjelaskan proses di mana pasien membuat janji baru atau membatalkan (menjadwalkan ulang) janji untuk sistem janji pada klinik dokter.

Pada Gambar 7.2, aktor terdiri atas seorang *Pasien* dan seorang *Resepsionis*, sedangkan objek terdiri atas *Pasien*, *UnpaidBills* (tagihan belum dibayar), *Appointment* (janji), dan *anAppt* (sebuah janji). Untuk setiap objek, nama kelas di mana mereka diberikan setelah nama objek (misalnya, *Pasien: Daftar\_Pasien* berarti bahwa *pasien* adalah *instance* (turunan) dari kelas *Daftar\_Pasien* yang berisi objek individu pasien).



Gambar 7.2  
Sequence Diagram untuk Use Case Sistem Bisnis  
"Membuat Janji" Pertemuan pada Klinik Dokter

## 7.8 Pemodelan Perilaku (*Behavioral*) Objek

Garis putus-putus vertikal di bawah setiap aktor dan objek untuk menunjukkan "garis hidup" (*lifeline*) aktor/objek dari waktu ke waktu. Kadang-kadang sebuah objek membuat objek sementara, dan dalam hal ini "X" ditempatkan di ujung garis hidup di titik objek tersebut dihancurkan (tidak ditampilkan lagi).

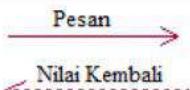
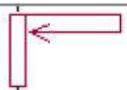
### Contoh

Bayangkan tentang objek "keranjang belanja" pada sebuah aplikasi penjualan berbasis Web. Keranjang belanja digunakan untuk menghimpun sementara item suatu pesanan, tetapi setelah pesanan dikonfirmasi, keranjang belanja tidak lagi diperlukan. Dalam hal ini, "X" akan ditempatkan pada titik di mana objek keranjang belanja telah dihancurkan. Ketika objek terus ada di sistem setelah digunakan dalam *sequence diagram*, garis hidup berlanjut ke bagian bawah diagram (seperti pada Gambar 7.1.)

Kotak persegi panjang tipis, yang disebut *execution occurrence* (kejadian eksekusi), dihamparkan sepanjang garis hidup untuk menunjukkan kapan kelas mengirim dan menerima pesan.

Pesan adalah komunikasi antar objek yang menyampaikan atau memanggil informasi/operasi, dengan ekspektasi bahwa aktivitas akan terjadi. Ada beberapa jenis pesan yang dapat digambarkan pada *sequence diagram*. Tiga jenis pesan biasanya digunakan: panggilan operasi, pengembalian, dan penciptaan objek yang lain.

NOTASI	DEFINISI
 <b>Nama Aktor</b> <div style="border: 1px solid red; padding: 2px; display: inline-block;">Nama Aktor</div>	<b>Aktor (Actor):</b> <ol style="list-style-type: none"> <li>Orang atau sistem yang memperoleh manfaat dan berada di luar sistem.</li> <li>Ditempatkan di bagian atas diagram.</li> <li>Jika melibatkan aktor bukan manusia, digambarkan sebagai persegi panjang.</li> </ol>
<div style="border: 1px solid red; padding: 2px; display: inline-block;">Objek : Kelas</div>	<b>Objek (Object):</b> <ol style="list-style-type: none"> <li>Berpartisipasi secara berurutan dengan mengirim dan/atau menerima pesan.</li> <li>Ditempatkan di bagian atas diagram.</li> </ol>
	<b>Garis Hidup (Lifeline):</b> <ol style="list-style-type: none"> <li>Menunjukkan masa hidup suatu objek selama dalam urutan proses.</li> <li>Berisi "X" pada titik di mana kelas tidak lagi berinteraksi.</li> </ol>
	<b>Penghancuran Objek (Object Destruction):</b> menunjukkan sebuah objek akan menghilang.
	<b>Kejadian Eksekusi (Execution Occurrence):</b>

NOTASI	DEFINISI
	<ol style="list-style-type: none"> <li>Persegi panjang sempit yang ditempatkan di atas garis kehidupan.</li> <li>Menunjukkan saat suatu objek mengirim atau menerima pesan.</li> </ol>
	<p>Pesan (<i>Message</i>) tipe <i>Call</i> dan tipe <i>Create</i>:</p> <ol style="list-style-type: none"> <li>Tipe <i>Call</i>: pengiriman/pemanggilan pesan/operasi oleh satu objek ke objek lainnya.</li> <li>Tipe <i>Create</i>: menciptakan objek yang lain.</li> <li>Panah padat menunjukkan Pesan, sedangkan Nilai Kembali diberi label panah putus-putus.</li> </ol>
	<p>Pesan (<i>Message</i>) pada Diri Sendiri:</p> <p>Objek mengirim/memanggil pesan/operasi yang ada pada dirinya sendiri.</p>

Gambar 7.3  
Notasi pada *Sequence Diagram*

Pesan tipe panggilan (*call*) operasi yang diteruskan antar objek/kelas ditunjukkan dengan garis padat yang menghubungkan dua objek, yang disebut garis tautan (*link*). Panah pada *link* menunjukkan ke arah mana pesan dikirimkan, dan nilai argumen apapun untuk pesan tersebut ditempatkan dalam tanda kurung di sebelah nama pesan. Kadangkala sebuah objek memanggil operasi yang ada pada dirinya sendiri (Rosa, 2011), Fowler (2005) menyebutnya sebagai "panggilan mandiri".

Pesan balasan (nilai kembali) digambarkan sebagai garis putus-putus dengan panah di ujung garis yang menggambarkan arah pengembalian. Namun, karena menambahkan pesan balasan cenderung mengacaukan diagram (kecuali pesan kembali mengandung informasi penting dalam diagram), pesan kembali sering tidak digambarkan (opsional).

Terkadang, suatu objek akan membuat/menciptakan objek lain. Ini ditunjukkan dengan pesan tipe *menciptakan* (*create*) yang dikirim langsung ke suatu objek, bukan ke jalur kehidupannya. Pada Gambar 7.1, objek *Pramuniaga* membuat objek *Penawaran*.

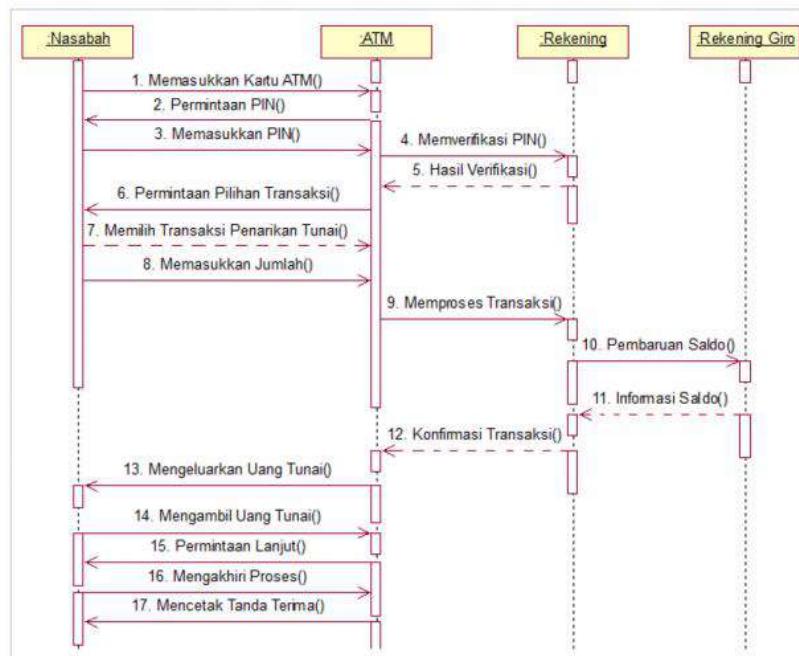
Urutan pesan berpindah dari atas ke bawah halaman, sehingga pesan yang terletak lebih tinggi pada diagram mewakili pesan yang muncul lebih awal dalam urutan, dibanding pesan bawah yang muncul kemudian. Pesan juga dapat diberi nomor untuk memberlakukan urutan. Pada Gambar 7.1, "Merekam Kendaraan" adalah pesan yang dikirimkan dari objek *Pramuniaga* ke objek *Kendaraan*, yang merupakan media bagi kendaraan yang terdaftar saat ini untuk dicatat informasinya sebagai kendaraan yang ditawarkan.

*Sequence diagram* menggambarkan objek yang berpartisipasi dalam *use case*, interaksi antar objek, serta pesan yang melewati objek-objek ini dari waktu ke waktu

## 7.10 Pemodelan Perilaku (*Behavioral*) Objek

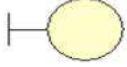
secara berurutan, dan kurang bagus dalam memperlihatkan detail algoritma, seperti *loop* dan behavior kondisional. Ini berbeda dengan *activity diagram* yang menekankan pada logika prosedur proses. Namun, ada kalanya kita ingin mengetahui logika prosedur tertentu dalam suatu proses pada *sequence diagram* (misalnya: pesan dikirim hanya jika kondisi terpenuhi). Dalam kasus tersebut, kondisi ditempatkan di antara kurung siku ([ ]), seperti "[Jika terjadi tukar tambah] Menentukan Nilai Tukar Tambah ()". Kondisi ditempatkan di depan nama pesan (lihat Gambar 7.1).

*Sequence diagram* tidak hanya digunakan untuk menggambarkan interaksi objek/kelas dalam proses bisnis organisasi (seperti pada Gambar 7.1), namun dapat juga digunakan untuk menggambarkan interaksi objek dalam proses bisnis/sistem kerja pada perangkat lunak dan atau perangkat keras, seperti contoh pada Gambar 7.2 dan Gambar 7.4.



Gambar 7.4  
Contoh *Sequence Diagram* untuk *Use Case*  
"Penarikan Tunai" pada Sistem Mesin ATM

*Sequence diagram* yang menggambarkan interaksi objek/kelas yang menyertakan objek database atau komponen GUI (*Graphical User Interface*) tertentu sebagai kelas pada sebuah program aplikasi (*software*), disajikan pada Gambar 7.5. Kelas-kelas biasanya diidentifikasi berdasarkan jenisnya, yaitu: *boundary class*, *control class*, dan *entity class* (Hermawan, 2004).

		
Notasi <i>Boundary Class</i>	Notasi <i>Control Class</i>	Notasi <i>Entity Class</i>

**Gambar 7.5**  
Notasi Beberapa Jenis Kelas

*Boundary class* adalah kelas yang menghubungkan *user* dengan sistem aplikasi. Oleh karena itu sering disebut sebagai *user interface class*. Biasanya setiap pasangan *actor – use case* memiliki minimal satu buah *boundary class*. Lebih lanjut, *user interface class* sering disamakan dengan *form* yang digunakan sebagai antarmuka antara sistem aplikasi dengan *user*, atau piranti-piranti keras yang menjadi perantara antara pengguna dengan sebuah perangkat komputer (misalnya: layar dan *keyboard* pada *Personal Computer* (PC); layar, *keypad*, slot kartu pada sistem mesin ATM).

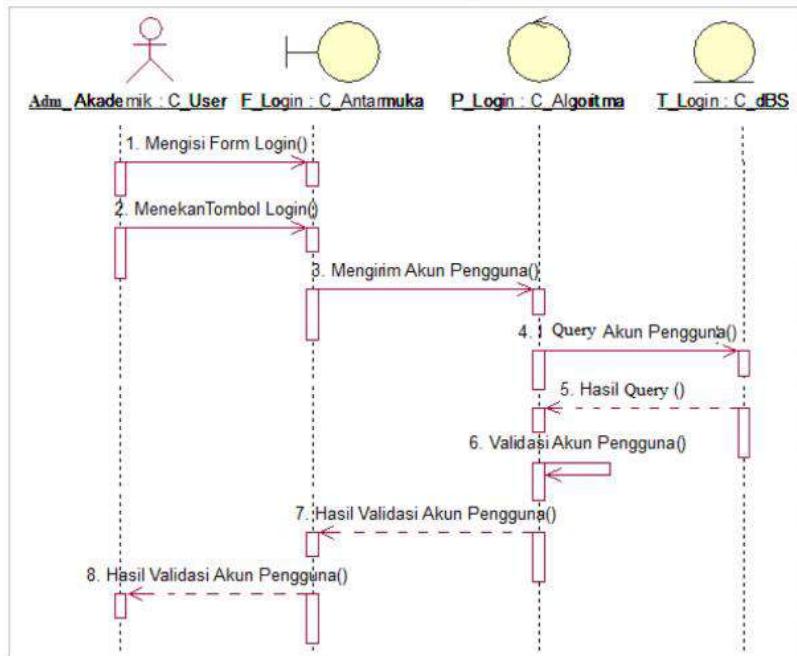
*Control class* adalah kelas yang mengkoordinasi atau mengendalikan aktivitas dalam sistem. Kelas ini menghubungkan *boundary class* dengan *entity class* (mengkoordinasi *entity class* mana yang perlu dikunjungi, kapan dan apa yang ingin didapatkan dari *entity class* tersebut). Masa hidup *control class* adalah dibuat saat *use case* dimulai dan mati saat *use case* selesai dieksekusi. Ini dapat disamakan dengan kegiatan mengeksekusi sebuah sub rutin pada program aplikasi (*software*). Pada umumnya sebuah *use case* (untuk sistem aplikasi) memiliki paling sedikit satu *control class*.

*Entity class* adalah kelas yang berhubungan dengan data atau informasi yang digunakan oleh sistem, yaitu media penyimpanan data yang dapat berupa sebuah database/tabel.

Gambar 7.6 menyajikan contoh *sequence diagram* untuk *use case* "login". Pada contoh ini, sistem login melibatkan satu pengguna (*actor*) yaitu *Operator Administrasi Akademik*, dan tiga buah objek dalam kategori kelas yang berbeda, yaitu: Antarmuka Form Login (*boundary class*) berada pada lapisan terluar sistem aplikasi login, Tabel Login (*entity class*) berada pada lapisan terdalam sistem aplikasi login, dan Prosedur/Method Login (*control class*) sebagai perantara yang mengatur hubungan Antarmuka objek Form Login dengan objek tabel/database Login. Model skenario sederhana yang dibangun pada *use case* "login" ini adalah (pada situasi tertentu algoritma login dapat dibangun lebih kompleks) sebagai berikut.

- Pada lapisan terluar, pengguna (petugas administrasi akademik) mengisi "nama pengguna" dan "kata kunci" pada "form login" (jenis *boundary class/lapis ke-2*), selanjutnya menekan tombol "Login" pada form login.
- Pada saat menekan tombol login, sistem mengirim data "nama user" dan "kata kunci" dan memerintahkan "algoritma login" (jenis *control class/lapis ke-3*) untuk melakukan validasi terhadap "nama pengguna" dan "kata kunci".

## 7.12 Pemodelan Perilaku (*Behavioral*) Objek



Gambar 7.6  
Contoh Sequence Diagram untuk Use Case  
"Login" pada Sebuah Perangkat Lunak Berbasis Desktop (Single User)

- c. "Algoritma Login" melakukan validasi terhadap "nama pengguna" dan "kata kunci" dengan cara (dalam kasus ini, validasi "nama pengguna" dan "kata kunci" dilakukan dengan operator "and", artinya: sistem dinyatakan valid jika keduanya valid. Sistem juga dapat dirancang untuk melakukan proses validasi "nama pengguna" dan "kata kunci" secara berjenjang)
  - 1) Memanggil data (*query*) dari dalam *class* "dbs"/tabel login (jenis *entity class/lapis ke-4*), atau dengan kata lain memerintahkan *entity class* untuk mengirimkan data "nama user" dan "kata kunci" ke dirinya (*class control*). Perhatikan bahwa aliran informasi balik (garis balik putus-putus) merupakan "nilai kembali" dari proses "query akun pengguna".
  - 2) Melakukan/memerintahkan dirinya sendiri (notasi "pesan pada diri sendiri") melakukan validasi "nama user" dan "kata kunci" pada, dengan cara membandingkan "nama pengguna" dan "kata kunci" yang ia terima dari "*class antarmuka*"/form login dan "nama pengguna" dan "kata kunci" yang ia terima dari "*class dBs*"
- d. "*Class Algoritma*" menyampaikan hasil validasi "nama pengguna" dan "kata kunci" ke "*class antarmuka*"/form login, dan selanjutnya "*class antarmuka*" meneruskan hasil validasi tersebut ke *class actor/user*. Perhatikan bahwa aliran informasi balik (garis balik putus-putus) berupa "hasil validasi akun pengguna"

merupakan "nilai kembali" dari proses "mengirim akun pengguna" untuk diverifikasi.

Perlu diketahui bahwa komposisi objek/*class* pada sebuah *sequence diagram* yang menggambarkan sistem kerja program aplikasi atau suatu perangkat keras, atau gabungan keduanya, akan berbeda-beda (tergantung pada skenario *use case* yang dibangun). Kadangkala terdapat lebih dari satu objek/*class* yang sejenis (misalnya terdapat lebih dari satu *boundary class*), atau bahkan tidak melibatkan secara lengkap semua jenis objek/*class* yang ada.

## 2. Membuat *Sequence Diagram*

Kita akan menggunakan skenario pada *use case* "memesan lagu" pada Modul 6, dengan langkah-langkah utama seperti yang disajikan pada gambar 7.7.

1. Pengguna meminta informasi lagu
2. Pengguna memasukkan lagu ke keranjang belanja
3. Pengguna *check out* dan memberikan informasi pembayaran
4. Persetujuan pembayaran
5. Lagu dirilis untuk diunduh

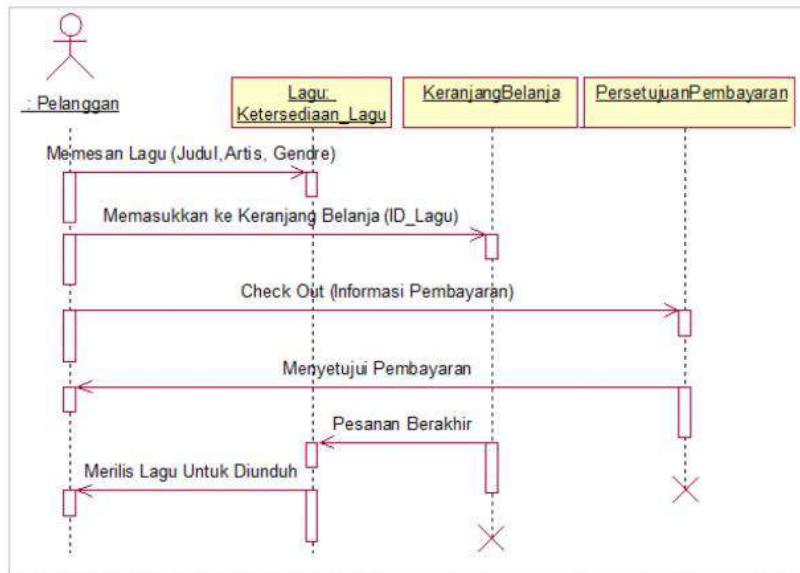
Gambar 7.7  
Contoh Skenario Pelanggan Memesan/Membeli Lagu

Langkah-langkah yang disajikan saat membuat *sequence diagram* mirip dengan langkah-langkah saat membuat DFD (*Data Flow Diagram*).

### a. Mengidentifikasi Object

Langkah pertama adalah mengidentifikasi *instance* dari kelas yang berpartisipasi dalam skenario yang dimodelkan; yaitu, objek yang berinteraksi satu sama lain dalam skenario *use case*. Pikirkan jenis informasi utama yang perlu ditangkap oleh sistem. Objek biasanya dapat diidentifikasi dari skenario *use case* yang dibuat selama pengembangan *use case diagram*. Sumber dan tujuan dari *use case* (yaitu, entitas eksternal atau penyimpanan data) biasanya merupakan titik awal yang baik untuk mengidentifikasi kelas (*class*). Selain itu, kelas dapat berupa aktor eksternal yang direpresentasikan pada *use case diagram*.

#### 7.14 Pemodelan Perilaku (*Behavioral*) Objek



Gambar 7.8  
Contoh *Sequence Diagram* untuk Skenario Memesan/Membeli Lagu

Contoh kelas yang terdapat dalam skenario *memesan (membeli) sebuah lagu* adalah kelas "Pelanggan" dan kelas "Lagu". Ini harus ditempatkan dalam kotak dan dicantumkan di bagian atas Gambar 7.8. Objek/kelas "Pelanggan" dan "Lagu" adalah objek internal dalam sistem, sedangkan objek "Keranjang Belanja" dan "Persetujuan Pembayaran" mewakili aktor eksternal yang muncul di *use case diagram*. Karena objek "Keranjang Belanja" dan "Persetujuan Pembayaran" juga berinteraksi dalam skenario, maka mereka juga dimasukkan ke dalam diagram.

Perlu diingat, pada saat ini kita mencoba untuk mengidentifikasi hanya objek yang terlibat dalam skenario dari sebuah *use case* tertentu. Oleh karena itu, jangan berfikir untuk mengidentifikasi semua objek dalam *use case* dengan sempurna. *Sequence diagram* pada skenario lainnya dapat mengungkap objek yang lainnya. *Class diagram* (yang dibuat di bagian sebelumnya) juga dapat menjelaskan bagaimana kelas-kelas didefinisikan dan diperhalus. Namun, berdasarkan *sequence diagram* yang dibuat, *class diagram* biasanya direvisi, karena analis telah memiliki pemahaman yang lebih baik tentang kelas setelah mereka mengembangkannya dengan tools yang lain (misalnya *sequence diagram*). Perhatikan, bahwa objek "Keranjang Belanja" dan "Persetujuan Pembayaran" hanya memiliki *lifeline* sementara dan dihancurkan sebelum skenario proses berakhir (tanda "X").

##### b. Menempatkan Lifeline dan Eksekusi

Proses selanjutnya adalah menyajikan waktu di mana sebuah objek berpartisipasi dalam skenario. Garis putus-putus vertikal ditambahkan di bawah setiap objek untuk menunjukkan keberadaan objek selama skenario, dan "X" harus ditempatkan di bawah

objek pada ujung *lifeline* (garis hidup) di mana mereka tidak lagi berinteraksi dengan objek lain. Kotak persegi panjang yang sempit di atas *lifeline* menunjukkan masa waktu objek mengirim dan menerima pesan.

#### c. Menambahkan Pesan

Proses akhir adalah menggambarkan panah untuk mewakili pesan yang diteruskan dari objek ke objek, dengan panah padat menunjuk ke arah transmisi pesan. Panah harus ditempatkan secara berurutan dari pesan pertama (di atas) hingga terakhir (di bawah) untuk menunjukkan urutan waktu proses. Parameter atau nilai apapun yang diteruskan bersama pesan harus ditempatkan dalam tanda kurung di samping nama pesan. Jika terdapat pesan yang diharapkan dikembalikan sebagai tanggapan atas pesan, gambarkan pesan yang dimaksud dengan panah putus-putus. Namun demikian, pesan balasan dapat tidak secara eksplisit ditampilkan pada diagram. Perhatikan Gambar 7.8, pada gambar tersebut tidak menyertakan pesan kembali ke Pelanggan sebagai tanggapan atas "permintaan lagu". Dalam hal ini, diasumsikan bahwa pelanggan akan menerima "pesan respon" tentang lagu yang diminta.

### C. COMMUNICATION DIAGRAM

Seperti halnya *sequence diagram*, *communication diagram* pada dasarnya memberikan pandangan tentang aspek dinamis dari sistem berorientasi objek. Dengan demikian, diagram tersebut dapat menunjukkan bagaimana sekumpulan objek berkolaborasi untuk mengimplementasikan *use case* atau skenario *use case*. Dalam hal ini, *communication diagram* dapat menggambarkan ketergantungan objek yang berbeda satu sama lain. *Communication diagram* pada dasarnya adalah diagram objek yang menunjukkan hubungan penyampaian pesan, bukan asosiasi agregasi atau generalisasi. *Communication diagram* sangat berguna untuk menunjukkan pola proses (yaitu, pola aktivitas yang terjadi pada sekumpulan kelas yang berkolaborasi).

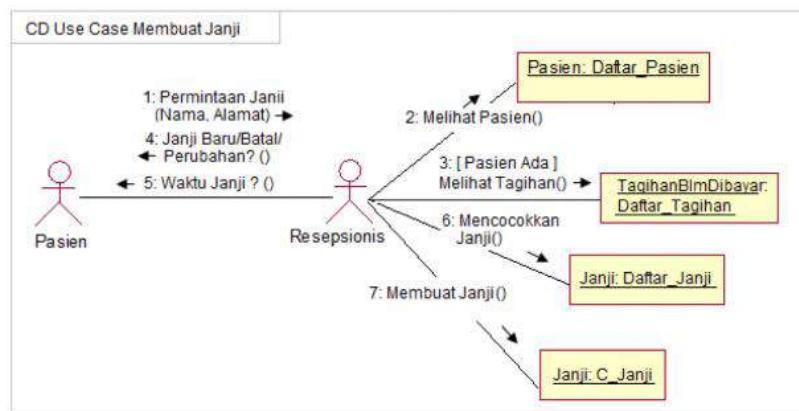
*Communication diagram* setara dengan *sequence diagram*, tetapi diagram tersebut menekankan aliran pesan melalui sekumpulan objek, sedangkan *sequence diagram* berfokus pada pengurutan waktu dari pesan yang disampaikan. Oleh karena itu, untuk memahami aliran kontrol atas sekumpulan objek yang berkolaborasi atau untuk memahami objek apa saja yang berkolaborasi untuk mendukung proses bisnis, *communication diagram* dapat digunakan, sedangkan untuk melihat waktu pemesanan pesan, harus menggunakan *sequence diagram*. Dalam beberapa kasus, keduanya dapat digunakan untuk lebih memahami aktivitas dinamis sistem (Dennis, 2009).

#### 1. Elemen-elemen *Communication Diagram*

Gambar 7.9 menunjukkan contoh *communication diagram* untuk *use case* "Membuat Janji" di Klinik Dokter (ekuivalen dengan *sequence diagram* Gambar 7.2).

## 7.16 Pemodelan Perilaku (*Behavioral*) Objek

Aktor dan objek yang berkolaborasi untuk mengeksekusi *use case* ditempatkan pada diagram komunikasi untuk menggambarkan penyampaian pesan yang terjadi di antara mereka. Perhatikan bahwa aktor dan objek pada Gambar 7.9 adalah yang sama dengan yang terdapat pada Gambar 7.2: seorang *Pasien*, seorang *Resepsionis*, *Pasien*, *UnpaidBills* (tagihan belum dibayar), *Appointment* (Janji), dan *anAppt* (sebuah janji). Sekali lagi, seperti pada *sequence diagram*, untuk masing-masing objek, nama kelas di mana mereka berada merupakan *instance* yang diberikan setelah nama objek (misalnya, *Pasien*: *Daftar\_Pasien*). Elemen-elemen *communication diagram* disajikan pada Gambar 7.10. Tidak seperti *sequence diagram*, *communication diagram* tidak memiliki sarana untuk secara eksplisit menunjukkan objek yang sedang dihapus atau diciptakan. Diasumsikan bahwa ketika pesan dihapus, dihancurkan, atau dikirim ke suatu objek lain, keberadaannya telah hilang, dan pesan baru akan menyebabkan objek baru muncul.



Gambar 7.9  
Contoh *Communication Diagram* untuk Skenario *Use Case*  
"Membuat Janji" di Klinik Dokter

Asosiasi ditampilkan antara aktor dan objek dengan garis tidak berarah. Misalnya, ada hubungan yang ditunjukkan antara *aktor Pasien* dan *aktor Resepsionis*. Pesan ditampilkan berupa label pada asosiasi. Label disertai garis dengan panah yang menunjukkan arah pengiriman pesan. Misalnya, pada Gambar 7.9, *aktor Pasien* mengirim pesan "Permintaan Janji ()" ke *aktor Resepsionis*, dan *aktor Resepsionis* mengirim pesan "Janji Baru/Batal/Perubahan? ()" dan "Waktu Janji? ()" ke *aktor Pasien*. Urutan pesan yang dikirim ditentukan dengan nomor urut. Pada Gambar 7.9, pesan "Permintaan Janji ()" adalah pesan pertama yang dikirim, sedangkan pesan "Janji Baru/Batal/Perubahan? ()" dan "Waktu Janji ()" adalah pesan keempat dan kelima yang dikirim secara terpisah.

Seperti *sequence diagram*, *communication diagram* dapat merepresentasikan pesan bersyarat. Misalnya, pada Gambar 7.9, pesan "Melihat\_Tagihan ()" hanya dikirim jika kondisi [pasien ada] terpenuhi. Jika pesan dikirim berulang kali, tanda bintang (\*)

ditempatkan setelah nomor urut. Sebuah asosiasi yang berputar (kembali) ke sebuah objek menunjukkan pendelegasian pada diri sendiri. Pesan tersebut ditampilkan sebagai label asosiasi.

NOTASI	DEFINISI
 <b>Aktor</b> <div style="border: 1px solid red; padding: 2px;">&lt;&lt; Aktor &gt;&gt;</div>	<p><b>Aktor (Actor):</b></p> <ol style="list-style-type: none"> <li>1. Orang atau sistem yang memperoleh manfaat dan berada di luar sistem.</li> <li>2. Berpartisipasi dalam kolaborasi dengan mengirim dan/atau menerima pesan.</li> <li>3. Digambarkan sebagai gambar tongkat (default) atau, jika melibatkan aktor bukan manusia, sebagai persegi panjang dengan &lt;&lt;actor&gt;&gt; di dalamnya (alternatif).</li> </ol>
<div style="border: 1px solid red; padding: 2px;">Objek : Kelas</div>	<p><b>Objek (Object):</b></p> <ol style="list-style-type: none"> <li>1. Berpartisipasi secara berurutan dengan mengirim dan/atau menerima pesan.</li> <li>2. Ditempatkan di bagian atas diagram.</li> </ol>
<hr/>	<p><b>Asosiasi (Association):</b></p> <ol style="list-style-type: none"> <li>1. Menunjukkan hubungan antara aktor dan/atau objek.</li> <li>2. Digunakan untuk mengirim pesan.</li> </ol>
<u>Nomor Urut: Pesan ➔</u>	<p><b>Pesan (Message):</b></p> <ol style="list-style-type: none"> <li>1. Menyampaikan informasi dari satu objek ke objek lainnya.</li> <li>2. Memiliki arah yang ditunjukkan dengan menggunakan panah.</li> <li>3. Memiliki urutan yang ditunjukkan dengan nomor urut.</li> </ol>
<u>Nomor Urut: [ Kondisi ]: Pesan ➔</u>	<p><b>Pesan Bersyarat (Guard Condition):</b> Merupakan syarat yang harus dipenuhi untuk pesan yang akan dikirim.</p>
<div style="border: 1px solid red; padding: 2px;">Konteks</div>	<p><b>Bingkai (Frame):</b> Menunjukkan konteks diagram komunikasi.</p>

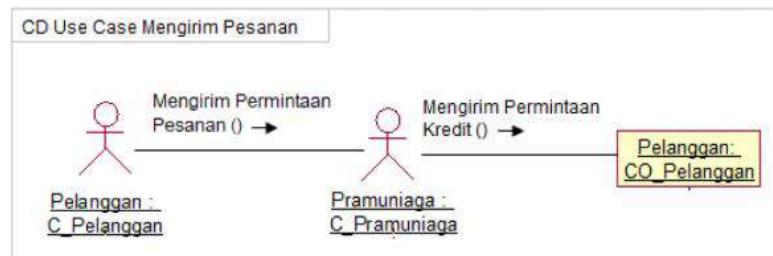
Gambar 7.10  
Notasi pada *Communication Diagram*

Ketika *communication diagram* terisi penuh dengan semua objek, itu bisa menjadi sangat kompleks dan sulit untuk dipahami. Jika ini terjadi, diagram perlu disederhanakan. Salah satu pendekatan untuk menyederhanakan diagram komunikasi (seperti yang dilakukan pada *use case diagram* dan *class diagram*), yaitu melalui penggunaan paket (pengelompokan kelas secara logis, dengan cara mengelompokkan

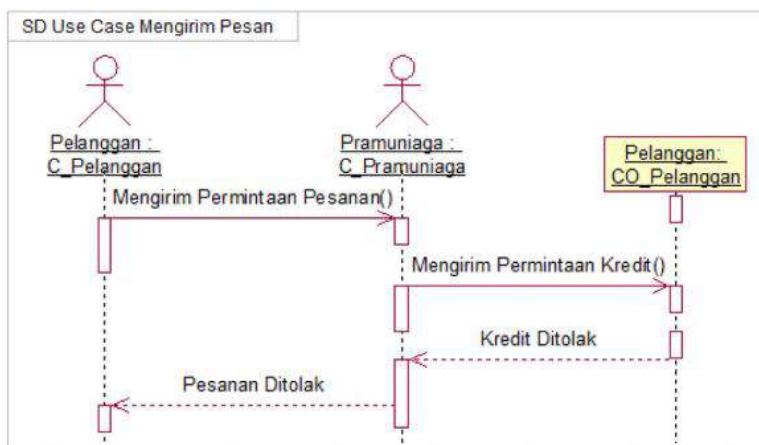
### 7.18 Pemodelan Perilaku (*Behavioral*) Objek

kelas-kelas yang saling berkaitan). Dalam kasus *communication diagram*, objeknya dikelompokkan berdasarkan pesan yang dikirim dan diterima dari objek lain.

Gambar 7.11 adalah contoh lain *communication diagram*. Diagram ini ekuivalen dengan *sequence diagram* pada Gambar 7.12. Namun, ketika membandingkan *communication diagram* dengan *sequence diagram* dalam gambar ini, kita dapat melihat bahwa terdapat informasi yang hilang.



Gambar 7.11  
Contoh Lain *Communication Diagram*



Gambar 7.12  
Contoh Pengembalian Nilai pada *Sequence Diagram*

*Communication diagram* tidak pernah menampilkan pengembalian dari pengiriman pesan, sedangkan *sequence diagram* dapat menampilkannya secara opsional. Pada *sequence diagram* Gambar 7.12, pesan "kredit ditolak" adalah nilai yang dikembalikan dari pesan "pengiriman permintaan kredit", demikian juga dengan pesan "pesanan ditolak" merupakan nilai yang dikembalikan dari pesan "mengirim permintaan pesanan".

## 2. Membangun *Communication Diagram*

Perlu diingat bahwa *communication diagram* pada dasarnya adalah diagram objek (*object diagram*) yang menunjukkan hubungan penyampaian pesan alih-alih

asosiasi agregasi atau generalisasi. Gambar 7.13 menunjukkan proses lima langkah yang digunakan untuk membangun *communication diagram* (Dennis, 2009):

1. Menentukan konteksnya (ruang lingkupnya)
2. Mengidentifikasi objek (aktor) dan asosiasi antara objek yang berpartisipasi dalam kolaborasi
3. Membuat communication diagram
4. Menambahkan pesan
5. Validasi diagram

**Gambar 7.13**  
Langkah-langkah Membangun *Communication Diagram*

**Langkah pertama** adalah menentukan konteks (ruang lingkup) dari *communication diagram*. Seperti *sequence diagram*, diagram konteks dapat melingkupi sebuah sistem, sebuah *use case*, atau sebuah skenario *use case*. Konteks diagram digambarkan sebagai bingkai berlabel yang melingkupi diagram.

**Langkah kedua** adalah mengidentifikasi objek (aktor) dan asosiasi yang menghubungkan objek (aktor) yang berpartisipasi dalam kolaborasi bersama. Perlu diingat bahwa objek yang berpartisipasi dalam kolaborasi adalah contoh kelas yang diidentifikasi selama pengembangan model struktural. Seperti proses pembuatan *sequence diagram*, kemungkinan besar terdapat objek tambahan, dan karenanya terdapat kelas baru yang ditemukan. Kejadian seperti ini normal, karena proses pengembangan yang mendasari adalah iteratif dan inkremental. Dengan demikian, selain *communication diagram* perlu dimodifikasi, *sequence diagram* dan model struktural mungkin juga harus dimodifikasi. Selain itu, kebutuhan fungsional tambahan juga dapat ditemukan, sehingga model fungsional juga perlu dimodifikasi.

**Langkah ketiga** adalah meletakkan objek (aktor) dan asosiasi mereka pada *communication diagram* dengan menempatkan mereka bersama-sama berdasarkan asosiasi yang mereka miliki dengan objek lain dalam kolaborasi. Dengan berfokus pada asosiasi antara objek (aktor) dan meminimalkan jumlah asosiasi yang saling bersilangan, kita dapat mempermudah pemahaman terhadap diagram.

**Langkah keempat** adalah menambahkan pesan ke asosiasi antara objek. Ini dilakukan dengan menambahkan "nama pesan" ke tautan asosiasi antara objek dan "panah" yang menunjukkan arah pesan yang dikirim. Selain itu, setiap pesan memiliki nomor urut yang terkait dengannya untuk menggambarkan urutan pesan berdasarkan waktu kejadian.

**Langkah kelima** (terakhir) adalah memvalidasi *communication diagram*. Tujuan dari langkah ini adalah untuk menjamin bahwa *communication diagram* dengan tepat menggambarkan proses yang mendasarinya, hal ini dilakukan dengan memastikan bahwa semua langkah dalam proses tersebut digambarkan pada diagram.



### Latihan

Untuk memperdalam pemahaman Anda mengenai materi di atas, kerjakanlah latihan berikut!

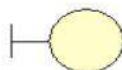
- 1) Jelaskan persamaan dan perbedaan mendasar antara *sequence diagram* dan *communication diagram*!
- 2) Jelaskan penggunaan *sequence diagram* dalam fase-fase pengembangan sistem!
- 3) Sebutkan dan gambarkan tiga notasi kelas (*class*) yang biasa digunakan dalam penggambaran *sequence diagram*, khususnya yang terkait dengan sistem kerja pada perangkat lunak dan atau perangkat keras. Jelaskan kegunaannya, serta berikan contoh objek yang diwakilkannya!
- 4) Misalkan terdapat sebuah aplikasi untuk **Merekam Data Peminjaman** pada **Sistem Perpustakaan**, dengan skenario bisnis berikut:
  - a) Anggota menyerahkan kartu tanda anggota ke Pustakawan
  - b) Pustakawan memasukkan nomor anggota ke sistem aplikasi
  - c) Sistem memverifikasi nomor anggota
  - d) Sistem menampilkan data identitas anggota
  - e) User memasukkan kode pustaka yang akan dipinjam
  - f) Sistem memverifikasi kode pustaka yang akan dipinjam
  - g) Sistem menampilkan data identitas pustaka yang akan dipinjam
  - h) Sistem menawarkan untuk menambah pustaka lainnya
  - i) Sistem merekam data peminjaman
  - j) Sistem mencetak file bukti peminjaman, dan mengakhiri proses.
 Gambarkan *Sequence Diagram* dan *Communication Diagram*-nya!

#### Petunjuk Jawaban Latihan

- 1) *Sequence diagram* dan *communication diagram* keduanya menyediakan model dinamis yang menggambarkan interaksi antara objek yang berpartisipasi dalam *use case*.  
Perbedaan utama antara kedua diagram tersebut adalah: *sequence diagram* berfokus pada pengurutan waktu atau pengurutan pesan yang dikirim antar objek, sedangkan *communication diagram* menyoroti aliran kontrol atas sekumpulan objek yang berkolaborasi (sifat kolaborasi dari objek) yang mendukung *use case*.
- 2) Penggunaan *sequence diagram* dalam fase-fase pengembangan sistem:  
*Sequence diagram* dapat digunakan pada fase analisis dan fase desain untuk memodelkan sistem secara konsep (model proses sistem); juga digunakan sangat spesifik pada fase implementasi, yaitu memodelkan sistem secara fisik, yang

menyertakan objek-objek seperti database atau komponen GUI (*Graphical User Interface*) tertentu sebagai kelas.

- 3) Tiga notasi *class* yang biasa digunakan dalam penggambaran *sequence diagram* (khususnya yang terkait dengan sistem kerja pada perangkat lunak dan atau perangkat kelas), kegunaannya, serta contoh objek yang diwakilkannya:
- Boundary class. Digunakan untuk menggambarkan objek-objek antarmuka sistem dengan aktor (pengguna). Contoh objek: antarmuka input dan antarmuka output pada program aplikasi, keyboard, dan layar komputer.



- Boundary class. Digunakan untuk menggambarkan objek-objek antarmuka sistem dengan aktor (pengguna). Contoh objek: antarmuka input dan antarmuka output pada program aplikasi, keyboard, dan layar komputer.
- Control class. Digunakan untuk menggambarkan sistem pengendali aktivitas dalam sistem. Contoh: algoritma atau sub rutin/prosedur dalam sebuah program aplikasi.

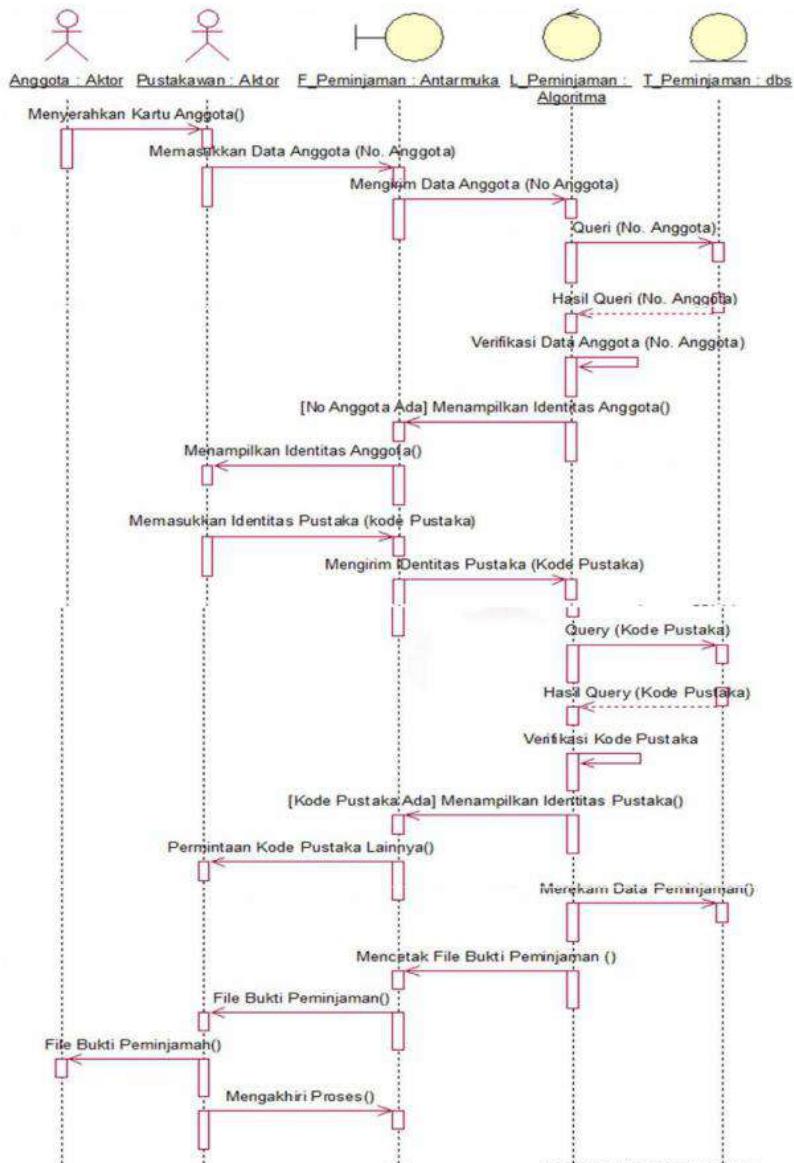


- Boundary class. Digunakan untuk menggambarkan objek-objek antarmuka sistem dengan aktor (pengguna). Contoh objek: antarmuka input dan antarmuka output pada program aplikasi.
- Control class. Digunakan untuk menggambarkan sistem pengendali aktivitas dalam sistem. Contoh: algoritma atau sub rutin/prosedur dalam sebuah program aplikasi.
- Entity class. Digunakan untuk menggambarkan media penyimpanan data. Contoh: tabel, database, cloud data, variabel memory dalam sebuah sistem aplikasi.



## 7.22 Pemodelan Perilaku (*Behavioral*) Objek

### 4) Sequence Diagram Merekam Data Peminjaman



### Rangkuman

Diagram interaksi (*interaction diagram*) digunakan untuk menggambarkan bagaimana objek berkolaborasi untuk mendukung *use case*. Ada dua jenis *interaction diagram*: *sequence diagram* dan *communication diagram*. Kedua diagram menyediakan model dinamis yang menggambarkan interaksi antara objek yang terkait dengan *use case*. Perbedaan utama antara kedua diagram tersebut adalah *sequence diagram*

berfokus pada pengurutan waktu atau urutan pesan yang dikirim antar objek, sedangkan *communication diagram* menyoroti aliran kontrol atas sekumpulan objek yang berkolaborasi.

*Sequence diagram* menggambarkan contoh kelas yang berpartisipasi dalam *use case* dan pesan yang lewat di antara mereka dari waktu ke waktu. Objek ditempatkan secara horizontal di atas *sequence diagram*, masing-masing memiliki garis putus-putus, yang disebut garis hidup, secara vertikal di bawahnya. Fokus kontrol, diwakili oleh persegi panjang tipis, ditempatkan di atas garis hidup untuk menunjukkan saat objek mengirim atau menerima pesan. Pesan adalah komunikasi antar objek yang menyampaikan informasi dengan ekspektasi bahwa aktivitas akan terjadi, dan pesan ditampilkan dengan panah yang menghubungkan dua objek yang menunjuk ke arah pesan tersebut disampaikan. Untuk membuat *sequence diagram*, pertama-tama identifikasi kelas yang berpartisipasi dalam *use case*, lalu tambahkan pesan yang lewat di antaranya. Terakhir, perlu menambahkan garis hidup dan fokus kontrol. *Sequence diagram* berguna untuk memahami spesifikasi waktu nyata dan untuk skenario kompleks dari sebuah *use case*.

*Communication diagram* menyoroti sifat kolaborasi dari objek yang mendukung *use case*. *Communication diagram* pada dasarnya adalah diagram objek yang menggambarkan hubungan dalam pengiriman pesan, bukan hubungan struktural. Untuk membuat *communication diagram*, pertama-tama identifikasi objek (aktor) dan asosiasi yang menghubungkan objek yang berpartisipasi dalam kolaborasi, lalu letakkan objek dan asosiasi mereka pada diagram. Tambahkan pesan yang lewat di antaranya. Terakhir, lakukan validasi terhadap diagram, untuk menjamin bahwa diagram dengan tepat menggambarkan proses yang mendasarinya.



### Tes Formatif 1

Pilihlah satu jawaban yang paling tepat!

- 1) Model perilaku internal (*behavioral model*) yang digunakan sebagai representasi detail proses bisnis yang digambarkan oleh model *use case* adalah ....
  - A. *activity diagram*
  - B. *sequence diagram*
  - C. *behavioral state machines diagram*
  - D. *data flow diagram*
  
- 2) Salah satu perbedaan antara *class diagram* dan *interaction diagram* adalah ....
  - A. *class diagram* berfokus pada pemodelan terstruktur, sedangkan *interaction diagram* berfokus pada pemodelan berorientasi objek
  - B. *class diagram* berfokus pada pemodelan tingkat objek, sedangkan *interaction diagram* berfokus pada pemodelan tingkat kelas
  - C. *class diagram* berfokus pada pemodelan tingkat kelas, sedangkan *interaction diagram* berfokus pada pemodelan tingkat objek
  - D. Pilihan jawaban A dan C benar

#### 7.24 Pemodelan Perilaku (*Behavioral*) Objek

---

- 3) Terdapat tiga jenis pesan yang dapat digambarkan pada *sequence diagram*, *kecuali* ....
  - A. pesan penghancuran objek
  - B. pesan/panggilan operasi
  - C. pesan pengembalian nilai dari pesan operasi
  - D. pesan penciptaan objek
- 4) Pesan yang disimbolkan dengan label panah putus-putus adalah ....
  - A. pesan/panggilan operasi
  - B. pesan pengembalian nilai dari pesan operasi
  - C. pesan penciptaan objek
  - D. jawaban A dan C benar
- 5) Berikut adalah ketentuan yang perlu diperhatikan dalam penggambaran *sequence diagram*:
  - A. Cakupan *sequence diagram* hanya pada lingkup yang menggambarkan skenario tunggal dalam *use case*
  - B. Objek-objek yang ada tidak harus ditempatkan dalam urutan tertentu, meskipun sangat bagus jika mengaturnya sesuai dengan urutan logis proses, seperti urutan partisipasi mereka dalam aktivitas. Untuk setiap objek, nama kelas tempat mereka berada diberikan setelah nama objek
  - C. *Sequence diagram* tidak memiliki sarana untuk secara eksplisit menunjukkan objek yang sedang dihapus
  - D. *Sequence diagram* tidak memanggil operasi yang ada pada dirinya sendiri
- 6) Berikut adalah ketentuan yang perlu diperhatikan dalam penggambaran *communication diagram*, *kecuali* ....
  - A. Sama seperti *sequence diagram*, *communication diagram* memiliki sarana untuk secara eksplisit menunjukkan objek yang sedang dihapus atau diciptakan
  - B. Asosiasi ditampilkan antara aktor dan objek dengan garis tidak berarah
  - C. Tidak pernah menampilkan pengembalian dari pengiriman pesan, sedangkan *sequence diagram* dapat menampilkannya secara opsional
  - D. Seperti *sequence diagram*, *communication diagram* dapat merepresentasikan pesan bersyarat
- 7) *Sequence diagram* digunakan untuk memvisualisasi bagaimana objek-objek berinteraksi, dan bukan untuk menyajikan prosedur proses atau sebagai cara pemodelan logika kontrol, seperti pada *activity diagram*. Pernyataan ini ....
  - A. Benar
  - B. Salah

- 8) *Sequence diagram* hanya digunakan untuk menggambarkan skenario tunggal sebuah *use case*. Pernyataan ini ....
- Benar
  - Salah
- 9) Membuat gambar *communication diagram* harus ekuivalen dengan gambar *sequence diagram*, terkecuali bahwa *communication diagram* tidak menampilkan pengembalian nilai pengiriman pesan, sedangkan *sequence diagram* dapat menampilkannya secara opsional. Pernyataan ini ....
- Benar
  - Salah
- 10) Dalam penggambaran *communication diagram*, tidak dibenarkan untuk menambah objek baru (selain objek-objek yang telah diidentifikasi selama pengembangan model struktural dan selama proses pembuatan *sequence diagram*). Pernyataan ini ....
- Benar
  - Salah

Cocokkanlah jawaban Anda dengan Kunci Jawaban Tes Formatif 1 yang terdapat di bagian akhir modul ini. Hitunglah jawaban yang benar. Kemudian, gunakan rumus berikut untuk mengetahui tingkat penguasaan Anda terhadap materi Kegiatan Belajar 1.

$$\text{Tingkat Penguasaan} = \frac{\text{Jumlah Jawaban yang Benar}}{\text{Jumlah Soal}} \times 100$$

Arti tingkat penguasaan

<70%	70% - 79%	80% - 89%	90% - 100%
kurang	cukup	baik	baik sekali

Apabila mencapai tingkat penguasaan 80% atau lebih, Anda dapat meneruskan dengan Kegiatan Belajar 2. **Bagus!** Jika masih di bawah 80%, Anda harus mengulangi materi Kegiatan Belajar 1, terutama bagian yang belum dikuasai.

## *Behavioral State Machines* *Diagram*

B eberapa kelas dalam *class diagram* mewakili sekumpulan objek yang cukup dinamis karena mereka melewati berbagai keadaan selama keberadaannya. Misalnya, status sebuah kendaraan dapat berubah dari waktu ke waktu dari "kendaraan baru" menjadi "kendaraan bekas", berdasarkan statusnya di dealer.

*State machine diagram* atau disebut juga sebagai *state chart diagram*, merupakan diagram untuk menggambarkan transisi dan perubahan keadaan (dari satu *state* ke *state* lainnya) suatu objek dalam suatu sistem. Dinamakan *state machine* karena diagram tersebut pada dasarnya adalah mesin yang menggambarkan beberapa keadaan suatu objek dan bagaimana objek tersebut berubah berdasarkan kejadian internal ataupun eksternal.

*Behavioral state machines diagram* adalah model dinamis yang menunjukkan status yang berbeda yang dilalui oleh satu objek selama hidupnya sebagai respons terhadap suatu peristiwa, bersama dengan respons dan tindakannya. Biasanya, *behavioral state machines* tidak digunakan untuk semua objek, tetapi hanya untuk mendefinisikan objek kompleks lebih lanjut guna membantu menyederhanakan desain algoritma untuk suatu prosedur. *Behavioral state machines* menunjukkan berbagai status objek dan peristiwa apa yang menyebabkan objek berubah dari satu status ke status lainnya. Dibandingkan dengan *interaction diagram*, *behavioral state machines diagram* harus digunakan untuk membantu memahami aspek dinamis dari satu kelas dan bagaimana instansinya berkembang dari waktu ke waktu, dan bukan melihat bagaimana *use case* atau *skenario use case* dieksekusi melalui serangkaian kelas. Dalam pendekatan berorientasi objek, menggambar *behavioral state machines diagram* dilakukan untuk sebuah *class* tunggal untuk menunjukkan behavior seumur hidup sebuah objek tunggal (Fowler, 2005). Namun demikian, tidak menutup kemungkinan menggambar *behavioral state machines diagram* dilakukan untuk kumpulan kelas, subsistem, atau seluruh sistem (Dennis, 2005).

Bagian ini akan menjelaskan penggunaan *behavioral state machines diagram* untuk memodelkan perubahan status yang dilalui objek yang kompleks. Seperti dalam pembuatan *interaction diagram*, saat kita membuat *behavioral state machines diagram* untuk suatu objek, ada kemungkinan kita akan mengungkap peristiwa tambahan (berupa operasi tambahan yang perlu dilakukan) yang perlu dimasukkan dalam model fungsional, termasuk dalam model struktural. Jadi, mungkin *interaction diagram* yang

telah dibuat sebelumnya harus dimodifikasi lagi. Sekali lagi, karena pengembangan berorientasi objek bersifat iteratif dan inkremental, diharapkan terjadi modifikasi berkelanjutan dari model sistem yang dikembangkan (fungsional, struktural, dan perilaku).

#### A. ELEMEN-ELEMEN BEHAVIORAL STATE MACHINE DIAGRAM

Gambar 7.14 menyajikan contoh diagram *behavioral state machines diagram* yang mewakili kelas *penawaran* dalam konteks *dealer kendaraan*. Dari diagram ini, kita dapat mengetahui bahwa sebuah *penawaran* yang *tiba/sampai* di dealer, *tercatat*. Jika pemilik dealer menilai bahwa penawaran itu *dapat diterima*, penawaran itu *ditutup* dan tidak lagi dianggap sebagai *penawaran* setelah dua hari berlalu. Jika sebuah penawaran *ditolak*, itu dapat dinegosiasikan ulang sampai *diterima*.



Gambar 7.14  
Contoh Behavioral State Machines Diagram untuk Penawaran Kendaraan

**Keadaan.** Keadaan/status (*state*) adalah sekumpulan nilai yang mendeskripsikan objek pada titik waktu tertentu, dan mewakili suatu titik dalam kehidupan objek di mana ia memenuhi beberapa kondisi, melakukan beberapa tindakan, atau menunggu sesuatu terjadi. Pada Gambar 7.14, *state* meliputi *tiba*, *menunggu keputusan*, *ditutup*, dan *dalam negosiasi*. Sebuah *state* digambarkan oleh simbol keadaan, yaitu persegi panjang dengan sudut membulat dengan label deskriptif yang mengkomunikasikan keadaan tertentu. *Keadaan awal* ditunjukkan dengan menggunakan lingkaran kecil berisi, dan *keadaan akhir* sebuah objek ditampilkan sebagai lingkaran yang mengelilingi sebuah lingkaran kecil yang diisi. Pengecualian ini menggambarkan kapan keadaan suatu objek dimulai dan keadaan berakhir.

Atribut atau properti dari suatu objek mempengaruhi keadaannya. Namun, tidak semua atribut atau perubahan atribut akan berpengaruh. Misalnya, pikirkan tentang alamat pelanggan. Atribut tersebut memiliki kemungkinan yang kecil untuk perubahan status penawaran.

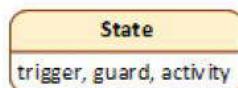
**Peristiwa.** Peristiwa (*event*) adalah nilai yang menggambarkan suatu objek, yang pada gilirannya mengubah keadaan objek tersebut. Ini bisa berupa: *trigger-signature*, *guard condition* [kondisi yang ditentukan menjadi benar], dan *activity*. Seluruh bagian

ini adalah pilihan. *Trigger-signature* biasanya berupa sebuah *event* tunggal yang memicu perubahan pada *state*. *Guard condition* (jika ada) merupakan sebuah keadaan *boolean* yang harus *true* (terpenuhi) supaya terjadi transisi. *Activity* adalah beberapa *behavior* yang dieksekusi selama transisi. Keadaan objek menentukan tanggapan yang tepat yang diberikan. Pada Gambar 7.14, *tiba di dealer*, *keputusan yang disetujui*, dan *selang waktu 2 hari* adalah peristiwa yang menyebabkan perubahan status *penawaran*.

Ketiga bagian pada *event* tersebut (*trigger-signature*, *guard condition*, dan *activity*) merupakan pilihan. Tidak adanya *activity* menunjukkan bahwa kita tidak melakukan apa-apa selama transisi tersebut. Tidak adanya *guard* menunjukkan bahwa kita selalu mengambil transisi tersebut jika *event* tersebut muncul. Tidak adanya *trigger-signature* (sangat jarang) menunjukkan bahwa kita langsung mengambil transisi tersebut. Panah digunakan untuk menghubungkan simbol *keadaan*, mewakili transisi antar *keadaan*.

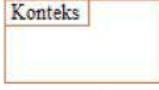
**Transisi.** Transisi (*transitions*) adalah hubungan yang merepresentasikan pergerakan suatu objek dari satu keadaan ke keadaan lain. Setiap panah diberi label dengan nama peristiwa (*event*) yang sesuai dan dengan parameter atau ketentuan yang mungkin berlaku. Misalnya, dua transisi dari *menunggu keputusan* ke *ditutup* dan *dalam negosiasi* berisi suatu pengkondisian.

Sebuah *state* dapat bereaksi terhadap *event* tanpa adanya *transition* dengan menggunakan aktivitas internal, yaitu memasukkan *event* (*trigger*, *guard*, dan *activity*) dalam kotak *state* itu sendiri (Fowler, 2005), disimbolkan seperti pada Gambar 7.15.



Gambar 7.15  
State dengan Aktivitas Internal

NOTASI	DEFINISI
Keadaan	<p>Keadaan/status (<i>state</i>):</p> <ol style="list-style-type: none"> <li>Ditampilkan sebagai persegi panjang dengan sudut membulat.</li> <li>Memiliki nama yang merepresentasikan keadaan suatu objek.</li> </ol>
●	<p>Keadaan Awal (<i>initial state</i>):</p> <ol style="list-style-type: none"> <li>Ditampilkan sebagai lingkaran kecil berisi.</li> <li>Merupakan titik di mana suatu objek mulai muncul.</li> </ol>
○	<p>Keadaan Akhir (<i>final state</i>):</p> <ol style="list-style-type: none"> <li>Ditampilkan sebagai lingkaran yang mengelilingi lingkaran kecil berisi (sasaran).</li> <li>Menandakan selesainya aktivitas.</li> </ol>
Peristiwa	Peristiwa ( <i>event</i> ):

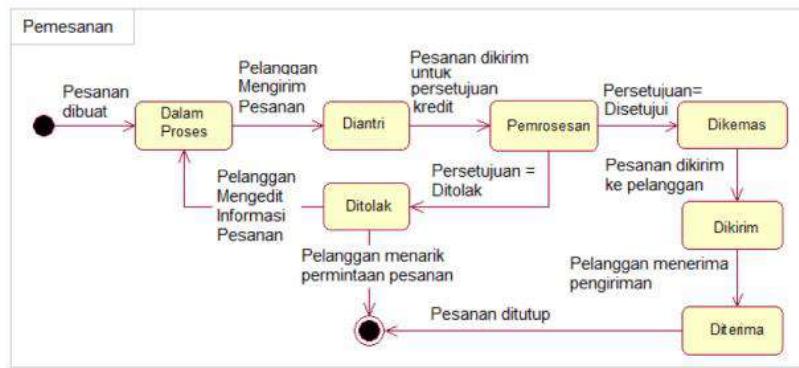
NOTASI	DEFINISI
	<ol style="list-style-type: none"> <li>1. Merupakan kejadian penting yang memicu perubahan <i>state</i> (keadaan).</li> <li>2. Dapat berupa: <i>trigger-signature</i>, kondisi [<i>guard condition</i>], dan <i>Activity</i>.</li> <li>3. Digunakan untuk memberi label transisi.</li> </ol>
	<p>Transisi (<i>transition</i>):</p> <ol style="list-style-type: none"> <li>1. Menunjukkan bahwa suatu objek dalam keadaan pertama akan memasuki keadaan kedua.</li> <li>2. Dipicu oleh terjadinya peristiwa (<i>event</i>) pelabelan transisi.</li> <li>3. Ditampilkan sebagai panah padat dari satu keadaan ke keadaan lain, diberi label dengan nama peristiwa.</li> </ol>
	<p>Bingkai (<i>frame</i>):</p> <p>Menunjukkan konteks <i>behavioral state machines</i>.</p>

Gambar 7.16  
Notasi pada *Behavioral State Machines*

Seperti dalam diagram *behavioral* lainnya, dalam banyak kasus, sangat berguna untuk menunjukkan secara eksplisit konteks/lingkup *behavioral state machines* menggunakan bingkai.

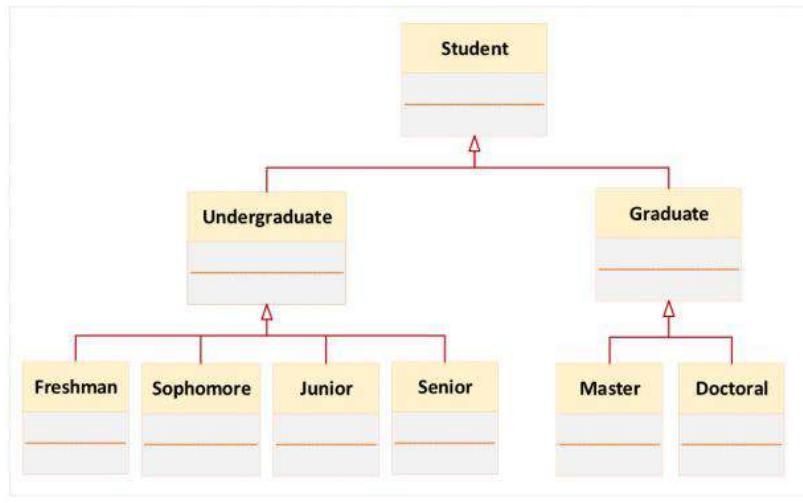
Gambar 7.17 adalah sebuah contoh lain diagram *behavioral state machines*. Gambar ini berhubungan dengan siklus hidup suatu sistem pemesanan. Objek pemesanan dikaitkan dengan skenario *use case Pemesanan* yang dijelaskan pada Gambar 7.11 dan 7.12. Terdapat cukup banyak informasi tambahan yang terkandung dalam *behavioral state machines* ini. Dengan demikian, untuk sistem pemrosesan pesanan, beberapa komponen *sequence diagram* dan *communication diagram* tambahan akan diperlukan untuk menggambarkan semua pemrosesan yang terkait dengan objek pemesanan. Jelas, karena *behavioral state machines* dapat menemukan kebutuhan pemrosesan tambahan, yang dapat sangat berguna dalam mengisi deskripsi lengkap dari sistem yang dikembangkan.

### 7.30 Pemodelan Perilaku (*Behavioral*) Objek

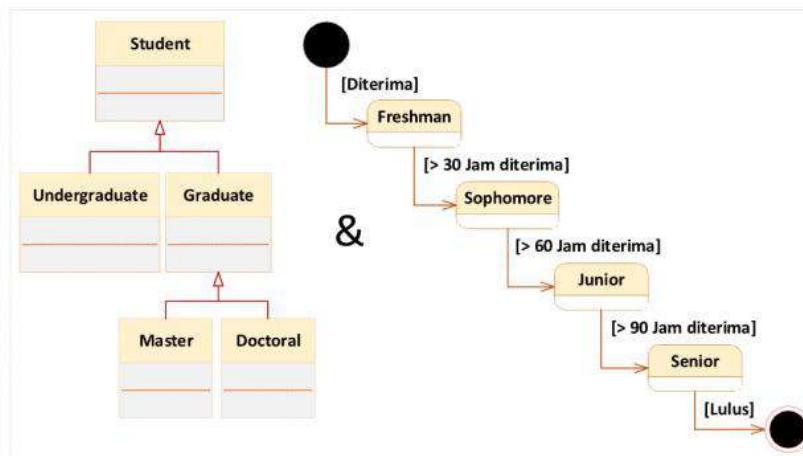


Gambar 7.17  
Behavioral State Machines Diagram untuk Sistem Pemesanan

Terkadang, antara status (*state*) dan *subclass* (subkelas) bisa membingungkan. Misalnya, pada Gambar 7.18, apakah kelas-kelas itu (*Freshman*, *Sophomore*, *Junior*, dan *Senior*) merupakan *subclass* dari kelas *Sarjana*; atau mereka mestinya hanya digambarkan sebagai sebuah *instance* dari kelas *Sarjana* yang berjalan selama masa pakainya? Dalam kasus ini, yang terakhir adalah jawaban yang lebih logis. Ketika kita mencoba untuk mengidentifikasi semua kelas potensial pada saat membuat model struktural, kita mungkin secara tidak sengaja mengidentifikasi *status* dari *superclass* yang relevan daripada *subclass* dari *superclass* tersebut.



(a) Subclass



(b) Subclass dan States

Gambar 7.18  
Perbandingan States dan Subclass

Ini adalah sebuah contoh bagaimana model fungsional, model struktural, dan model perilaku (*behavioral*) dapat saling berkaitan. Dari perspektif pemodelan, daripada kita menghapus *subclass* *Freshman*, *Sophomore*, *Junior*, dan *Senior* dari model struktural (karena dianggap tidak relevan sebagai suatu *subclass*), lebih baik untuk tetap menangkap informasi itu sebagai bagian dari model struktural 7.18 (a) dan menghapusnya (menggantinya dengan *state*) ketika kita membuat model perilaku 7.18 (b). Mengabaikannya, akan berpeluang menyebabkan kehilangan informasi penting tentang domain masalah. Ingat, pengembangan berorientasi objek bersifat iteratif dan inkremental, yang pada awalnya dapat saja membuat banyak kesalahan sebelum kita melanjutkan ke model domain masalah yang "benar".

## B. MEMBUAT BEHAVIORAL STATE MACHINE DIAGRAM

*Behavioral state machine diagram* pada umumnya dibuat untuk menggambarkan satu kelas dari *class diagram*. Biasanya, kelas sangat dinamis dan kompleks, sehingga membutuhkan pemahaman yang baik tentang status mereka dari waktu ke waktu dan peristiwa yang memicu perubahan. Kita harus memeriksa *class diagram* untuk mengidentifikasi kelas mana yang perlu menjalani serangkaian perubahan status yang kompleks dan menggambar diagram untuk masing-masing kelas. Mari kita amati prosedur membuat *behavioral state machine diagram* dari contoh kelas *memilih lagu* pada sistem *Download Musik Digital* (Dennis, 2012).

**Menentukan Konteks.** Seperti model perilaku lainnya, langkah pertama dalam proses ini adalah menentukan **konteks behavioral state machine**, yang ditunjukkan pada label bingkai diagram. Konteks *behavioral state machine* pada umumnya berupa sebuah

*kelas*, namun bisa juga berupa *kumpulan kelas*, *subsistem*, atau *seluruh sistem* (Dennis, 2009).

**Identifikasi State.** Langkah kedua adalah mengidentifikasi berbagai *state* (status/kedaan) yang akan dimiliki oleh objek (memilih lagu) selama *lifetime* (masa pakainya). Informasi ini dikumpulkan dari membaca deskripsi *use case*, berdiskusi (wawancara) dengan pengguna, dan mengandalkan teknik elisitasi kebutuhan (yang yang telah dibahas pada Modul 3). Kita harus mulai dengan menulis langkah-langkah tentang apa yang terjadi pada suatu kelas dari waktu ke waktu, dari awal hingga akhir, mirip dengan cara membuat "langkah-langkah utama pada aliran peristiwa normal" sebuah deskripsi *use case*. Gambar 7.19 menunjukkan urutan dari awal hingga akhir, dari sebuah kelas *memilih lagu*.

1. Pelanggan menambahkan barang ke dalam keranjang belanja
2. Setelah selesai, pelanggan *check out* dan mengajukan pembelian
3. Pembelian menunggu pembayaran
4. Pembayaran disetujui
5. Pembelian menunggu persetujuan akhir pelanggan
6. Pelanggan mengonfirmasi pembelian tersebut
7. Merilis lagu untuk *download*

Gambar 7.19  
Prosedur *Download Lagu* pada Kelas Memilih Lagu

**Identifikasi Transisi.** Langkah selanjutnya adalah mengidentifikasi urutan *status* yang akan dilalui objek *memilih lagu* selama *lifetime* dan kemudian menentukan dengan tepat apa yang menyebabkan setiap *status* terjadi. Tempatkan gambar *status* pada diagram untuk mewakili *status* dan beri label transisi untuk menggambarkan peristiwa yang terjadi yang menyebabkan perubahan *status*. Misalnya, jika Pelanggan *menambahkan lagu ke keranjang*, maka pesanan akan beralih dari *status awal* ke *status dalam keranjang* (Gambar 7.20). Selama *status dalam keranjang*, lagu yang dipilih menunggu keputusan pelanggan untuk *mengajukan pembelian* lagu tersebut. Pada saat itu, lagu tersebut berstatus *menunggu pembayaran*. Setelah *pembayaran disetujui*, lagu tersebut berstatus *menunggu konfirmasi pembelian terakhir* dari pelanggan. Saat *konfirmasi diterima*, lagu berstatus *dirilis untuk diunduh*.



Gambar 7.20  
*Behavioral State Machine Diagram* untuk Memilih Lagu

**Validasi Diagram.** Langkah terakhir adalah melakukan validasi terhadap *behavioral state machine diagram*, dengan memastikan setiap status dapat dilalui (dicapai dan ditinggalkan), kecuali status akhir.



### Latihan

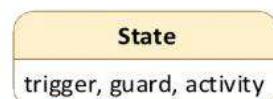
Untuk memperdalam pemahaman Anda mengenai materi di atas, kerjakanlah latihan berikut!

- 1) Sebutkan langkah-langkah dalam membangun *behavioral state machine diagram*?
- 2) Jenis peristiwa apa saja yang dapat menyebabkan transisi status pada *behavioral state machine diagram*?
- 3) Apakah *state* selalu digambarkan menggunakan persegi panjang bulat pada *behavioral state machine diagram*? Jelaskan alasannya!
- 4) Seorang pasien masuk ke klinik dokter dan dirawat setelah *check in*. Ketika seorang dokter mendianosanya dan menemukan pasien dalam keadaan sehat, dia dipulangkan dan tidak lagi dianggap sebagai pasien setelah dua minggu berlalu. Namun, jika hasil diagnosa ditemukan tidak sehat, dia tetap di bawah pengawasan sampai hasil diagnosis berubah menjadi keadaan sehat, untuk selanjutnya dipulangkan dan tidak lagi dianggap sebagai pasien setelah dua minggu berlalu. Gambarkan *behavioral state machine diagram*-nya!

#### Petunjuk Jawaban Latihan

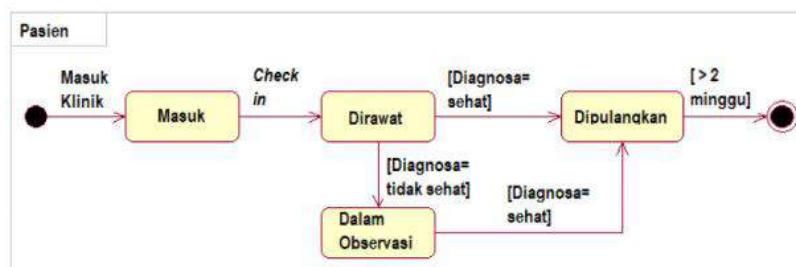
- 1) Langkah-langkah dalam membangun *behavioral state machine diagram*:
  - a) Menentukan konteks diagram, apakah lingkup class, subsistem, atau seluruh sistem.
  - b) Mengidentifikasi keadaan (*state*), dengan cara mempelajari informasi yang terdapat di dalam deskripsi *use case*, dan atau berdiskusi langsung dengan pihak pengguna sistem dengan menggunakan teknik-teknik elisitasi sistem.

- c) Mengidentifikasi transisi (*transition*), dengan cara mengidentifikasi urutan *state* yang akan dilalui objek selama masa hidupnya (*lifetime*). Selanjutnya, menempatkan *state* pada diagram dan memberi label transisi untuk memberikan gambaran mengenai peristiwa yang terjadi yang menyebabkan perubahan *state*.
  - d) Memvalidasi diagram untuk memastikan setiap *state* telah dilalui (dicapai dan ditinggalkan), kecuali *final state*.
- 2) Hal-hal yang dapat menyebabkan transisi status pada *behavioral state machine diagram*:
- a) *Trigger-signature*: biasanya berupa sebuah *event* tunggal yang memicu perubahan pada state.
  - b) *Guard condition* ([ ]): merupakan sebuah keadaan boolean (logika) yang harus *true* (terpenuhi) supaya terjadi transisi.
  - c) *Activity* adalah beberapa *behavior* yang dieksekusi selama transisi.
- 3) *State* tidak selalu digambarkan menggunakan persegi panjang bulat pada *behavioral state machine diagram*. Ada kalanya sebuah *state* bereaksi terhadap *event* tanpa adanya transisi (menggunakan aktivitas internal), maka *event* (*trigger*, *guard*, dan *activity*) harus dimasukkan ke dalam kotak *state*, sehingga bentuk persegi panjang bulat sebagai simbol *state* harus diberi perantara antara nama *state* dan *event*-nya (disebut dengan nama *state with internal behavior*).



Terdapat juga *state* berupa *initial state* (keadaan awal) yang digambarkan sebagai lingkaran kecil berisi, dan *final state* (keadaan akhir) yang digambarkan sebagai lingkaran yang mengelilingi lingkaran kecil berisi.

- 4) *Behavioral State Machine Diagram* "Pasien":





## Rangkuman

*Behavioral state machine* menunjukkan status berbeda yang dilalui oleh kelas tunggal selama fase hidupnya sebagai respons terhadap peristiwa, bersama dengan respons dan tindakan. Status (*state*) adalah sekumpulan nilai yang mendeskripsikan objek pada titik waktu tertentu, dan mewakili sebuah titik dalam kehidupan suatu objek di mana ia memenuhi beberapa kondisi, melakukan beberapa tindakan, atau menunggu sesuatu terjadi. Peristiwa adalah sesuatu yang terjadi pada titik waktu tertentu dan mengubah nilai yang mendeskripsikan suatu objek, yang pada gilirannya, mengubah status objek. Saat objek bergerak dari satu status ke status lain, mereka mengalami transisi.

Saat menggambar *behavioral state machine*, seperti diagram perilaku lainnya, hal pertama adalah menetapkan konteks diagram: sistem, subsistem, kumpulan kelas, atau kelas individu. Kemudian, persegi panjang dengan sudut membulat ditempatkan pada model untuk mewakili berbagai status/keadaan yang akan diambil konteksnya. Selanjutnya, panah ditarik di antara persegi panjang untuk menunjukkan transisi, dan label peristiwa dituliskan di atas panah untuk menjelaskan peristiwa yang menyebabkan terjadinya transisi. Biasanya, *behavioral state machine* digunakan untuk menggambarkan aspek dinamis dari kelas yang kompleks.



## Tes Formatif 2

Pilihlah satu jawaban yang paling tepat!

- 1) *Behavioral state machine diagram* pada umumnya digunakan untuk menggambarkan ....
  - A. sebuah kelas
  - B. kumpulan kelas
  - C. sub sistem
  - D. seluruh sistem
  
- 2) Elemen-elemen dasar dari behavioral state machine diagram, *kecuali* ....
  - A. *class*
  - B. keadaan (*state*)
  - C. transisi (*transition*)
  - D. bingkai (*frame*)
  
- 3) Sekumpulan nilai yang mendeskripsikan objek pada suatu waktu tertentu, di mana nilai tersebut dapat berubah-ubah, disebut ....
  - A. peristiwa (*event*)
  - B. transisi (*transitions*)
  - C. atribut atau properti objek
  - D. keadaan (*state*)

### 7.36 Pemodelan Perilaku (*Behavioral*) Objek

---

- 4) Keadaan (*state*) sebuah objek dapat diubah oleh suatu ....
  - A. peristiwa (*event*)
  - B. transisi (*transitions*)
  - C. atribut atau properti objek tersebut
  - D. Semua jawaban A, B, dan C benar
- 5) Elemen *behavioral state machine diagram* yang menghubungkan suatu keadaan (*state*) dengan keadaan lainnya adalah ....
  - A. peristiwa (*event*)
  - B. transisi (*transitions*)
  - C. bingkai (*frame*)
  - D. relasi asosiasi
- 6) *Guard condition* di dalam *behavioral state machine diagram* dilambangkan dengan ....
  - A. =
  - B. ()
  - C. []
  - D. /
- 7) Dalam menggambar *behavioral state machine diagram* suatu *class*, sumber informasi yang dapat digunakan dalam mengidentifikasi berbagai *state* yang dimiliki oleh objek, *kecuali* ....
  - A. *best practice*
  - B. mempelajari aliran peristiwa normal pada deskripsi *use case*
  - C. wawancara/berdiskusi dengan pengguna sistem
  - D. mempelajari proses bisnis sistem
- 8) Notasi *behavioral state machine diagram* di bawah ini menunjukkan ....
  - A. *initial state*
  - B. *exit state*
  - C. *stop state*
  - D. *final state*
- 9) *Behavioral state machine diagram* tidak dapat merepresentasikan pesan bersyarat. Pernyataan ini ....
  - A. Benar
  - B. Salah

- 10) Dalam penggambaran *behavioral state machines diagram*, tidak dibenarkan untuk menambah objek baru (selain objek-objek yang telah diidentifikasi selama pengembangan model struktural dan selama proses pembuatan *sequence diagram*). Pernyataan ini ....
- A. Benar  
B. Salah

Cocokkanlah jawaban Anda dengan Kunci Jawaban Tes Formatif 2 yang terdapat di bagian akhir modul ini. Hitunglah jawaban yang benar. Kemudian, gunakan rumus berikut untuk mengetahui tingkat penguasaan Anda terhadap materi Kegiatan Belajar 2.

$$\text{Tingkat Penguasaan} = \frac{\text{Jumlah Jawaban yang Benar}}{\text{Jumlah Soal}} \times 100$$

Arti tingkat penguasaan



Apabila mencapai tingkat penguasaan 80% atau lebih, Anda dapat meneruskan dengan modul selanjutnya. **Bagus!** Jika masih di bawah 80%, Anda harus mengulangi materi Kegiatan Belajar 2, terutama bagian yang belum dikuasai.

## Kunci Jawaban Tes Formatif

### *Tes Formatif 1*

- 1) B
- 2) C
- 3) A
- 4) B
- 5) B
- 6) A
- 7) A
- 8) B
- 9) A
- 10) B

### *Tes Formatif 2*

- 1) A
- 2) A
- 3) D
- 4) A
- 5) B
- 6) C
- 7) A
- 8) D
- 9) B
- 10) B

## Glosarium

Boundary Class	: Objek perantara antara user dengan sistem (aplikasi) pada sequence diagram
Call message	: Pesan dari sebuah objek yang ditujukan untuk mengirim operasi ke objek lainnya pada sequence diagram
Control Class	: Objek yang mengkoordinasi atau mengendalikan aktivitas dalam sistem (aplikasi) pada sequence diagram
Create message	: Pesan dari sebuah objek yang ditujukan untuk menciptakan objek lain pada sequence diagram
Entity Class	: Objek berupa media penyimpanan data/informasi yang dapat berupa sebuah database/tabel atau media penyimpanan fisik pada sistem non digital.
Execution occurrence	: Kotak persegi panjang tipis yang dihamparkan sepanjang lifeline (garis hidup) pada sequence diagram, untuk menunjukkan kapan kelas mengirim dan menerima pesan
Final State	: Notasi berbentuk lingkaran yang mengelilingi lingkaran kecil berisi, yang menandakan selesainya aktivitas pada behavioral state machines diagram
Guard condition	: Suatu kondisi (pesan bersyarat) yang harus dipenuhi oleh sebuah pesan yang akan dikirim dari sebuah objek ke objek lainnya pada sequence diagram atau communication diagram
Initial State	: Notasi berbentuk lingkaran kecil berisi, yang merupakan titik di mana suatu objek mulai muncul pada behavioral state machines diagram
Lifeline	: Garis putus-putus vertikal yang terletak di bawah setiap aktor dan atau objek untuk menunjukkan "masa hidup" aktor/objek dari waktu ke waktu pada sequence diagram

#### 7.40 Pemodelan Perilaku (*Behavioral*) Objek

---

Model Perilaku	: Model yang menggambarkan aspek dinamis internal dari suatu sistem
Nilai Kembali	: Pesan balasan yang dihasilkan dari sebuah penyampaian pesan dari sebuah objek ke objek lainnya pada sequence diagram
Object destruction	: Tanda "X" yang disematkan pada ujung lifeline pada sequence diagram, sebagai pertanda sebuah objek akan menghilang dalam prosesnya
Query	: Proses mengambil informasi (sebagian atau seluruhnya) yang dilakukan oleh control class dari entity class, untuk keperluan verifikasi suatu input yang diberikan ke dalam aplikasi pada sequence diagram
State	: Keadaan suatu objek dan bagaimana objek tersebut berubah berdasarkan kejadian internal ataupun eksternal
Transition	: Keadaan yang merepresentasikan pergerakan suatu objek dari satu keadaan ke keadaan lain pada behavioral state machines diagram
Trigger-signature	: Event tunggal yang memicu perubahan pada sebuah state pada behavioral state machines diagram

## Daftar Pustaka

Dennis, A., Wixom, B.H., & Tegarden, D. (2009). *Systems analysis & design with UML Version 2.0*. Third edition. United States of America: John Wiley & Sons, Inc.

Dennis, A., Wixom, B.H., Roth, R.M. (2012). *Systems analysis & design*. 5th Edition. United States of America: John Wiley & Sons, Inc.

Fowler, M. (2005). *UML distilled*. Edisi 3. Yogyakarta: ANDI.

Rosa, A.S., & Salahuddin, M. (2011). *Rekayasa perangkat lunak (terstruktur dan berorientasi objek)*. Bandung: Modula.

Hermawan, J. (2004). *Analisis, desain, dan pemrograman berorientasi objek dengan UML*. Yogyakarta: ANDI.

**MSIM4302**  
**Edisi 1**

## **MODUL 08**

# **Transisi dari Analisis Sistem ke Desain/Perancangan Sistem**

Bahar, S.T., M.Kom.

## Daftar Isi

### Modul 08 8.1

Transisi dari Analisis Sistem ke Desain/Perancangan Sistem

#### Kegiatan Belajar 1

Transformasi *Entity Relationship Diagram* ke Diagram Relasi

Latihan

Rangkuman

Tes Formatif 1

8.4

Latihan

Rangkuman

Tes Formatif 2

8.27

8.29

8.29

#### Kegiatan Belajar 2

Strategi Akuisisi Sistem Informasi

8.33

Latihan

Rangkuman

Tes Formatif 2

8.42

8.43

8.44

Kunci Jawaban Tes Formatif

Glosarium

Daftar Pustaka

8.47

8.48

8.50



## Pendahuluan

Tujuan dari fase analisis adalah untuk mengetahui apa yang dibutuhkan bisnis, sedangkan tujuan dari fase desain adalah untuk memutuskan bagaimana membangunnya. Pada fase awal desain, tim proyek mengubah kebutuhan bisnis untuk sistem menjadi kebutuhan sistem yang menjelaskan detail teknis untuk membangun sistem. Tidak seperti kebutuhan bisnis, yang tercantum dalam definisi kebutuhan dan dikomunikasikan melalui *use case* dan proses-proses logik serta model data, kebutuhan sistem dikomunikasikan melalui kumpulan dokumen desain dan proses fisik serta model data. Secara bersama-sama, dokumen desain dan model fisik membentuk cetak biru untuk sistem baru.

Pemodelan data secara logika dikaji secara meluas dalam disiplin ilmu tersendiri, yaitu Perancangan Basis Data, sehingga pada buku ini tidak akan dibahas secara mendetail. Pembahasan hanya difokuskan pada bagaimana konsep peralihan model data logik menjadi model data fisik (model data fisik dibahas pada fase desain), yaitu bagaimana mentransformasi (memetakan) *Entity Relationship Diagram* (ERD) sebagai suatu model data logik menjadi diagram relasi yang merupakan model transisi menuju model data fisik.

Fase desain memiliki sejumlah aktivitas yang mengarah ke cetak biru sistem. Bagian awal yang penting dari fase desain (yang merupakan fase transisi dari fase analisis ke fase desain) adalah pemeriksaan beberapa strategi akuisisi sistem untuk memutuskan mana yang akan digunakan untuk memenuhi kebutuhan sistem. Sistem dapat dibangun dari awal, dibeli, dan disesuaikan, atau dialihdayakan kepada orang lain, dan tim proyek perlu menyelidiki kelangsungan hidup setiap alternatif. Keputusan untuk membuat, membeli, atau melakukan *outsourcing* memengaruhi tugas desain yang dilakukan selama sisa fase.

Setelah mempelajari modul ini mahasiswa diharapkan secara lebih rinci mampu:

1. menjelaskan batasan antara fase analisis sistem dan fase desain sistem;
2. menjelaskan elemen-elemen dasar ERD;
3. membaca ERD;
4. mentransformasi/memetakan ERD ke Diagram Relasi;
5. menjelaskan strategi-strategi akuisisi sistem informasi;
6. menjelaskan kelebihan dan kekurangan masing-masing strategi akuisisi sistem informasi.

# Transformasi *Entity Relationship Diagram* ke Diagram Relasi

Fase peralihan dari model data logik ke model data fisik dilakukan dengan membangun skema basis data tertentu. Model data relasional adalah model yang paling umum digunakan saat ini, yang dikenal dengan istilah model Basis Data Relasional. Beberapa referensi mendefinisikan ERD (*Entity Relationship Diagram*) sebagai model data konseptual dan diagram relasi sebagai model data logik (Nugroho, 2004). Referensi lainnya mendefinisikan ERD sebagai model data logik (Dennis, 2012).

Transformasi/pemetaan ERD ke diagram relasi merupakan proses yang mekanis; dalam arti proses ini memiliki aturan-aturan tertentu. Beberapa *tool CASE* (*Computer Aided Software Engineering*) dapat melakukan proses transformasi secara otomatis, namun demikian, kita perlu mengetahui prosedurnya secara manual dengan beberapa alasan (Nugroho, 2004):

1. CASE tidak dapat memodelkan data yang kompleks seperti relasi *ternary* dan relasi *supertype/subtype*. Dengan demikian, kita perlu melakukan transformasi secara manual.
2. Dalam beberapa kasus, terdapat beberapa alternatif transformasi untuk sebuah kasus tertentu, sehingga kita dapat memilih proses yang yang paling efektif untuk menghasilkan diagram relasi yang optimal.
3. Kadangkala hasil proses yang disajikan oleh *tool CASE* perlu diuji secara manual.

Modul ini akan membahas langkah-langkah transformasi ERD ke skema relasi. Masukan dari proses ini adalah diagram E-R, sedangkan keluarannya adalah skema-skema relasional. Dengan demikian, untuk keterpaduan pembahasan, pada bagian awal pembahasan terlebih dahulu dibahas secara umum mengenai model ERD.

## A. PEMODELAN DATA SECARA LOGIKA DENGAN ERD

*Entity Relationship Diagram* (ERD) adalah gambar yang menunjukkan informasi yang dibuat, disimpan, dan digunakan oleh sistem bisnis. Seorang analis dapat membaca ERD untuk menemukan potongan-potongan informasi dalam suatu sistem dan bagaimana informasi tersebut diatur dan saling terkait satu sama lain. Pada ERD, jenis informasi serupa dicantumkan bersama dan ditempatkan di dalam kotak yang disebut

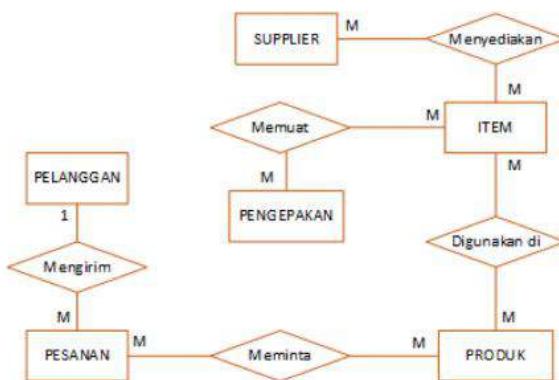
entitas. Garis dibuat antara entitas untuk merepresentasikan hubungan antar data, dan simbol khusus ditambahkan ke diagram untuk mengkomunikasikan aturan bisnis yang perlu didukung oleh sistem. ERD tidak menyiratkan ada urutan proses, meskipun entitas yang terkait satu sama lain biasanya ditempatkan berdekatan.

### 1. Membaca ERD

Gambar 8.1 merupakan contoh penggambaran model data untuk perusahaan manufaktur kecil (Nugroho, 2004). Perusahaan ini membeli item-item dari beberapa supplier yang berbeda yang kemudian mengirimkan item-item itu ke pabrik. Item-item itu kemudian akan dirakit menjadi produk tertentu yang akan dijual ke pelanggan. Setiap pelanggan mungkin saja memiliki satu atau lebih pesanan untuk produk-produk tertentu.

Diagram pada Gambar 8.1 memperlihatkan himpunan entitas-entitas serta hubungan (relasi) satu sama lain (untuk saat ini demi kesederhanaan kita mengabaikan atribut-atribut untuk masing-masing himpunan entitas). Himpunan entitas digambarkan dalam bentuk empat persegi panjang sementara jajaran genjang menunjukkan relasi-relasi. Entitas-entitas pada Gambar 8.1 adalah sebagai berikut.

- Pelanggan.** Perorangan atau organisasi yang sudah dan akan memesan produk.
- Produk.** Produk tertentu yang mungkin akan dan telah dipesan oleh pelanggan.
- Pesanan.** Transaksi yang berhubungan dengan penjualan satu atau lebih produk ke pelanggan dan diidentifikasi dengan nomor transaksi.
- Item.** Tipe komponen yang digunakan untuk merakit suatu produk.
- Supplier.** Organisasi atau perusahaan lain (juga perorangan) yang mungkin menyediakan item tertentu bagi perusahaan.
- Pengepakan.** Item-item yang dikirimkan dalam paket-paket tertentu dari supplier ke perusahaan.



Gambar 8.1  
Contoh *Entity Relationship Diagram* (ERD)

Perhatikan bahwa sangat penting untuk secara jelas mendefinisikan setiap jenis entitas dengan apa yang telah kita kenali sebagai metadata, yaitu data yang menjelaskan data yang lain. Sebagai contoh, adalah penting untuk mengetahui bahwa entitas **pelanggan** mencakup juga perorangan atau organisasi yang belum membeli produk ke perusahaan. Dalam banyak kasus, adalah umum bahwa setiap unit dalam organisasi memberi makna yang tidak sama untuk suatu terminologi yang sama. Sebagai contoh, Departemen Akuntansi mungkin mendefinisikan **pelanggan** sebagai perorangan atau organisasi yang telah melakukan pembelian, sedangkan Departemen Pemasaran mungkin mendefinisikan **pelanggan** sebagai perorangan atau organisasi yang memiliki kontak dengan perusahaan atau yang telah melakukan pembelian dari perusahaan kita atau dari perusahaan pesaing kita. Model ERD yang baik seharusnya dilengkapi dengan metadata supaya penafsirannya benar dan akurat.

Simbol pada setiap garis pada ERD mencerminkan kardinalitas relasi (kardinalitas adalah jumlah suatu entitas yang berelasi dengan entitas yang lainnya). Dengan memperhatikan Gambar 8.1, kita dapat memberi pernyataan-pernyataan sebagai berikut.

- a. **Supplier** dapat menyediakan banyak **item** (kata *dapat* berarti mungkin saja supplier tidak menyediakan item apapun). Setiap item harus disediakan oleh supplier (kata *harus disediakan* berarti item disediakan oleh paling sedikit satu supplier).
- b. Setiap **item** harus digunakan pada paling sedikit satu **produk**, dan mungkin digunakan pada beberapa **produk**. Kebalikannya, setiap **produk** harus menggunakan satu atau lebih **item**.
- c. Setiap **pengepakan** harus memuat satu atau lebih **item**. Sebuah item mungkin termuat dalam beberapa **pengepakan**.
- d. **Pelanggan** mungkin mengirim lebih dari satu **pesanan**. Bagaimanapun juga, setiap pesanan harus dikirim oleh satu (dan hanya satu) **pelanggan**.
- e. **Pesanan** harus meminta satu atau lebih **produk**. Suatu **produk** mungkin tidak diminta oleh satu **pesanan** pun, atau mungkin juga diminta pada satu atau lebih **pesanan**.

Notasi ERD secara lengkap digambarkan pada Gambar 8.2. Konstruksi dasar dari model ERD terdiri dari entitas, relasi, serta atribut. Seperti yang disajikan pada Gambar 8.2, ada beberapa variasi untuk tiap-tiap konstruksi dasar itu. Keragaman dari model ERD mengijinkan perancang untuk dapat memodelkan situasi dunia nyata dengan akurat, yang pada gilirannya meningkatkan popularitas model ERD ini.

## 2. Elemen-elemen ERD

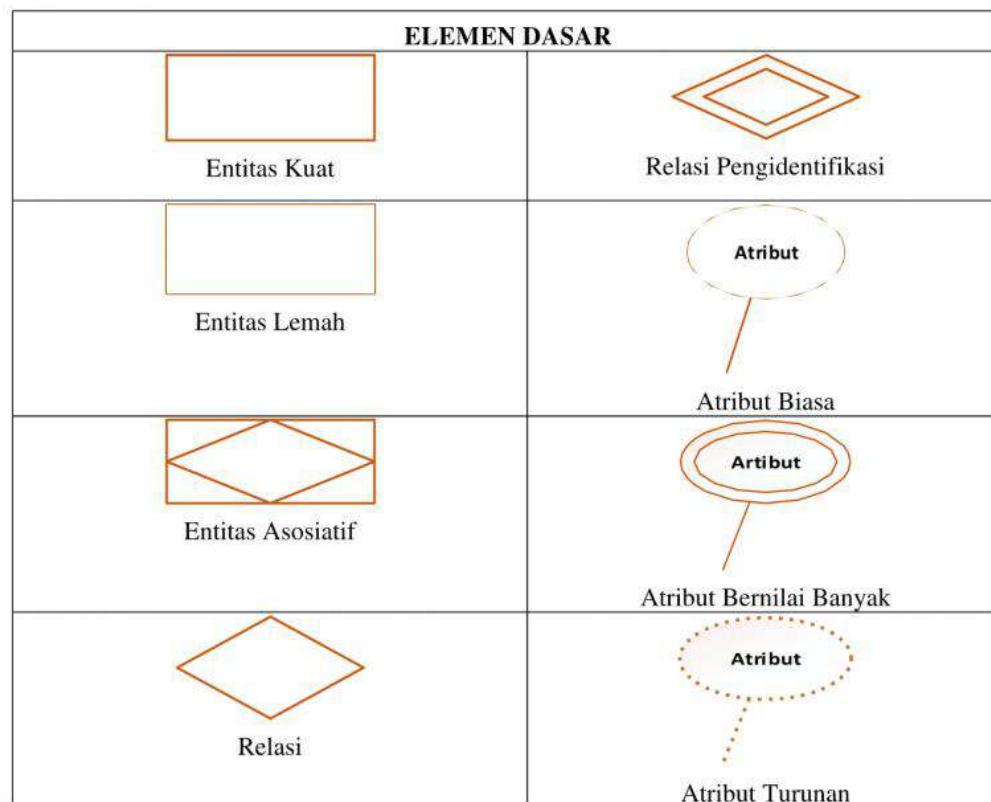
Ada tiga elemen dasar dalam ERD sebagai bahasa pemodelan data (entitas, atribut, dan relasi/hubungan), yang masing-masing diwakili oleh simbol grafik yang berbeda. Ada beberapa model simbol berbeda yang dapat digunakan pada ERD, seperti:

model IDEF1X, model Chen, dan model Crow's Foot (Dennis, 2012). Tidak ada satu set simbol yang mendominasi penggunaan di industri, dan tidak ada yang lebih baik dari yang lain. Pembahasan pada modul ini akan menggunakan model *Chen*. Gambar 8.2 merangkum tiga elemen dasar ERD dan varian simbol yang akan digunakan.

a. *Entitas*

Entitas adalah orang, tempat, objek kejadian, atau konsep dalam lingkup pengguna yang oleh organisasi perlu dipelihara datanya. Beberapa contoh entitas disajikan berikut.

- 1) Entitas *orang*: Mahasiswa, Karyawan, Dosen, Pemain Musik, dan sebagainya
- 2) Entitas *tempat*: Kota, Jalan, Negara, Provinsi, dan sebagainya
- 3) Entitas *objek*: Mesin, Mobil, Gedung, Pesawat Terbang, dan sebagainya
- 4) Entitas *kejadian*: Penjualan, Pembelian, Registrasi Mahasiswa, dan sebagainya
- 5) Entitas *konsep*: Kuliah, Kursus, Mata Kuliah, dan sebagainya



Gambar 8.2  
Elemen-elemen ERD

## 8.8 Transisi dari Analisis Sistem Ke Desain/Perancangan Sistem

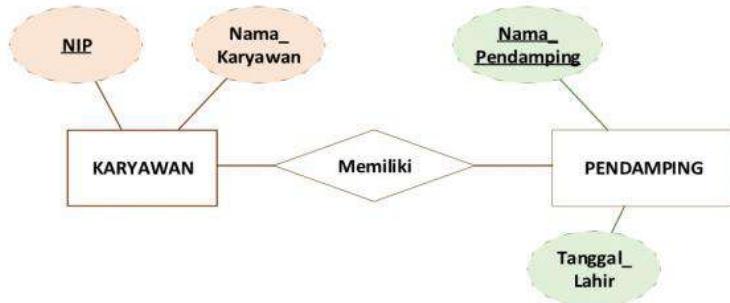
### 1) Entitas Kuat dan Entitas Lemah

Kebanyakan entitas dalam suatu organisasi dapat digolongkan sebagai entitas kuat (*strong entity*) yaitu entitas yang mandiri, yang keberadaannya tidak bergantung pada keberadaan entitas yang lainnya. Instansiasi entitas kuat selalu memiliki karakteristik yang unik (dinamakan *identifier* atau sering disebut sebagai pengidentifikasi, atau lebih tepat atribut pengidentifikasi) yaitu, sebuah atribut tunggal atau gabungan atribut yang secara unik dapat digunakan untuk membedakannya dari entitas kuat yang lain.

Secara berlawanan, dapat didefinisikan bahwa entitas lemah (*weak entity*) adalah entitas yang keberadaannya sangat bergantung pada keberadaan entitas yang lainnya. Entitas lemah tidak memiliki arti apa-apa dan tidak dikehendaki kehadirannya dalam diagram ERD tanpa kehadiran entitas di mana mereka bergantung. Entitas di mana entitas lemah bergantung dinamakan *identifying owner* (atau lebih sering dinamakan *owner/pemilik*).

Entitas lemah tidak memiliki *identifier*-nya sendiri. Secara umum, dalam diagram ERD entitas lemah memiliki atribut yang berperan sebagai *partial identifier* (*identifier* yang berfungsi secara sebagian). Selama fase perancangan, *identifier* penuh akan diberikan pada entitas lemah dengan cara mengkombinasikan *partial identifier* dengan *identifier* pemiliknya (*owner*).

Contoh:



Gambar 8.3  
Entitas Kuat dan Entitas Lemah

Pada Gambar 8.3, *Karyawan* adalah entitas kuat dengan pengidentifikasi *NIP* (dituliskan dengan garis di bawahnya). *Pendamping* adalah sebuah entitas lemah (digambarkan dengan kotak dengan garis ganda), yang berarti istri atau suami seorang *Karyawan*. Relasi antara entitas lemah dan entitas kuat dinamakan *relasi pengidentifikasi*. *Memiliki* adalah *relasi pengidentifikasi* yang digambarkan dengan jajaran genjang dengan garis ganda. Atribut nama *Pendamping* berfungsi sebagai *pengidentifikasi sebagian* (*partial identifier*). Kelak pada tahap perancangan, *Nama*

*Pendamping* akan dikombinasikan dengan *NIP* untuk membentuk pengidentifikasi penuh untuk entitas *Pendamping*.

## 2) Entitas Asosiatif

Dalam beberapa kasus, suatu entitas mungkin terbentuk dari suatu relasi. Jika ini terjadi entitas yang dihasilkan dinamakan dengan entitas asosiatif. Entitas ini hanya terbentuk oleh relasi tertentu; ia tidak berdiri sendiri secara mandiri.

Contoh:



Gambar 8.4  
Entitas Asosiatif

ERD pada Gambar 8.4 memiliki notasi yang baru yaitu *entitas asosiatif* (jajaran genjang dengan tulisan *ijazah* dalam kotak empat persegi panjang). Perhatikan bahwa relasi yang terjadi sebenarnya adalah *mahasiswa mengambil kuliah*. Tetapi, kita tahu pada akhir suatu masa perkuliahan dan jika memenuhi syarat akademis tertentu (misalnya jumlah SKS yang telah diambil minimal 144 SKS dengan IPK minimal 2 dan jumlah nilai D tidak lebih dari 12 SKS), mahasiswa yang bersangkutan akan mendapatkan *ijazah* (*ijazah* adalah entitas sebab ia hadir baik secara fisik maupun konseptual). Dengan kata lain, *ijazah* hanya dapat diperoleh setelah seorang mahasiswa mengambil (dan menyelesaikan) kuliah; lain hal tidak, misalnya jika seorang mahasiswa meninggalkan perkuliahan tanpa menyelesaiannya, ia tidak akan mendapatkan *ijazah*. Terlihat di sini, bahwa entitas *ijazah* muncul dari relasi *mahasiswa mengambil kuliah*; lain hal tidak, sehingga dapat dikatakan bahwa entitas *ijazah* adalah *entitas asosiatif*.

### b. Atribut

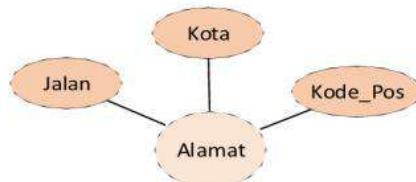
Setiap entitas memiliki karakteristik yang berasosiasi dengannya. Atribut adalah properti atau karakteristik yang dimiliki oleh suatu entitas di mana properti arti atau karakteristik itu bermakna bagi organisasi. Misalnya, untuk pencatatan data mahasiswa di suatu universitas, entitas *Mahasiswa* mungkin memiliki atribut-atribut *NIM*, *Nama*, *Alamat*. Atribut-atribut ini adalah atribut yang paling bermakna untuk pencatatan mahasiswa di Universitas. Atribut lainnya seperti *Tinggi\_Badan*, *Berat\_Badan* tidak memiliki makna bagi universitas sehingga mereka tidak perlu dicantumkan pada pencatatan data *Mahasiswa* di universitas. Lain halnya jika pencatatan dilakukan untuk *event* olah raga tertentu (pertandingan gulat atau basket antar universitas), mungkin atribut *Tinggi\_Badan* dan *Berat\_Badan* perlu dicatat.

Basis data sebenarnya adalah kumpulan dari nilai-nilai atribut untuk semua entitas yang terlibat pada proses bisnis dalam organisasi. Masing-masing entitas akan dibedakan dengan entitas yang lainnya berdasarkan nilai pengidentifikasinya (*identifier*). Pada Gambar 8.3, *NIP* dan *Nama\_Karyawan* adalah atribut dari entitas *Karyawan*.

1) Atribut Komposit

Beberapa atribut dapat dipecah (didekomposisi) menjadi beberapa komponen. Suatu contoh yang paling umum adalah atribut *Alamat*. Atribut *Alamat* dapat dipecah menjadi atribut *Jalan*, *Kota*, serta *Kode\_Pos*. Atribut komposit adalah atribut yang dapat dipecah menjadi atribut-atribut yang lainnya.

Contoh:



Gambar 8.5  
Atribut Komposit

Atribut komposit menyediakan fleksibilitas lebih besar. Jenis atribut ini digunakan pada basis data untuk kemudahan mencari informasi spesifik (tertentu). Misalnya pada Gambar 8.5, jika pada basis data dilakukan pencarian/permintaan informasi yang berkaitan dengan pemecahan atribut komposit, misalnya pertanyaan: Berapa orang karyawan yang tinggal di kota Bandung? Atau berapa orang Mahasiswa yang Kos di Jalan Pemuda? Maka pemecahan (dekomposisi) itu memang perlu dilakukan. Akan tetapi, jika diperkirakan operasi-operasi tersebut tidak pernah muncul pada basis data, lebih baik jika atribut-atribut tersebut tidak didekomposisi. Dekomposisi atribut sering memunculkan permasalahan waktu pemrosesan yang lebih lama, serta penggunaan ruang penyimpanan yang besar.

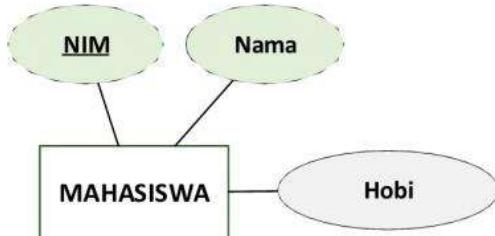
Jika suatu atribut tidak memerlukan proses dekomposisi lebih lanjut, maka atribut tersebut dinamakan **atribut sederhana** (sering hanya disebut atribut). Contoh atribut sederhana untuk kasus entitas Mahasiswa pada Gambar 8.4 adalah *No\_Telepon* (terdiri atas *Kode\_Negara*, *Kode\_Wilayah*, dan *Nomor\_Telepon*). Atribut tersebut tidak perlu didekomposisi lebih lanjut karena tidak ada manfaatnya (kecuali jika kita ingin menyimpan data *Kode\_Area* secara terpisah dengan nomor telepon setiap mahasiswa).

2) Atribut Bernilai Banyak

Pada umumnya setiap atribut adalah bernilai tunggal. Misalnya, seorang Mahasiswa tentunya memiliki satu nilai untuk atribut *NIM* (dinamakan **atribut**

**tunggal/single attribute**). Tetapi, ada kasus-kasus tertentu di mana atribut memiliki nilai lebih dari satu untuk suatu entitas tertentu. Dalam hal yang terakhir ini, atribut yang bersangkutan dinamakan **atribut bernilai banyak (multi-value attribute)**.

Contoh:



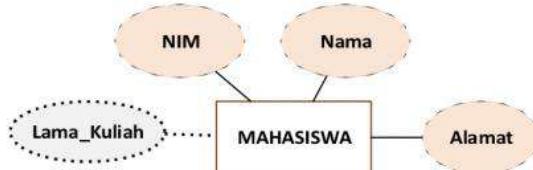
Gambar 8.6  
Atribut Bernilai Banyak

Pada Gambar 8.6, dapat dilihat bahwa atribut *Hobi* digambarkan dengan elips berbatas ganda, yang menandakan *atribut bernilai banyak*. Dikatakan atribut bernilai banyak karena seorang Mahasiswa dapat memiliki beberapa Hobi/kegemaran, misalnya Berenang, Bermain Sepak Bola, dan Membaca.

### 3) Atribut Turunan

Ada beberapa atribut yang nilainya dapat diperoleh dari atribut yang lain. Misalnya, atribut *NIM* (Nomor Induk Mahasiswa) pada entitas *Mahasiswa* pada umumnya mengandung unsur *Tahun Masuk*, *Jurusan*, dan *Nomor Registrasi*. Pada Gambar 8.7, misalnya seorang *Mahasiswa* memiliki *NIM = 312018025555*, berarti Mahasiswa yang bersangkutan masuk pada tahun 2018. Jika kita menambahkan atribut *Lama\_Kuliah*, maka nilai atribut *Lama\_Kuliah* adalah pengurangan antara **tahun saat proses perhitungan dilakukan** dengan **tahun Mahasiswa yang bersangkutan masuk** (yang diidentifikasi langsung dari *NIM*).

Contoh:



Gambar 8.7  
Atribut Turunan

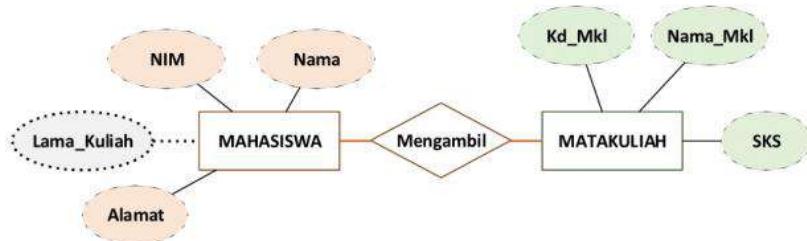
Jika hasil analisis kebutuhan pada fase analisis sistem menunjukkan banyak kasus perhitungan seperti kasus *Lama\_Kuliah* pada Gambar 8.7, perancang sistem perlu mempertimbangkan untuk menggunakan model atribut turunan. Meski model atribut

turunan kelihatannya menggunakan banyak ruang penyimpanan, namun akan mempercepat proses komputasi saat atribut-atribut turunan digunakan secara intensif dalam aplikasi-aplikasi tertentu.

### c. Relasi

Relasi adalah asosiasi ‘yang berarti’ antara suatu entitas dengan entitas yang lainnya. Kata ‘yang berarti’ menunjukkan bahwa relasi dapat digunakan untuk melayani permintaan data atau informasi yang tidak dapat disediakan hanya dengan kehadiran satu entitas (secara mandiri). Relasi digambarkan dengan *jajaran genjang* berisi *label* yang berupa *kata kerja* yang pendek. Misalnya, terdapat permintaan informasi mengenai *Matakuliah* apa saja yang diprogramkan oleh seorang *Mahasiswa* tertentu pada semester tertentu? Entitas *Mahasiswa* tidak dapat menyediakan informasi yang utuh atas pertanyaan tersebut, melainkan harus menciptakan sebuah relasi ke entitas *Matakuliah*.

Contoh sebuah relasi sederhana antara entitas *Mahasiswa* dengan entitas *Matakuliah* disajikan pada Gambar 8.8.



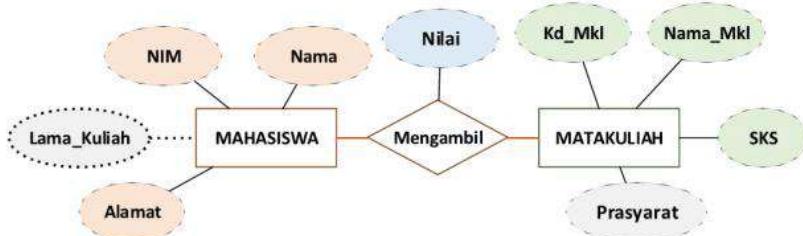
Gambar 8.8  
Relasi

Instansiasi relasi adalah asosiasi antara satu atau lebih relasi. Misalnya relasi Mahasiswa Mengambil Mata Kuliah pada Gambar 8.8, instansiasinya mungkin dapat berupa: *Arisandi Mengambil Pemodelan Software*, atau *Nurmilawati Mengambil Basis Data*.

#### 1) Atribut pada Relasi

Sebuah atribut tidak hanya dapat berasosiasi dengan sebuah entitas, melainkan dapat juga berasosiasi dengan sebuah relasi. Atribut yang berasosiasi dengan sebuah relasi disebut atribut relasi.

Contoh:



Gambar 8.9  
Atribut Relasi

Relasi *Mahasiswa mengambil Matakuliah* pada Gambar 8.9 akan menghasilkan sebuah atribut *Nilai*. Pertanyaannya adalah: "Di mana atribut *Nilai* akan ditempatkan? Jika ditempelkan pada entitas *Mahasiswa* (*Nilai* merupakan atribut dari entitas *Mahasiswa*), maka berarti semua *Matakuliah* yang diambil oleh seorang *Mahasiswa* menghasilkan *Nilai* yang sama; ini tidak realistik. Jika atribut *Nilai* ditempelkan pada entitas *Matakuliah* (*Nilai* merupakan atribut dari entitas *Mahasiswa*), maka berarti semua *Mahasiswa* yang mengambil *Matakuliah* tertentu akan memiliki nilai yang sama; ini juga tidak realistik. Jadi, konsep yang tepat adalah: atribut *Nilai* harus ditempelkan pada relasi *Mengambil*, dengan model relasi 'banyak ke banyak'. Jenis relasi banyak ke banyak (pada relasi *Mengambil*) akan menyebabkan *seorang Mahasiswa* atau *beberapa Mahasiswa* yang **Mengambil** satu atau lebih dari satu *Matakuliah* dapat memunculkan *Nilai* yang berbeda (secara berulang).

Atribut yang menempel pada suatu relasi (atribut relasi) dapat lebih dari satu jumlahnya. Contoh: pada relasi *Mahasiswa Mengambil Matakuliah*, selain atribut *Nilai* yang dapat menempel pada relasi **Mengambil**, dapat juga menempel atribut lain seperti *Dosen\_Penguji* dan *Tanggal\_Ujian*.

## 2) Derajat Relasi

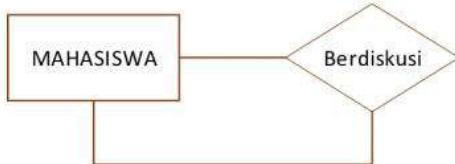
Derajat relasi adalah jumlah entitas yang berpartisipasi pada suatu relasi. Pada relasi *Mahasiswa Mengambil Matakuliah*, derajat relasinya adalah 'derajat 2' sebab terdapat dua entitas yang terlibat dalam relasi **Mengambil** (entitas *Mahasiswa* dan entitas *Matakuliah*).

Derajat-derajat relasi yang umum dijumpai pada penggambaran ERD adalah: derajat 1 (*unary*), derajat 2 (*binary*), dan derajat 3 (*ternary*). Relasi dengan derajat relasi lebih dari 3 secara teoritis dimungkinkan, tetapi pada kenyataanya sangat jarang dijumpai di dunia nyata (relatif sulit untuk diimplementasikan di sistem basis data) (Nugroho, 2004).

### a) Unary Relationship

Relasi derajat 1 (*unary relationship*) adalah relasi yang hanya melibatkan satu entitas. Jenis relasi ini dinamakan juga relasi rekursif.

Contoh:



Gambar 8.10  
Relasi Berderajat 1

Pada Gambar 8.10, relasi **Berdiskusi** menghubungkan setiap instansiasi entitas *Mahasiswa* satu (atau sekelompok mahasiswa) dengan yang lainnya. Jika diterjemahkan akan menjadi: seorang (sekelompok) *Mahasiswa Berdiskusi* dengan *Mahasiswa* (kelompok Mahasiswa) yang lain.

b) Binary Relationship

Relasi derajat 2 (binary relationship) adalah relasi yang melibatkan dua entitas. Gambar 8.8 adalah contoh relasi berderajat 2.

c) Ternary Relationship

Relasi derajat 3 (ternary relationship) adalah relasi yang melibatkan/menyambungkan tiga entitas dalam satu relasi. Dengan demikian, relasi derajat 3 tidaklah sama maknanya dengan 3 relasi biner.

Contoh:



Gambar 8.11  
Relasi Berderajat 3

Jika diterjemahkan, relasi berderajat 3 pada Gambar 8.11 menjadi: *Mahasiswa Mengambil* (mengikuti) *Matakuliah* tertentu, menempati *Ruang Kuliah* tertentu. Pada kasus ini, relasi derajat 3 hanya berlaku jika dalam Prosedur Bisnis diasumsikan setiap *Matakuliah* tidak ditentukan untuk menempati *Ruang Kuliah* tertentu (secara pasti). Artinya, suatu *Matakuliah* tertentu diperbolehkan menggunakan *Ruang Kuliah* tertentu

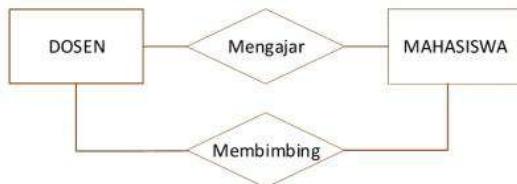
secara bebas. Mengapa demikian? Ketika seorang *Mahasiswa Mengambil* suatu *Matakuliah* tertentu, pada saat itu tidak secara otomatis dapat diketahui *Matakuliah* tersebut akan diselenggarakan pada *Ruang Kuliah* apa, hingga harus ditentukan secara manual.

Namun demikian, jika diasumsikan setiap *Matakuliah* telah ditentukan untuk menempati sebuah *Ruang Kuliah* secara pasti (suatu *Matakuliah* hanya dapat menempati satu *Ruang Kuliah* yang telah ditentukan), maka relasi berderajat 3 tidak dapat diterapkan pada kasus ini. Relasi harus didesain dalam bentuk relasi derajat 2, di mana entitas *Ruang Kuliah* diubah menjadi atribut yang menempel pada entitas *Matakuliah*, atau entitas *Ruang Kuliah* membentuk sebuah relasi baru yang terhubung ke entitas *Matakuliah*. Mengapa demikian? Ketika seorang *Mahasiswa Mengambil* suatu *Matakuliah* tertentu, pada saat itu secara otomatis dapat diketahui *Matakuliah* tersebut akan diselenggarakan pada *Ruang Kuliah* apa, tanpa harus ditentukan secara manual (atribut *Kd\_Mkl* akan secara otomatis mengenali atribut *Nm\_Ruang* di mana suatu *Matakuliah* tertentu akan diselenggarakan).

### 3) Relasi Ganda

Dalam proses bisnis, mungkin perlu menggambarkan relasi lebih dari satu (disebut relasi ganda/*multiple relationship*) antara entitas-entitas yang sama.

Contoh:



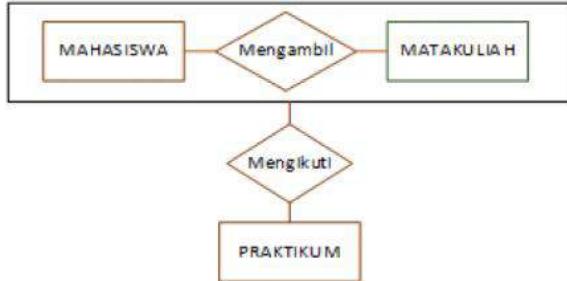
Gambar 8.12  
Relasi Ganda

Jika diterjemahkan, relasi ganda pada Gambar 8.12 menjadi: *Dosen Mengajar Mahasiswa* (dalam sebuah perkuliahan) dan pada saat yang sama Dosen juga dapat *Membimbing Mahasiswa* (misalnya dalam penyelesaian Laporan Tugas Akhir atau sebagai *Pembimbing Akademik Mahasiswa*).

### 4) Agregasi

Agregasi adalah suatu keadaan di mana suatu relasi hanya dapat direalisasikan setelah relasi yang lain tercipta/terpenuhi terlebih dahulu (memiliki relasi prasyarat). Relasi bertipe agregasi tidak dimungkinkan ada jika relasi yang menjadi prasyaratnya tidak terealisasi.

*Contoh:*



Gambar 8.13  
Agregasi

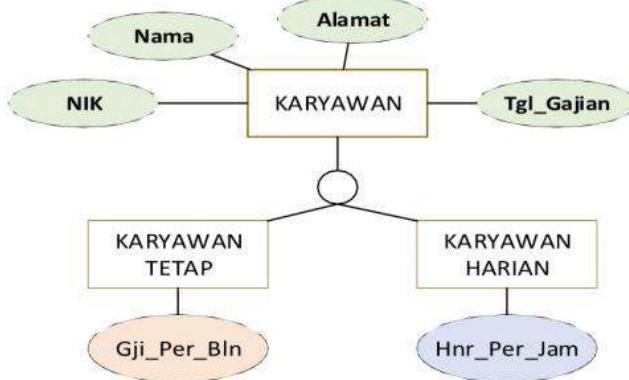
Pada Gambar 8.13, misalkan seorang **Mahasiswa Mengambil Matakuliah Pemrograman**, di mana kegiatan perkuliahan untuk *Matakuliah* tersebut terdiri dari Perkuliahan *Teori di kelas* dan perkuliahan *Praktikum di Laboratorium*. Jika Mahasiswa yang bersangkutan *tidak mengambil Matakuliah (Teori) Pemrograman* maka ia tidak diperkenankan *mengambil/mengikuti Praktikum Pemrograman*. Pada Gambar 8.13, relasi yang menjadi prasyarat adalah **Mahasiswa Mengambil Matakuliah**, dikelompokkan dalam sebuah kotak yang menggambarkan bahwa relasi itu menjadi satu kesatuan sebelum dihubungkan dengan relasi lainnya (relasi **Mahasiswa Mengikuti Praktikum**).

##### 5) Pewarisan Atribut Menggunakan Konsep Relasi Supertype dan Subtype

Konsep supertype dan subtype adalah salah satu konsep lanjut dalam pemodelan dengan ERD. Konsep lanjut ini dinamakan EER (*Enhanced Entity Relationship*) dan model data yang menggunakan konsep-konsep lanjut ini dinamakan *Diagram EER*.

Pada umumnya konsep-konsep lanjut (seperti supertype dan subtype) tidak terlalu banyak digunakan pada masa-masa awal pemodelan data, namun dengan berkembangnya paradigma pemrograman/pemodelan berorientasi objek, konsep relasi supertype dan subtype mulai digunakan. Selain itu, aturan-aturan bisnis dan tipe-tipe data yang makin kompleks dalam organisasi juga membuat konsep supertype dan subtype makin sering digunakan.

Contoh:



Gambar 8.14  
Relasi Supertype dan Subtype

Pada Gambar 8.14, misalkan suatu perusahaan memiliki dua kategori karyawan yaitu *Karyawan Harian* dan *Karyawan Tetap*, masing-masing atributnya adalah:

- Karyawan Harian*: NIK, Nama, Alamat, Tgl\_Gajian, serta Hnr\_Per\_Jam
- Karyawan Tetap*: NIK, Nama, Alamat, Tgl\_Gajian, dan Gji\_Per\_Bln

Dapat dilihat bahwa kedua kategori Karyawan tersebut memiliki beberapa atribut dengan nama yang sama, yaitu: Nama, Alamat, serta Tgl\_Gajian. Sebagai tambahan, atribut Hnr\_Per\_Jam adalah atribut yang unik untuk Entitas *Karyawan Harian*, sedangkan Atribut yang unik untuk *Karyawan Tetap* adalah Gji\_Per\_Bln. Jika analis dihadapkan pada situasi seperti ini, maka ada beberapa pilihan yang mungkin dilakukan.

- Mendefinisikan suatu entitas tunggal yang dinamakan Karyawan, kemudian menambahkan semua atribut yang terdaftar untuk masing-masing kategori Karyawan. Konsekuensinya, untuk kategori Karyawan tertentu, sejumlah atribut akan kosong (bernilai Null). Hal ini bertentangan dengan konsep desain database yang efektif.
- Mendefinisikan dua entitas yang terpisah: entitas *Karyawan Tetap* dan entitas *Karyawan Harian*. Konsekuensinya, pendekatan ini gagal untuk menghindari redundansi data.
- Mendefinisikan entitas *Karyawan* sebagai supertype dengan *Karyawan Harian* dan *Karyawan Tetap* masing-masing sebagai subtype-nya (seperti pada Gambar 8.14).

Pewarisan atribut adalah keadaan di mana entitas subtype mewarisi nilai-nilai dari semua atribut pada supertype. Pewarisan (biasa juga disebut *inheritance*) merupakan suatu karakteristik penting di mana kita tidak perlu lagi menuliskan atribut-atribut yang ada pada supertype sebagai atribut-atribut pada subtype (lihat kembali

Modul 5). Hal ini memungkinkan untuk meminimalkan redunsansi (pengulangan data yang tidak perlu).

*d. Kardinalitas dan Modalitas*

**Kardinalitas** adalah banyaknya *instance* dari satu entitas yang dikaitkan dengan *instance* dari entitas lainnya (Dennis, 2012). Kardinalitas untuk hubungan biner (hubungan antara dua entitas) adalah 1:1, 1:N, atau M:N, dan kita akan membahas masing-masing secara bergantian. Angka "1" dapat diganti dengan simbol "|/tiang/bar", sedangkan "M" atau "N" dapat diganti dengan "n" (kaki gagak).

Hubungan 1:1 (dibaca sebagai "satu ke satu") berarti bahwa satu *instance* dari entitas induk dikaitkan dengan satu *instance* dari entitas anak.

*Contoh*

Sebuah perusahaan memberikan Satu tempat parkir khusus yang telah dipesan kepada setiap Karyawan yang "berprestasi bulan ini." Sebaliknya Satu tempat parkir tertentu yang telah dipesan, diperuntukkan hanya untuk Satu Karyawan tertentu yang berprestasi.

Hubungan 1:N (dibaca sebagai "satu ke banyak") berarti bahwa satu *instance* dari entitas induk dikaitkan dengan banyak *instance* dari entitas anak, namun *instance* entitas anak hanya terkait dengan satu *instance* induk.

*Contoh*

Seorang Dosen "Penasihat Akademik" dapat membimbing banyak Mahasiswa, namun seorang Mahasiswa hanya boleh dibimbing oleh Satu orang Dosen Penasihat Akademik.

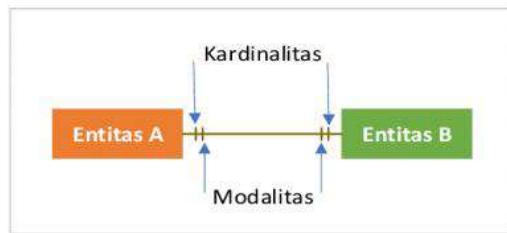
Jenis hubungan ketiga adalah hubungan M:N (dibaca sebagai "banyak ke banyak"). Dalam kasus ini, banyak contoh *instance* induk dapat berhubungan dengan banyak contoh *instance* entitas anak.

*Contoh*

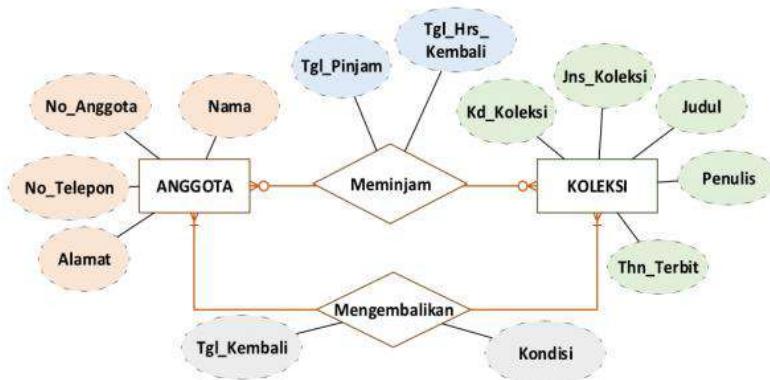
Seorang Dosen dapat Mengampu Banyak Mata Kuliah, dan suatu Mata Kuliah tertentu dapat diampu oleh Banyak Dosen (Kelompok Dosen).

**Modalitas** adalah suatu hubungan untuk menunjukkan apakah *instance* entitas anak atau induk diperlukan (harus) berpartisipasi dalam hubungan tersebut (Dennis, 2012). Nilai modalitas ditampilkan sebagai "0" jika hubungan bersifat *opsional* atau ketika tidak ada kebutuhan. Jika nilai modalitas direpresentasikan sebagai "1" (tiang/bar) maka ada keharusan untuk terjadinya suatu partisipasi. Istilah lain dari modalitas adalah *participation constrain*.

Gambar 8.15 menunjukkan posisi (letak) kardinalitas dan modalitas, sedangkan Gambar 8.16 adalah contoh penggunaan kardinalitas dan modalitas pada ERD.



Gambar 8.15  
Posisi Kardinalitas dan Modalitas

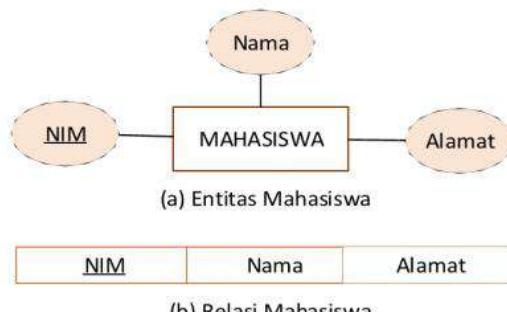


Gambar 8.16  
ERD dengan Kardinalitas dan Modalitas

## B. MENTRANSFORMASI DIAGRAM ER KE DIAGRAM RELASI

### 1. Pemetaan (Transformasi) Entitas Biasa

Setiap entitas biasa dapat dipetakan ke skema relasi. Pemberian nama pada relasi secara umum adalah sama dengan nama entitas. Atribut-atribut pada entitas menjadi atribut-atribut relasi. Atribut pengidentifikasi pada entitas berubah menjadi kunci primer (*primary key*) pada skema relasi. Perhatikan contoh pada Gambar 8.17.

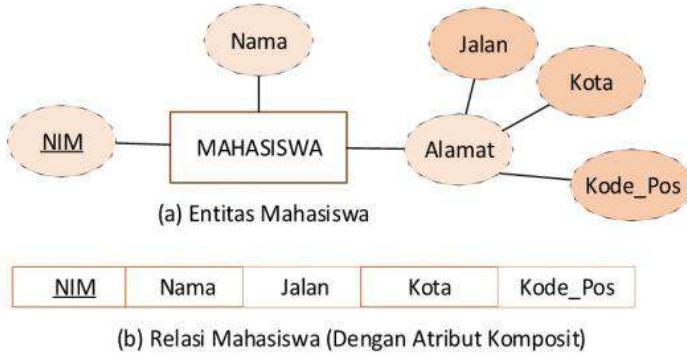


Gambar 8.17  
Pemetaan Entitas Biasa

Untuk penyederhanaan pemahaman, Gambar 8.17(b) tidak menyertakan atribut-atribut seperti yang disajikan pada Gambar 8.17(a).

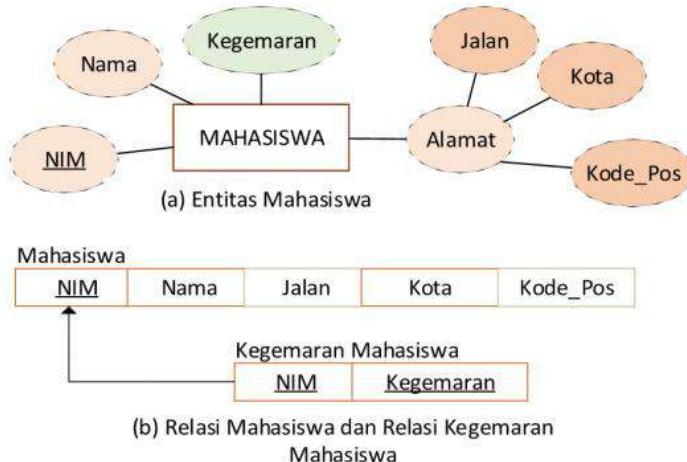
## 2. Pemetaan Atribut Komposit

Atribut komposit dipetakan dengan cara menggabungkannya dengan atribut-atribut biasa, namun tidak menyertakan atribut induk dari atribut komposit tersebut (Gambar 8.18). Pada Gambar 8.18, atribut komposit *Alamat* terdiri atas *Jalan*, *Kota*, dan *Kode\_Pos*.



Gambar 8.18  
Pemetaan Atribut Komposit

## 3. Pemetaan Atribut Bernilai Banyak



Gambar 8.20  
Pemetaan Atribut Bernilai Banyak

Sebuah atribut bernilai banyak (*multivalue attribute*) akan membentuk dua buah relasi. Relasi ke-1 mengandung semua atribut selain atribut bernilai banyak, sedangkan

relasi ke-2 terdiri atas atribut bernilai banyak tersebut ditambah 1 atribut kunci tamu yang terhubung dengan kunci primer pada relasi induknya (lihat Gambar 8.19).

Relasi *Kegemaran Mahasiswa* memiliki atribut bukan kunci primer (sering disebut deskriptor). Setiap baris pada relasi ini mencatat fakta bahwa setiap *Mahasiswa* dapat memiliki banyak *Kegemaran*. Ini menyediakan kesempatan kepada user untuk dapat menambahkan atribut lain untuk mendeskripsikan atribut bernilai banyak (*kegemaran*), misalnya: *Tahun awal mula digelutinya* setiap *Kegemaran* yang ada. Ujung pangkal tanda panah menunjukkan atribut kunci tamu (pada relasi *Kegemaran Mahasiswa*), sedangkan ujung tanda panah mengarah ke atribut kunci primer pada relasi induknya (relasi *Mahasiswa*).

#### 4. Pemetaan Entitas Lemah

Entitas lemah tidak memiliki keberadaan yang mandiri melainkan hanya hadir melalui relasi pengidentifikasi dengan entitas lain yang dinamakan *Entitas Pemilik* (lihat Gambar 8.20).



Gambar 8.20  
Pemetaan Entitas Lemah

Ujung pangkal tanda panah menunjukkan atribut kunci tamu (pada relasi **Pendamping**), sedangkan ujung tanda panah mengarah ke atribut kunci primer pada relasi Pemilik (relasi **Karyawan**).

#### 5. Pemetaan Relasi *Binary*

##### a. Relasi *Satu ke Satu*

Jika terdapat relasi *Binary* dengan kardinalitas *Satu ke Satu*, maka harus dibuat dua relasi yang masing-masing mewakili setiap entitas. Kunci primer dari salah satu

## 8.22 Transisi dari Analisis Sistem Ke Desain/Perancangan Sistem

relasi dicantumkan sebagai kunci tamu pada relasi yang lainnya. Persoalannya adalah kunci primer dari relasi yang mana yang harus menjadi kunci tamu pada relasi yang lainnya? Untuk kasus seperti ini, kita harus memperhatikan modalitas yang tercipta.



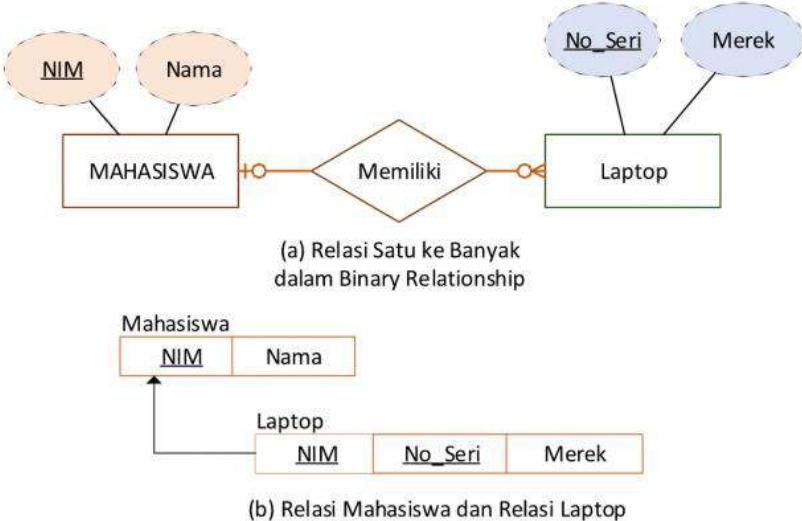
Gambar 8.21  
Pemetaan Relasi *Binary* dengan Kardinalitas Satu ke Satu

Pada Gambar 8.21, seorang Mahasiswa hanya dapat memilih 1 orang Dosen Pembimbing, sebaliknya seorang Dosen Pembimbing hanya boleh Membimbing 1 orang Mahasiswa (Kardinalitas Satu ke Satu). Mahasiswa Wajib untuk memilih Pembimbing, sedang Dosen Pembimbing tidak Wajib memiliki Mahasiswa Bimbingan. Perhatikan bahwa Modalitas *Pembimbing* pada sisi *Mahasiswa* adalah "Tidak Wajib", sedangkan Modalitas *Mahasiswa* pada sisi *Pembimbing* adalah "Wajib". Dengan demikian, atribut kunci primer yang terdapat pada entitas/relasi *Mahasiswa* akan menjadi atribut kunci tamu pada entitas/relasi *Pembimbing*.

Bagaimana jika modalitas pada kedua sisi adalah sama? Kita dapat memilih secara bebas atribut kunci primer dari salah satu entitas untuk menjadi atribut kunci tamu pada entitas lainnya.

### b. Relasi Satu ke Banyak

Jika terdapat relasi binary dengan kardinalitas *Satu ke Banyak*, maka harus dibuat dua relasi yang masing-masing mewakili setiap entitas. Atribut kunci primer pada sisi entitas kardinalitas Satu menjadi atribut kunci tamu pada sisi entitas kardinalitas Banyak (tanpa perlu memperhatikan jenis modalitas yang tercipta).

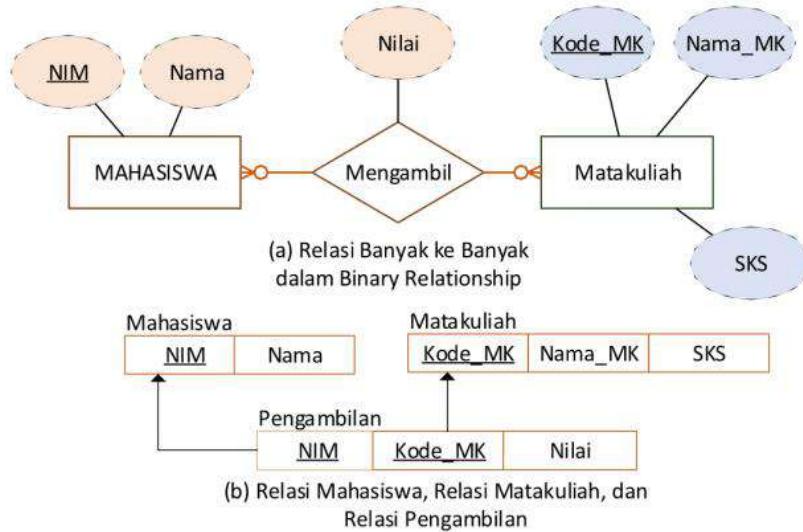


**Gambar 8.22**  
Pemetaan Relasi Binary dengan Kardinalitas Satu ke Banyak

Pada Gambar 8.22, seorang Mahasiswa dapat memiliki lebih dari 1 Laptop, sebaliknya sebuah Laptop hanya dapat dimiliki oleh 1 orang Mahasiswa (kardinalitas Satu ke Banyak). Dengan demikian, atribut kunci primer yang terdapat pada entitas/relasi *Mahasiswa* akan menjadi atribut kunci tamu pada entitas/relasi *Laptop*. Dalam kasus kardinalitas Satu ke Banyak, jenis modalitas yang terbentuk tidak memengaruhi posisi kunci primer/kunci tamu.

#### c. Relasi Banyak ke Banyak

Jika terdapat relasi binary dengan kardinalitas *Banyak ke Banyak*, maka harus dibuat tiga relasi. Dua relasi masing-masing mewakili setiap entitas, ditambah satu relasi transaksi yang terbentuk mewakili kedua entitas yang ada (relasi ke-3). Kedua entitas menggunakan kunci primer masing-masing, dan menjadi atribut kunci tamu pada relasi baru yang terbentuk (relasi ke-3). Dalam kasus ini, tidak perlu memperhatikan jenis modalitas yang tercipta.

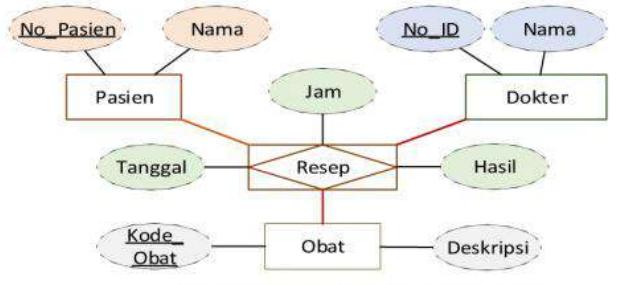


Gambar 8.23  
Pemetaan Relasi Binary dengan Kardinalitas Banyak ke Banyak

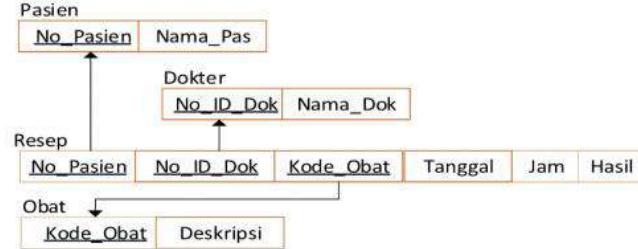
Pada Gambar 8.23, seorang Mahasiswa dapat memprogramkan (mengambil) lebih dari 1 Mata Kuliah, sebaliknya sebuah Mata Kuliah dapat diprogramkan oleh 1 orang Mahasiswa (kardinalitas Banyak ke Banyak). Dengan demikian, atribut kunci primer yang terdapat pada entitas/relasi *Mahasiswa* dan *Mata Kuliah* menjadi milik masing-masing entitas, dan menjadi kunci tamu pada relasi baru/relasi ke-3 (relasi Pengambilan) yang terbentuk. Relasi baru/relasi ke-3 (Pengambilan) memiliki atribut bawaannya sendiri (dalam kasus ini adalah atribut *Nilai*).

## 6. Pemetaan Relasi *Ternary*

Untuk melakukan pemetaan relasi yang menghubungkan 3 atau lebih entitas, disarankan untuk terlebih dahulu menciptakan suatu relasi *asosiatif*. Kunci pengidentifikasi untuk relasi *asosiatif* yang terbentuk adalah kunci-kunci primer yang berasal dari ketiga entitas yang terlibat dalam relasi (memiliki kemiripan dengan sistem pemetaan pada relasi binary Banyak ke Banyak). Pada beberapa kasus, atribut tambahan diperlukan untuk menyertai atribut kunci-kunci tamu yang terbentuk pada relasi asosiatif.



(a) Relasi Ternary Dalam Bentuk Asosiatif



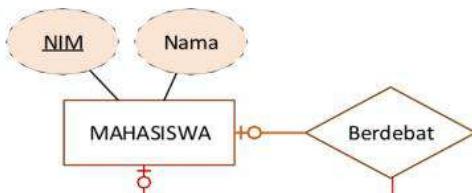
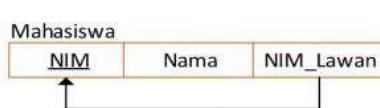
(b) Relasi Pasien, Dokter, Obat dan Resp

**Gambar 8.24**  
Pemetaan Relasi *Ternary*

Gambar 8.24 memperlihatkan relasi *ternary* yang menghubungkan entitas *Pasien*, *Dokter*, dan *Obat* dalam kasus *Pemeriksaan Kesehatan*. Dalam kasus ini, diciptakan sebuah relasi *asosiatif* "Resep" untuk menghubungkan ketiga entitas/relasi yang ada, sehingga pemetaan menghasilkan 4 buah relasi (*Pasien*, *Dokter*, *Obat*, dan *Resep*).

## 7. Pemetaan Relasi *Unary*

Relasi *Unary* sering disebut relasi *rekursif*. Perhatikan Gambar 8.25 berikut ini:

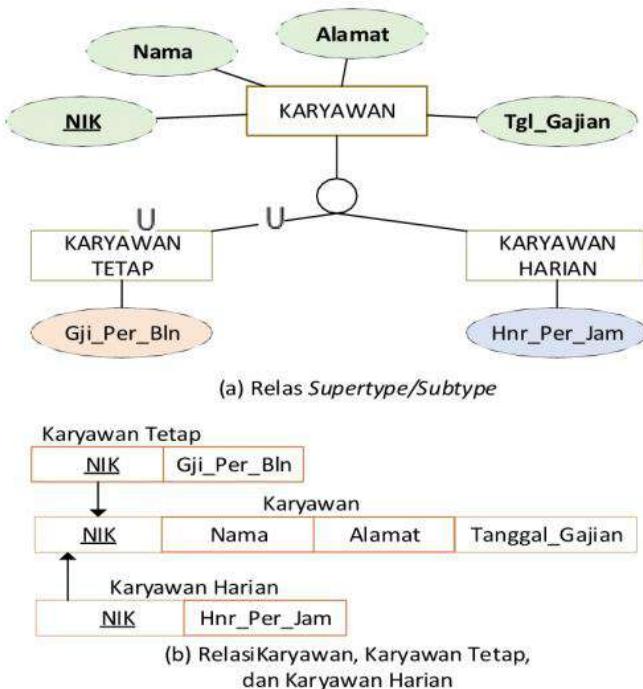
(a) Relasi *Unary*(b) Relasi *Mahasiswa*

**Gambar 8.25**  
Pemetaan Relasi *Unary*

Pada Gambar 8.25, seorang Mahasiswa dapat berdebat dengan seorang Mahasiswa lainnya (kardinalitas Satu ke Satu), namun keduanya tidak wajib mengikuti debat. Seorang Mahasiswa juga dapat berdebat dengan sekelompok Mahasiswa lainnya (kardinalitas Satu ke Banyak) atau sekelompok Mahasiswa dapat Berdebat dengan sekelompok Mahasiswa lainnya (kardinalitas Banyak ke Banyak). Hasil pemetaan relasi *Unary* hanya menghasilkan sebuah relasi (lihat Gambar 8.25(b)).

### 8. Pemetaan Relasi Supertype/Subtype

Pemetaan relasi *Supertype/Subtype* pada dasarnya adalah konsep pemetaan untuk mendukung model basis data berorientasi objek, yaitu konsep *Pewarisan*, seperti yang disajikan pada Gambar 8.26.



Gambar 8.26  
Pemetaan Relasi Supertype/Subtype

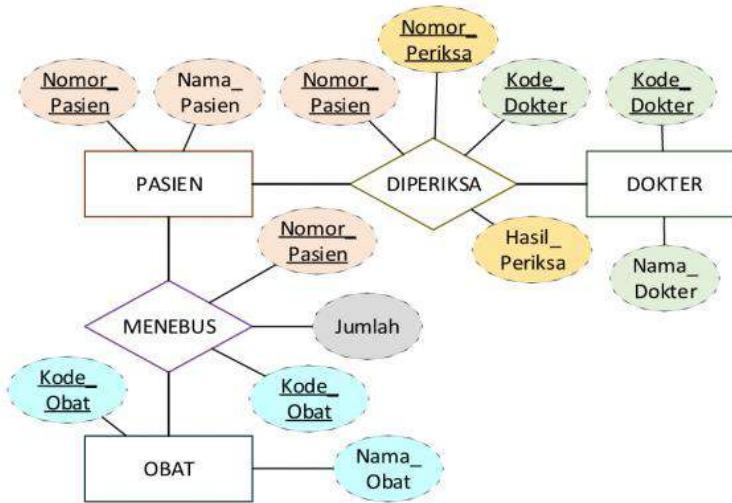
Pada Gambar 8.26 (b) terbentuk 3 buah relasi sesuai dengan jumlah entitas yang terdapat pada relasi supertype/subtype (satu relasi supertype dan dua relasi subtype). Atribut kunci primer yang terdapat pada relasi supertype (*Karyawan*) menjadi atribut kunci tamu pada kedua relasi subtype-nya (*Karyawan Tetap* dan *Karyawan Harian*).



## Latihan

Untuk memperdalam pemahaman Anda mengenai materi di atas, kerjakanlah latihan berikut!

- 1) Apa itu ERD? Jelaskan kegunaannya, serta keterkaitannya dengan sistem database!
- 2) Perhatikan gambar ERD berikut:

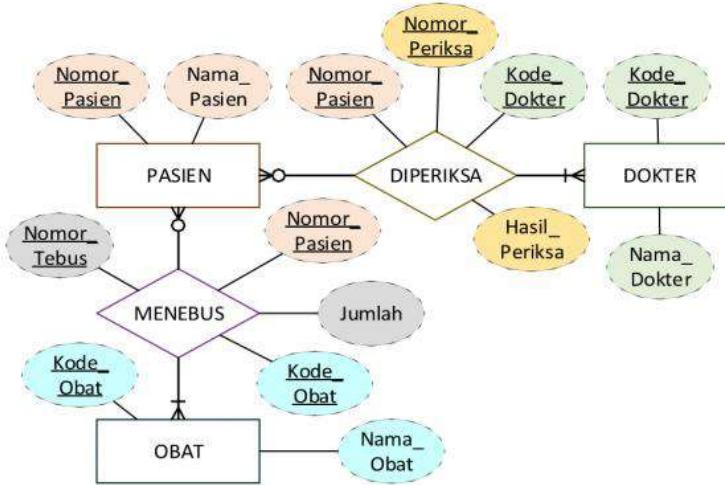


- a) Lengkapi gambar di atas dengan simbol kardinalitas dan modalitas!
- b) Deskripsikan kardinalitas dan modalitas terkait dengan gambar tersebut!
- c) Transformasikan ERD tersebut ke diagram relasi!

### Petunjuk Jawaban Latihan

- 1) ERD adalah diagram yang menyajikan informasi yang dibuat, disimpan, dan digunakan oleh suatu sistem bisnis. ERD digunakan untuk menemukan pola informasi dalam suatu sistem dan bagaimana informasi tersebut saling terkait satu sama lain. Pola informasi ini digunakan untuk mendesain sistem database sebagai media penyimpanan data transaksi yang terjadi dalam sistem bisnis di kemudian hari.

2) a)



b) Deskripsi:

- (1) DOKTER dapat memeriksa beberapa PASIEN, namun tidak wajib memeriksa PASIEN
- (2) PASIEN wajib diperiksa oleh Dokter (minimal 1 kali) dan PASIEN dapat diperiksa oleh beberapa DOKTER
- (3) PASIEN wajib menebus OBAT dan dapat menebus beberapa jenis OBAT
- (4) OBAT tertentu dapat ditebus oleh PASIEN manapun, namun OBAT tertentu tidak wajib ditebus oleh PASIEN

c) Diagram Relasi



Pada gambar tersebut di atas tercipta dua relasi yang keberadaannya bukan dari entitas murni, melainkan sebagai hasil kardinalitas/hubungan Banyak ke Banyak. Kedua relasi tersebut adalah relasi PEMERIKSAAN (hasil kardinalitas Banyak ke Banyak

antara entitas PASIEN dan entitas DOKTER) dan relasi PENEBUSAN (hasil kardinalitas Banyak ke Banyak antara entitas PASIEN dan entitas OBAT).



## Rangkuman

*Entity Relationship Diagram* (ERD) adalah teknik paling umum untuk menggambar model data, cara formal untuk merepresentasikan data yang digunakan dan dibuat oleh sistem bisnis.

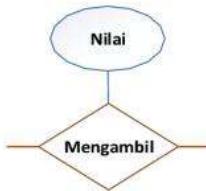
Ada tiga elemen dasar dalam bahasa pemodelan data, yang masing-masing diwakili oleh simbol grafik yang berbeda. *Entitas* adalah blok bangunan dasar untuk model data. Ini dapat berupa orang, tempat, atau benda yang datanya dikumpulkan. *Atribut* adalah jenis informasi yang ditangkap dari suatu entitas. Atribut yang secara unik dapat mengidentifikasi satu contoh dari suatu entitas disebut pengenal. *Relasi*, menunjukkan asosiasi antar entitas. Relasi antar entitas memiliki kardinalitas (banyaknya *instance* dari suatu entitas yang terkait dengan entitas lainnya) dan modalitas (keharusan untuk berpartisipasi dalam sebuah relasi).



## Tes Formatif 1

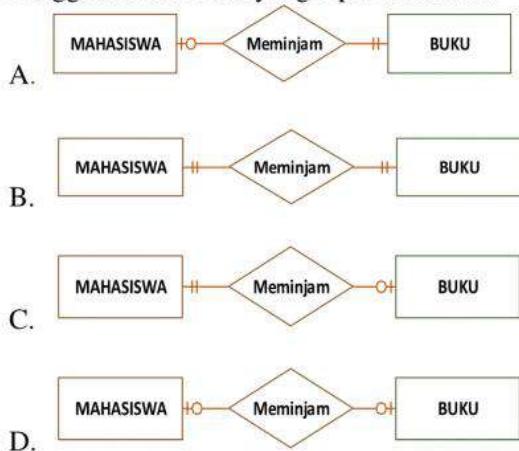
Pilihlah satu jawaban yang paling tepat!

- 1) Peralihan dari fase Analisis Sistem ke Fase Desain Sistem ditandai dengan aktivitas berupa ....
  - A. membangun *Data Flow Diagram*
  - B. membangun ERD
  - C. mengubah kebutuhan bisnis untuk sistem menjadi kebutuhan sistem berupa kebutuhan detail teknis untuk mengembangkan sistem
  - D. Pilihan jawaban A dan B benar
- 2) Tiga elemen dasar ERD adalah ....
  - A. Entitas, Aktor, Relasi
  - B. Entitas, Atribut, Relasi
  - C. *Entity Class*, Atribut, Relasi
  - D. *Control Class*, Atribut, Relasi
- 3) Jenis entitas yang tidak dapat mandiri, yang keberadaannya bergantung pada keberadaan entitas yang lainnya adalah ....
 
  - A. *Entity Class*
  - B. Entitas Kuat

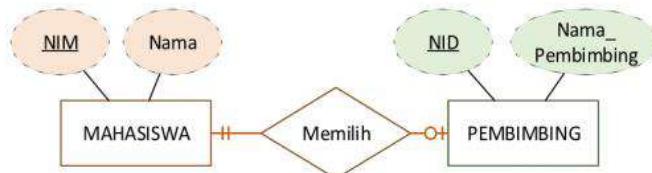
- C. Entitas Lemah  
D. Entitas Pengidentifikasi
- 4) Simbol atribut seperti pada gambar berikut ini adalah jenis atribut ....
- 
- A. Atribut Turunan  
B. Atribut Biasa  
C. Atribut Bayangan  
D. Atribut Bernilai Banyak
- 5) Simbol penghubung antara entitas kuat dan entitas lemah adalah ....
- A.  B. 
- C.  D. Semua jawaban salah
- 6) Atribut relasi seperti pada penggalan gambar di bawah ini muncul pada jenis relasi ....
- 
- A. Satu ke Satu ( 1 : 1 )  
B. Satu ke Banyak ( 1 : N )  
C. Banyak ke Satu ( M : 1 )  
D. Banyak ke Banyak ( M : N )
- 7) Konsep relasi *supertype/subtype* berkaitan dengan pemodelan berorientasi objek adalah ....
- A. *Polymorphism*  
B. *Superclass* dan *Subclass*  
C. Pewarisan (*inheritance*)  
D. Pilihan Jawaban B dan C benar
- 8) Suatu hubungan untuk menunjukkan apakah *instance* dari sebuah entitas "harus" atau "tidak harus" berpartisipasi pada entitas lainnya, disebut ....
- A. Modalitas  
B. Kardinalitas

- C. *Participation Constraint*  
 D. Pilihan jawaban A dan C benar
- 9) Perhatikan pernyataan relasi berikut: "Seorang mahasiswa wajib meminjam buku, dan hanya boleh meminjam 1 buku dalam suatu waktu peminjaman. Sebuah buku tertentu tidak wajib dipinjam oleh mahasiswa, namun jika buku tersebut dipinjam mahasiswa maka hanya boleh dipinjam oleh 1 orang mahasiswa dalam suatu waktu".

Penggambaran relasi yang tepat adalah ....



- 10) Perhatikan ERD berikut:



Jika ERD ini ditransformasikan ke Diagram Relasi, maka akan terbentuk ....

- A. Dua relasi (*Mahasiswa* dan *Pembimbing*). Atribut NIM akan menjadi kunci primer pada relasi *Mahasiswa* dan menjadi kunci tamu pada relasi *Pembimbing*
- B. Dua relasi (*Mahasiswa* dan *Pembimbing*). Atribut kunci primer dapat dipilih secara bebas dari salah satu di antara dua Entitas yang ada (NIM atau NID)
- C. Dua relasi (*Mahasiswa* dan *Pembimbing*). Atribut NID akan menjadi kunci primer pada relasi *Pembimbing* dan menjadi kunci tamu pada relasi *Mahasiswa*
- D. Tiga relasi (*Mahasiswa*, *Pemilihan*, dan *Pembimbing*). Atribut NIM dan atribut NID keduanya menjadi atribut kunci tamu pada relasi *Pemilihan*

Cocokkanlah jawaban Anda dengan Kunci Jawaban Tes Formatif 1 yang terdapat di bagian akhir modul ini. Hitunglah jawaban yang benar. Kemudian, gunakan rumus berikut untuk mengetahui tingkat penguasaan Anda terhadap materi Kegiatan Belajar 1.

$$\text{Tingkat Penguasaan} = \frac{\text{Jumlah Jawaban yang Benar}}{\text{Jumlah Soal}} \times 100$$

Arti tingkat penguasaan

<70%	70% - 79%	80% - 89%	90% - 100%
kurang	cukup	baik	baik sekali

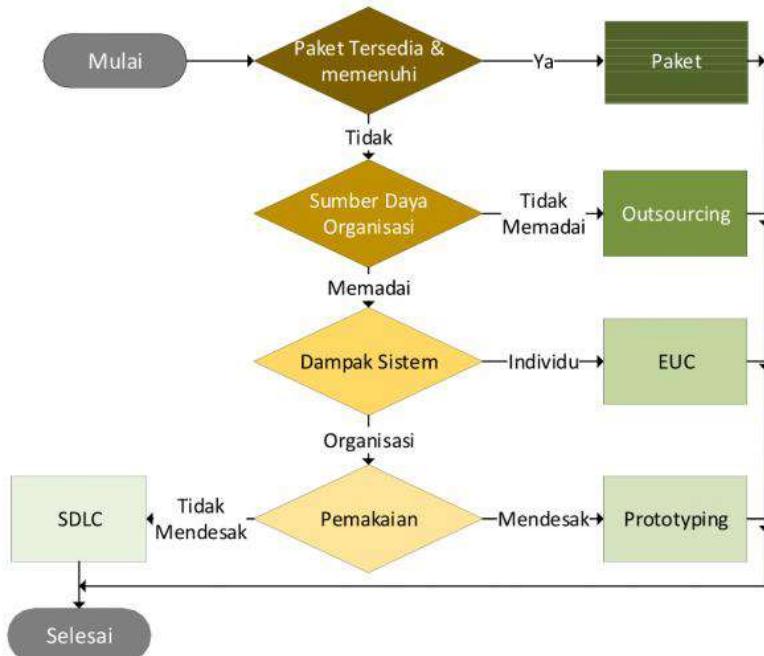
Apabila mencapai tingkat penguasaan 80% atau lebih, Anda dapat meneruskan dengan Kegiatan Belajar 2. **Bagus!** Jika masih di bawah 80%, Anda harus mengulangi materi Kegiatan Belajar 1, terutama bagian yang belum dikuasai.

## Strategi Akuisisi Sistem Informasi

Pada modul-modul sebelumnya telah dibahas fase analisis SDLC, yang diasumsikan sebagai cara untuk mendapatkan kebutuhan sistem baru secara sempurna. Fase analisis menekankan bahwa tim pengembang harus fokus pada menentukan kebutuhan logis sistem selama fase analisis, dan *menunda* memikirkan tentang bagaimana sistem harus diperoleh hingga fase desain.

Secara implisit, diasumsikan bahwa sistem akan dirancang, dikembangkan, dan dilaksanakan oleh tim proyek. Ini bukanlah asumsi yang sepenuhnya realistik. Dalam banyak proyek, tim mungkin mengidentifikasi bahwa beberapa bagian atau bahkan semua perangkat lunak sistem baru akan diperoleh dari beberapa penyedia luar. Beberapa organisasi telah menetapkan kebijakan akuisisi yang sangat mendukung perangkat lunak yang dibeli. Apakah ini berarti bahwa semua pekerjaan yang dijelaskan di modul-modul sebelumnya (fase analisis) dapat dilewati? Pekerjaan yang dilakukan dalam fase analisis masih penting untuk keberhasilan proyek, terutama alat dan teknik yang digunakan untuk menentukan, mendefinisikan, dan memperjelas kebutuhan bisnis dan kebutuhan pengguna. Penting untuk mengetahui apa yang kita butuhkan sebelum mencari produk yang dapat memberikan kesesuaian terbaik. Jika tidak, akan berisiko membiarkan vendor perangkat lunak mendefinisikan secara sepahap apa yang kita butuhkan, dan kita akan mendapatkan perangkat lunak yang tidak memenuhi kebutuhan bisnis dan kebutuhan pengguna yang sebenarnya.

Terdapat tiga pendekatan dalam strategi akuisisi sistem baru: (1) mengembangkan sendiri aplikasi (kustom); (2) membeli sistem yang telah ada (paket) dan mungkin menyesuaikannya dengan kebutuhan; dan (3) mengandalkan vendor eksternal, pengembang, atau penyedia layanan untuk membangun atau menyediakan sistem (*outsourcing*). Masing-masing pilihan ini memiliki kekuatan dan kelebihannya, dan masing-masing lebih sesuai dalam situasi yang berbeda. Gambar 8.27 menunjukkan strategi dalam akuisisi sistem informasi (Jogiyanto, 2005):



**Gambar 8.27**  
Pemilihan Strategi Akuisisi Sistem Informasi

### A. PENGEMBANGAN KUSTOM

Banyak tim proyek berasumsi bahwa pengembangan khusus (kustom) atau membangun sistem baru dari awal oleh pengembang di lingkungan internal organisasi (Jogiyanto, 2005, menyebutnya dengan istilah *End User Development*) adalah cara terbaik untuk membuat sistem. Tim pengembang memiliki kendali penuh atas tampilan dan fungsi sistem. Misalkan sebuah Perusahaan Penjualan Barang menginginkan fitur berbasis Web yang terhubung dengan strategi sistem penjualan khusus mereka yang selalu diperbarui, proyek tersebut mungkin melibatkan program yang sangat khusus dan kompleks. Alternatifnya, perusahaan mungkin memiliki lingkungan teknis di mana semua sistem informasi dibangun dari teknologi standar dan desain antarmuka khusus sehingga konsisten dan lebih mudah untuk diperbarui dan didukung. Dalam kasus tersebut, akan sangat efektif untuk membuat sistem baru dari awal yang memenuhi kebutuhan yang sangat khusus ini.

Dalam beberapa situasi, tantangan yang ditangani dengan sistem baru begitu signifikan dan menuntut rekayasa sistem yang serius diperlukan untuk menyelesaiannya. Dalam kasus ini, pengembang benar-benar tidak dapat menemukan solusi paket yang mampu memenuhi kebutuhan proyek, sehingga proyek pengembangan kustom adalah satu-satunya pilihan yang layak.

Pengembangan kustom memungkinkan pengembang menjadi fleksibel dan kreatif dalam memecahkan masalah bisnis. Perusahaan Penjualan Barang mungkin

membayangkan antarmuka Web yang menjadikan pembelian barang oleh pelanggan sebagai pendukung strategis yang penting. Perusahaan mungkin ingin menggunakan informasi dari sistem untuk lebih memahami pelanggannya yang memesan barang melalui Web, dan mungkin menginginkan fleksibilitas untuk mengembangkan sistem dengan memasukkan teknologi seperti perangkat lunak penambangan data dan sistem informasi geografis untuk melakukan pemasaran. Aplikasi kustom akan lebih mudah diubah dengan menyertakan komponen yang memanfaatkan teknologi terkini yang dapat mendukung upaya strategis tersebut.

Membangun sistem secara kustom juga membangun keterampilan teknis dan pengetahuan fungsional di dalam perusahaan. Saat pengembang bekerja dengan pengguna bisnis, pemahaman mereka tentang pertumbuhan bisnis organisasi dan mereka menjadi lebih mampu menyelaraskan sistem informasi dengan strategi dan kebutuhan. Pengembangan proyek sejenis di masa mendatang yang menerapkan teknologi serupa menjadi lebih mudah.

Namun demikian, pengembangan aplikasi secara kustom membutuhkan upaya khusus yang mencakup waktu yang lama dan kerja keras (Dennis, 2012). Banyak perusahaan memiliki staf pengembangan yang sudah terlalu banyak menangani pekerjaan. Menghadapi banyak sekali permintaan sistem, staf tidak punya waktu untuk proyek lain. Selain itu, berbagai keterampilan (teknis, interpersonal, fungsional, manajemen proyek, pemodelan) semuanya harus tersedia agar proyek dapat berjalan dengan lancar. Para profesional *information system*, terutama individu yang sangat terampil, cukup sulit untuk dipekerjakan dan dipertahankan.

Terdapat dua pilihan dalam mengembangkan sistem secara internal/kustom (Jogiyanto, 2005), yaitu: pengembangan sistem dilakukan oleh organisasi (*End User Development/EUD*) dan pengembangan sistem oleh pemakai (*End User Computing/EUC*). Pilihan ini didasarkan pada dampak dari sistem yang akan dikembangkan. Jika dampaknya sempit (hanya pada individu pemakai sistem yang sekaligus pengembang sistem itu saja), maka EUC dapat menjadi pilihan. Namun, jika dampaknya luas sampai pada lingkup organisasi, pengembangan sistem dengan metode EUC akan berisiko, karena jika terjadi kesalahan, dampaknya akan berpengaruh pada pemakai sistem lainnya atau pada sistem bisnis secara luas.

Pengembangan sistem secara internal dapat dilakukan dengan menggunakan metode SDLC (telah dibahas secara detail pada Modul 2 Kegiatan Belajar 2) atau menggunakan metode *prototyping*. Metode SDLC menghasilkan model sistem yang sempurna namun dengan waktu pengembangan yang cukup panjang/lama, sedangkan metode *prototyping* menjanjikan waktu pengembangan yang singkat namun dengan hasil awal yang belum sempurna. Metode *prototyping* banyak digunakan untuk mengembangkan sistem informasi yang harus segera dioperasikan, jika terlambat permasalahan yang akan diselesaikan sudah menjadi tidak relevan lagi, sehingga proses pengambilan keputusan menjadi terlambat.

*Prototyping* adalah proses pengembangan suatu prototipe secara cepat untuk digunakan terlebih dahulu dan ditingkatkan terus menerus sampai didapatkan sistem yang utuh. Proses membangun sistem ini dilakukan dengan membuat prototipe atau model awal, mencobanya dan meningkatkannya secara berulang sampai didapatkan sistem yang lengkap (disebut proses iteratif) dari pengembangan sistem (Jogiyanto, 2005).

Tahapan-tahapan yang dilakukan dalam pengembangan sistem menggunakan metode *prototyping* adalah sebagai berikut.

1. Mengidentifikasi kebutuhan pemakai yang paling mendasar.  
Pengembang sistem dapat mewawancara pemakai sistem tentang kebutuhan yang paling minimal terlebih dahulu. Proses ini sama dengan proses analisis pada pengembangan sistem model SDLC.
2. Membangun prototipe.  
Prototipe dibangun oleh pengembang sistem dengan cepat. Hal ini dimungkinkan karena pengembang sistem hanya membangun bagian yang paling mendasar dari keseluruhan sistem yang paling dibutuhkan oleh pemakai sistem. Penggunaan alat-alat bantu seperti DBMS dan CASE juga turut mempercepat pembangunan sistem prototipe.
3. Menggunakan prototipe.  
Pemakai sistem dianjurkan untuk menggunakan prototipe untuk menilai kekurangan-kekurangan dari sistem yang sedang dibangun, selanjutnya memberikan masukan untuk penyempurnaan sistem.
4. Merevisi dan meningkatkan prototipe.  
Pembuat sistem menyempurnakan prototipe berdasarkan masukan pemakai sistem. Kegiatan nomor 3 dan nomor 4 merupakan siklus/perulangan hingga diperoleh sistem yang dinyatakan memenuhi kebutuhan pemakai.

## B. PAKET

Banyak organisasi membeli perangkat lunak yang telah dibuat (dikemas), daripada mengembangkan sendiri. Perangkat lunak yang telah tersedia di pasar umum (kemasan) disebut dengan istilah *paket (package)*. Ada ribuan program perangkat lunak yang tersedia secara komersial yang telah ditulis untuk melayani banyak tujuan. Misalnya perangkat lunak pengolah kata, pernahkah kita mempertimbangkan untuk membuat perangkat lunak pengolah kata sendiri? Itu seperti sangat tidak masuk akal, mengingat banyaknya paket perangkat lunak pengolah kata bagus yang tersedia dengan biaya yang relatif murah.

Demikian pula, sebagian besar perusahaan memiliki kebutuhan, seperti penggajian atau piutang, yang dapat dipenuhi dengan baik oleh perangkat lunak kemasan. Jauh lebih efisien untuk membeli program yang telah dibuat, diuji, dan dibuktikan. Perangkat lunak kemasan dapat dibeli dan dipasang dengan cepat

dibandingkan dengan sistem yang dibuat secara khusus. Sistem paket menggabungkan keahlian dan pengalaman vendor yang membuat perangkat lunak.

Perangkat lunak kemasan dapat berupa program aplikasi sederhana untuk fungsi tunggal (misalnya aplikasi untuk mengunduh), hingga sistem kompleks multi fungsi seperti Aplikasi Perencanaan Sumber Daya Perusahaan (*Enterprise Resource Planning/ERP*).

Satu masalah adalah bahwa perusahaan yang menggunakan sistem terpaket harus menerima fungsionalitas yang disediakan oleh sistem, sedangkan aplikasi terpaket jarang ada yang sempurna. Jika sistem paket memiliki area cakupan yang besar, implementasinya dapat berarti perubahan substansial dalam cara perusahaan menjalankan bisnis. Membiarakan teknologi menggerakkan bisnis organisasi bisa menjadi cara yang berbahaya (Dennis, 2012).

Sebagian besar aplikasi yang dikemas memungkinkan beberapa penyesuaian atau manipulasi parameter sistem untuk mengubah cara kerja fitur tertentu. Paket perangkat lunak akuntansi dapat menawarkan pilihan berbagai cara untuk menangani arus kas atau pengendalian persediaan sehingga dapat mendukung praktik akuntansi di berbagai organisasi. Jika jumlah penyesuaian cukup banyak dan paket perangkat lunak memiliki beberapa fitur yang tidak berfungsi sebagaimana yang dibutuhkan perusahaan, tim proyek dapat membuat solusi. Salah satu solusi adalah program *add-on* yang dibuat khusus untuk diintegrasikan dengan aplikasi terpaket untuk menangani kebutuhan khusus. Ini bisa menjadi cara yang bagus untuk membuat fungsionalitas yang dibutuhkan yang tidak ada dalam paket perangkat lunak. Namun, solusi ini harus menjadi pilihan terakhir, karena beberapa alasan. Pertama, penyelesaian masalah tidak didukung oleh vendor yang menyediakan perangkat lunak terpaket, jadi ketika pemutakhiran dilakukan ke sistem utama, solusi itu akan tidak efektif. Selain itu, jika muncul masalah, vendor cenderung menyalahkan solusi tersebut sebagai penyebab dan menolak memberikan dukungan (Dennis, 2012).

Agar tujuan organisasi bisnis memilih paket dapat tercapai dengan baik, ada beberapa faktor yang perlu diperhatikan dalam memilih paket (Jogiyanto, 2005), yaitu:

1. Spesifikasi yang dibutuhkan oleh bisnis. Spesifikasi merupakan kemampuan paket untuk memenuhi kebutuhan bisnis.
2. Ketersediaan paket yang dibutuhkan oleh bisnis.
3. Mengevaluasi kemampuan paket. Beberapa kriteria perlu diperhatikan dalam mengevaluasi kemampuan paket, yaitu:
  - a. Fungsi
    - 1) Fungsi-fungsi yang ditawarkan oleh paket sesuai dengan kebutuhan bisnis
    - 2) Fungsi-fungsi yang dapat digunakan dan fungsi-fungsi yang tidak dapat didukung oleh paket
    - 3) Berapa luas modifikasi harus dilakukan

- 4) Seberapa baik paket mendukung kebutuhan bisnis saat ini dan di masa mendatang
- b. Fleksibilitas
  - 1) Tingkat kesulitan ketika paket dimodifikasi
  - 2) Fitur-fitur yang dapat dimodifikasi sesuai kebutuhan
  - 3) Kesediaan vendor memodifikasi paket
- c. Kemudahan
  - 1) Seberapa mudah paket digunakan
  - 2) Pelatihan yang harus dilakukan untuk dapat menggunakannya
- d. Perangkat keras dan perangkat lunak dukungan
  - 1) Jenis komputer apa yang mendukung paket
  - 2) Spesifikasi teknis komputer yang cocok
- e. Karakteristik file dan basis data
  - 1) Struktur basis data yang digunakan oleh paket
  - 2) Sejauh mana pemakai dapat mengakses data langsung dari basis data aplikasi paket
  - 3) Sejauh mana basis data aplikasi paket dapat diintegrasikan dengan basis data sistem yang ada
  - 4) Sejauh mana struktur basis data aplikasi paket dapat dimodifikasi
- f. Instalasi
  - 1) Kemudahan instalasi paket
  - 2) Kemudahan mengkonversi dari sistem lama ke sistem yang baru
- g. Perawaran
  - 1) Ketersediaan pemutakhiran aplikasi paket oleh vendor
  - 2) Seberapa mudah pemutakhiran dapat dilakukan
  - 3) Pihak mana yang melakukan perawatan sistem paket
  - 4) Sumber daya yang dibutuhkan untuk melakukan perawatan sistem
- h. Dokumentasi
  - 1) Ketersediaan dan kelengkapan dokumentasi
  - 2) Seberapa mudah memahami dan menggunakan dokumen-dokumen yang tersedia

- i. Kualitas Vendor
  - 1) Pengalaman dan reputasi vendor
  - 2) Dukungan yang disediakan
  - 3) Respon vendor terhadap keluhan pelanggan
  
- j. Biaya
  - 1) Harga paket
  - 2) Biaya perawatan
  - 3) Harga kompetitif dengan vendor lain

Sistem perangkat lunak *paket* mempunyai beberapa kelebihan, di antaranya (Jogiyanto, 2005):

- a. Kualitas dari paket dapat diandalkan; pada umumnya paket dikembangkan oleh tim pengembang yang berkualitas, dikembangkan dengan biaya yang cukup mahal, sudah diuji kualitasnya sebelum paket dipasarkan, serta telah mengalami penyempurnaan setelah menerima berbagai keluhan dari pembeli sebelumnya.
- b. Dapat digunakan seketika; sehingga cocok untuk kebutuhan sistem yang mendesak.
- c. Harga paket relatif murah; walaupun biaya pengembangan paket mahal, akan tetapi paket dipasarkan kepada banyak pengguna, sehingga harga jual per paket menjadi relatif murah dibanding jika harus mengembangkan sendiri.
- d. Dapat digunakan untuk rekayasa ulang proses bisnis; memilih paket-paket *best practice* yaitu paket dengan proses bisnis yang terbaik yang pernah diterapkan di suatu organisasi. Rekayasa ulang proses bisnis (*business process reengineering/BPR*) dapat dilakukan dengan menerapkan paket *best practice* dan memodifikasi organisasi untuk mengikuti proses yang terdapat dalam paket.

- Selain memiliki beberapa kelebihan, paket juga memiliki beberapa keterbatasan.
- a. Sering tidak relevan dengan fungsi-fungsi bisnis yang unik; paket harus dimodifikasi untuk disesuaikan dengan kebutuhan bisnis.
  - b. Jika terdapat permasalahan atau ketidaksesuaian dengan kebutuhan bisnis, sulit mengerjakan sendiri perbaikan, modifikasi, dan pengembangan paket; mengubah kode program lebih sulit dilakukan dibanding dengan jika membuatnya dari awal.
  - c. Basis data tidak terintegrasi dengan aplikasi *existing*; paket umumnya menggunakan program tertentu dengan struktur basis data yang unik dan berbeda dengan struktur basis data lainnya.
  - d. Tidak memberikan keuntungan kompetisi; paket yang sama umumnya digunakan oleh banyak organisasi pesaing, sehingga keuntungan kompetisi ketika menggunakan paket menjadi tidak maksimal.

### C. OUTSOURCING

Jika paket tidak tersedia di pasaran atau tidak memberikan keuntungan bagi bisnis, prioritas biasanya jatuh pada *outsourcing*. Pilihan akuisisi yang membutuhkan sumber daya internal paling sedikit adalah *outsourcing*, yang berarti mempekerjakan vendor eksternal, pengembang, atau penyedia layanan untuk membuat atau memasok sistem. Istilah *outsourcing* telah mencakup berbagai cara untuk mendapatkan layanan dan produk TI. Perusahaan *outsourcing* yang disebut penyedia layanan aplikasi (*Application Service Provider/ASP*) menyediakan aplikasi perangkat lunak dan/atau layanan terkait perangkat lunak melalui jaringan area luas atau Internet. Dalam pendekatan untuk memperoleh perangkat lunak ini, ASP menghosting dan mengelola aplikasi perangkat lunak, serta memiliki, mengoperasikan, dan memelihara server yang menjalankan aplikasi. ASP juga mempekerjakan orang yang dibutuhkan untuk memelihara aplikasi.

Organisasi bisnis yang ingin menggunakan *outsourcing* menyepakati kontrak aplikasi perangkat lunak dengan ASP yang membuatnya tersedia untuk pelanggan melalui jaringan area luas atau Internet, baik yang diinstal pada komputer klien atau melalui browser. Pelanggan ditagih oleh ASP untuk aplikasi tersebut berdasarkan penggunaan atau berdasarkan biaya bulanan atau tahunan.

*Software as a Service* (SaaS) adalah istilah populer yang pada dasarnya merupakan pengembangan dari model ASP. Istilah ini biasanya digunakan untuk menjelaskan situasi di mana vendor SaaS mengembangkan dan mengelola perangkat lunak mereka sendiri daripada mengelola dan menghosting perangkat lunak independen pihak ketiga (model ASP yang lebih tradisional).

Ada berbagai jenis penyedia layanan aplikasi. Beberapa memberikan aplikasi bisnis kelas atas yang dapat melayani seluruh perusahaan. Beberapa lebih fokus pada melayani klien bisnis kecil hingga menengah. Beberapa ASP mengkhususkan diri pada kebutuhan bisnis tertentu (seperti misalnya *Customer Relationship Management/CRM*), sementara beberapa mengkhususkan diri pada industri tertentu (misalnya, perawatan kesehatan).

Mendapatkan akses ke paket perangkat lunak melalui penyedia layanan aplikasi (*outsourcing*) memiliki banyak keuntungan: (1) biaya kerjasama yang rendah dan, dalam banyak kasus, waktu penyiapan yang sangat singkat. Model *pay-as-you-go* seringkali jauh lebih murah, kecuali pengguna layanan secara intensif (sering/terus menerus); (2) investasi pada staf Teknologi Informasi dapat dikurangi, dan investasi dalam infrastruktur Teknologi Informasi khusus seringkali dapat dihindari; (3) pada umumnya jasa yang diberikan oleh *outsources* lebih berkualitas dibanding dikerjakan secara internal oleh organisasi bisnis, karena *outsourcer* memang profesional dan ahli pada bidangnya; (4) sistem *outsourcing* mengurangi risiko kegagalan investasi yang mahal; (5) organisasi bisnis dapat memfokuskan sumber daya untuk pekerjaan lain yang lebih produktif.

Perusahaan *outsourcing* yang akan mengembangkan sistem kustom atas nama pelanggan juga tersedia. Ada keuntungan besar jika orang lain mengembangkan sistem. Vendor mungkin lebih berpengalaman dalam teknologi atau memiliki lebih banyak sumber daya, seperti programmer berpengalaman. Banyak perusahaan memulai kesepakatan *outsourcing* untuk mengurangi biaya, sedangkan yang lain melihatnya sebagai peluang untuk menambah nilai bisnis. Misalnya, alih-alih membuat program yang menangani proses pembelian atau membeli paket yang sudah ada sebelumnya, suatu organisasi bisnis penjualan mungkin memutuskan untuk membiarkan penyedia layanan Web menyediakan layanan komersial untuk mereka.

Untuk alasan apapun, *outsourcing* bisa menjadi alternatif yang baik untuk sistem baru; namun, itu bukan tanpa risiko. Jika organisasi bisnis memutuskan untuk menyerahkan pembuatan sistem baru di tangan orang lain, organisasi tersebut dapat memikirkan beberapa hal: (1) bocornya informasi rahasia; (2) kehilangan kendali atas pengembangan di masa depan; (3) para profesional di lingkungan internal bisnis tidak mendapatkan keuntungan dari keterampilan yang bisa dipelajari dari proyek, alih-alih, keahlian dialihkan ke organisasi luar.

Sebagian besar risiko dapat diatasi ketika organisasi bisnis memutuskan untuk melakukan *outsourcing*, tetapi ada dua yang sangat penting: (1) nilai kebutuhan untuk proyek secara menyeluruh (organisasi bisnis tidak boleh melakukan *outsourcing* terhadap apa yang tidak dipahami). Jika organisasi bisnis telah melakukan perencanaan dan analisis kebutuhan secara ketat, maka organisasi itu telah memahami kebutuhannya; (2) hati-hati memilih vendor atau pengembang dengan rekam jejak layanan yang terbukti tidak baik pada jenis sistem dan teknologi yang dibutuhkan oleh sistem bisnis organisasi.

Membuat kontrak dengan agen *outsourcing* secara adil adalah seni karena organisasi bisnis perlu menyeimbangkan fleksibilitas dengan cermat dengan kebutuhan yang didefinisikan dengan jelas. Kebutuhan sering kali berubah seiring waktu, jadi organisasi bisnis tidak ingin kontrak terlalu spesifik dan kaku sehingga perubahan tidak dapat dilakukan. Pikirkan tentang seberapa cepat teknologi seperti World Wide Web (www) berubah. Sulit untuk meramalkan bagaimana sebuah proyek dapat berkembang dalam jangka waktu yang lama. Kontrak jangka pendek memberikan ruang untuk penilaian ulang jika perlu diubah atau jika hubungan tidak berjalan seperti yang diharapkan kedua belah pihak. Dalam semua kasus, hubungan dengan agen *outsourcing* harus dipandang sebagai kemitraan di mana kedua belah pihak mendapatkan keuntungan dan berkomunikasi secara terbuka.

Mengelola hubungan dengan *outsourcing* adalah pekerjaan penuh waktu. Jadi, seseorang perlu ditugaskan penuh waktu untuk mengelola agen *outsourcing*, dan level orang tersebut harus sesuai dengan ukuran pekerjaannya (keterlibatan *outsourcing* bernilai jutaan dolar harus ditangani oleh eksekutif tingkat tinggi). Sepanjang hubungan, kemajuan harus dilacak dan diukur berdasarkan tujuan yang telah ditentukan sebelumnya.



### Latihan

Untuk memperdalam pemahaman Anda mengenai materi di atas, kerjakanlah latihan berikut!

- 1) Sebutkan dan jelaskan tiga model akuisisi sistem informasi bagi organisasi!
- 2) Jelaskan kelebihan dan kekurangan model akuisisi secara kustom!
- 3) Jelaskan dua model akuisisi sistem secara kustom!
- 4) Jelaskan kelebihan akuisisi sistem informasi menggunakan cara paket !
- 5) Jelaskan dua model akuisisi sistem secara *outsourcing*!

#### Petunjuk Jawaban Latihan

- 1) Tiga model akuisisi sistem informasi bagi organisasi:
  - a) Kustom adalah mengembangkan sistem baru secara internal dari awal untuk memenuhi kebutuhan khusus bagi manajemen organisasi.
  - b) Paket adalah membeli perangkat lunak yang telah dikemas (tersedia di pasar umum).
  - c) *Outsourcing* adalah mempekerjakan vendor eksternal, pengembang, atau penyedia layanan untuk membuat atau memasok perangkat lunak dan atau layanan terkait perangkat lunak bagi manajemen organisasi.

- 2) Kelebihan dan kekurangan model akuisisi sistem secara kustom:

##### Kelebihan:

- a) pengembang memiliki kendali penuh atas sistem yang dikembangkan saat ini dan di masa mendatang;
- b) pengembang menjadi fleksibel dan kreatif dalam memecahkan masalah bisnis;
- c) meningkatkan pemahaman karyawan terhadap pertumbuhan bisnis organisasi.

##### Kekurangan:

- a) pekerjaan utama staf dalam organisasi berpotensi terbengkalai karena staf terbebani pekerjaan lain;
- b) organisasi harus menyediakan Sumber Daya Manusia yang mempunyai keterampilan di berbagai bidang untuk mendukung proyek berjalan dengan lancar, sementara hal tersebut sulit untuk dipenuhi.

- 3) Dua model akuisisi sistem secara kustom:
  - a) *End User Development*: pengembangan sistem secara internal dalam cakupan organisasi (cakupan luas)
  - b) *End User Computing*: pengembangan sistem secara internal dalam lingkup pemakai sistem (cakupan sempit)
- 4) Kelebihan akuisisi sistem informasi menggunakan cara paket:
  - a) kualitas sistem dapat diandalkan, sebab sistem paket telah teruji kualitasnya setelah mengalami penyempurnaan dari berbagai keluhan pembeli sebelumnya;
  - b) dapat digunakan dengan segera, terutama untuk memenuhi kebutuhan sistem yang mendesak;
  - c) relatif lebih murah dibanding jika harus mengembangkan sendiri;
  - d) model paket *best practice* dapat digunakan untuk memodifikasi organisasi mengikuti proses yang terdapat dalam paket.
- 5) Dua model akuisisi sistem secara *outsourcing*:
  - a) Model ASP (*Application Service Provider*); vendor atau provider mengembangkan aplikasi untuk digunakan oleh organisasi bisnis. Namun dalam hal ini, provider mengelola, mengoperasikan, dan memelihara server aplikasi; selanjutnya provider menagih biaya penggunaan berdasarkan pemakaian.
  - b) Model SaaS (*Software as a Service*); provider mengembangkan dan mengelola perangkat lunak mereka sendiri, dan membuatnya tersedia untuk digunakan oleh organisasi bisnis; selanjutnya provider menagih biaya penggunaan berdasarkan pemakaian (misalnya, aplikasi Zoom yang dapat disewa untuk keperluan *virtual convergence*).



## Rangkuman

Selama fase desain, tim proyek perlu mempertimbangkan tiga pendekatan untuk mengakuisisi sistem baru, yaitu mengembangkan sendiri aplikasi secara khusus (kustom); membeli sistem yang dikemas (paket) dan menyesuaikannya dengan kebutuhan; dan mengandalkan vendor eksternal, pengembang, atau penyedia sistem untuk membangun dan/atau mendukung sistem (*outsourcing*).

Pengembangan secara kustom memungkinkan pengembang menjadi fleksibel dan kreatif dalam cara mereka memecahkan masalah bisnis, dan membangun pengetahuan teknis dan fungsional dalam organisasi. Namun, banyak organisasi memiliki staf pengembangan yang berkomitmen untuk tetap memenuhi banyak permintaan sistem yang sedang berjalan saat ini, sehingga mereka tidak punya waktu untuk mencerahkan waktu bagi proyek baru yang akan dikembangkan. Jauh lebih efisien untuk membeli program paket, diuji, dan dibuktikan; dan sistem paket dapat

dibeli dan dipasang dalam waktu yang relatif singkat, jika dibandingkan dengan pengembangan secara kustom.

Strategi akuisisi ketiga adalah melakukan *outsourcing* proyek dan membayar vendor eksternal, pengembang, atau penyedia layanan untuk membuat sistem. Ini bisa menjadi alternatif yang baik untuk pengembangan sistem baru. Namun, itu tidak dapat terwujud tanpa biaya. Jika sebuah organisasi memutuskan untuk menyerahkan pembuatan sistem baru di tangan orang lain, organisasi harus berhati-hati dalam hal informasi rahasia dan kehilangan kendali atas pengembangan di masa depan.



### Tes Formatif 2

---

Pilihlah satu jawaban yang paling tepat!

- 1) Berikut adalah tiga pendekatan dalam akuisisi sistem informasi baru, *kecuali* ....
  - A. mengembangkan sendiri sistem (kustom)
  - B. menyewa sistem dari perusahaan lain yang menggunakan sistem sejenis dengan kebutuhan organisasi
  - C. membeli sistem yang sudah ada di pasar bebas (paket)
  - D. mengandalkan vendor eksternal (*outsourcing*)
- 2) Jika manajemen organisasi membutuhkan suatu sistem informasi yang spesifik dan memiliki fleksibilitas yang tinggi, strategi akuisisi yang tepat adalah ....
  - A. kustom
  - B. paket
  - C. *outsourcing*
  - D. Semua pilihan jawaban benar
- 3) Cara yang dapat digunakan dalam pengembangan sistem secara internal (kustom):
  - A. SDLC
  - B. mengandalkan vendor eksternal
  - C. *prototyping*
  - D. Jawaban A dan C benar
- 4) Proses pengembangan suatu sistem secara cepat agar dapat digunakan segera, kemudian ditingkatkan terus menerus sampai didapatkan sistem yang utuh, disebut metode pengembangan ....
  - A. SDLC
  - B. *Extreme Programming*
  - C. *Prototyping*
  - D. Semua jawaban salah

- 5) Jika manajemen organisasi hanya memiliki sumber daya untuk mengoperasikan sistem dan tidak memiliki sumber daya yang cukup untuk mengembangkan sendiri sistem informasi, maka strategi akuisisi sistem yang tepat adalah ....
- paket
  - kustom
  - outsourcing*
  - kerjasama dengan perusahaan yang menggunakan sistem sejenis dengan kebutuhan organisasi
- 6) Berikut adalah teknik memilih paket yang efektif, *kecuali* ....
- memilih paket yang memiliki fungsi-fungsi paling relevan dengan kebutuhan bisnis
  - kesediaan vendor memodifikasi paket
  - mempertimbangkan kemudahan instalasi dan penggunaan paket
  - harga paket yang mahal tidak menjadi halangan
- 7) Berikut adalah kelemahan akuisisi sistem informasi menggunakan cara paket, *kecuali* ....
- fungsi-fungsi bisnis dalam paket sering tidak relevan
  - sulit melakukan perbaikan jika terjadi masalah atau perubahan kebutuhan fungsional bisnis
  - sulit mengintegrasikan dengan sistem yang telah ada
  - kualitas paket tidak dapat diandalkan
- 8) Pilihan model akuisisi sistem yang membutuhkan sumber daya internal paling sedikit adalah ....
- paket
  - kustom
  - outsourcing*
  - Rapid Application Development (RAD)*
- 9) Berikut adalah keuntungan mendapatkan akses ke produk perangkat lunak melalui sistem *outsourcing*, *kecuali* ....
- sistem segera dapat digunakan
  - meningkatkan sumber daya manusia di bidang teknologi informasi bagi organisasi
  - mengurangi investasi dalam infrastruktur teknologi informasi yang cukup besar
  - kualitas sistem yang ditawarkan umumnya lebih baik dari sistem yang dikembangkan secara internal

#### 8.46 Transisi dari Analisis Sistem Ke Desain/Perancangan Sistem

- 10) Beberapa risiko yang perlu dipahami ketika memutuskan memilih model akuisisi dengan sistem *outsourcing* adalah ....
- A. organisasi tidak dapat berkembang dalam berbagai aspek
  - B. bocornya informasi rahasia
  - C. kehilangan kendali atas pengembangan di masa depan
  - D. Pilihan jawaban B dan C benar

Cocokkanlah jawaban Anda dengan Kunci Jawaban Tes Formatif 2 yang terdapat di bagian akhir modul ini. Hitunglah jawaban yang benar. Kemudian, gunakan rumus berikut untuk mengetahui tingkat penguasaan Anda terhadap materi Kegiatan Belajar 2.

$$\text{Tingkat Penguasaan} = \frac{\text{Jumlah Jawaban yang Benar}}{\text{Jumlah Soal}} \times 100$$

Arti tingkat penguasaan

<70%	70% - 79%	80% - 89%	90% - 100%
kurang	cukup	baik	baik sekali

Apabila mencapai tingkat penguasaan 80% atau lebih, Anda dapat meneruskan dengan modul selanjutnya. **Bagus!** Jika masih di bawah 80%, Anda harus mengulangi materi Kegiatan Belajar 2, terutama bagian yang belum dikuasai.

## Kunci Jawaban Tes Formatif

### *Tes Formatif 1*

- 1) C
- 2) B
- 3) C
- 4) A
- 5) C
- 6) D
- 7) D
- 8) D
- 9) A
- 10) C

### *Tes Formatif 2*

- 1) B
- 2) A
- 3) D
- 4) C
- 5) A
- 6) D
- 7) D
- 8) C
- 9) B
- 10) D

## Glosarium

Akuisisi	: Cara mendapatkan perangkat lunak untuk mendukung operasi bisnis organisasi
Application Service Provider	: Penyedia layanan aplikasi, vendor outsourcing yang menyediakan layanan aplikasi
Binary Relationship	: Relasi berderajat dua
Business Process Reengineering	: Rekayasa ulang proses bisnis
Custom	: Pengembangan aplikasi yang dilakukan sendiri dalam lingkungan manajemen organisasi
Customer Relationship Management	: Aplikasi yang khusus menyediakan layanan keluhan pelanggan
End User Computing	: Pengembangan sistem secara internal dalam lingkup pemakai sistem
End User Development	: Pengembangan sistem secara internal dalam cakupan organisasi
Enterprise Resource Planning	: Sistem informasi yang diperuntukkan bagi perusahaan manufaktur maupun jasa yang berperan mengintegrasikan dan mengotomatiskan proses bisnis yang berhubungan dengan aspek operasi, produksi, maupun distribusi di perusahaan bersangkutan
Identifier	: Relasi pengidentifikasi, menghubungkan entitas lemah dan entitas kuat
Identifying owner	: Entitas tempat entitas lemah bergantung
Multi-value attribute	: Atribut yang nilainya dapat lebih dari satu
Outsourcing	: mempekerjakan vendor eksternal untuk membuat atau memasok sistem informasi
Package	: Aplikasi yang tersedia/dapat dibeli di pasar umum
Partial identifier	: Relasi pengidentifikasi yang berfungsi secara sebagian

Pay-as-you-go	: Jenis pembayaran yang dilakukan pada penyedia jasa perangkat lunak sesuai dengan waktu penggunaannya
Program add-on	: Perangkat lunak yang berfungsi untuk menambahkan fitur tambahan untuk sebuah program
Prototyping	: Metode pengembangan sistem informasi secara cepat dan iteratif tanpa mempertimbangkan kesempurnaan hasil pengembangan pertama
Rekursif	: Fungsi dalam pemrograman yang memanggil dirinya sendiri
Single attribute	: Atribut yang nilainya tunggal
Software as a Service	: Memanfaatkan aplikasi yang dikelola oleh vendor pihak ketiga (pada umumnya bersifat online) tanpa harus mengurus aplikasi tersebut
Strong Entity	: Entitas yang keberadaannya tidak bergantung pada entitas lain (mandiri)
Subtype	: Suatu entitas yang merupakan pengkhususan dari supertype
Supertype	: Suatu entitas yang atribut-atributnya bersifat umum
Ternary Relationship	: Relasi berderajat tiga
Unary Relationship	: Relasi berderajat satu
Weak Entity	: Entitas yang keberadaannya bergantung pada entitas lain (tidak dapat mandiri)
Vendor	: Penyedia layanan jasa pengembangan dan atau pemeliharaan sistem informasi

## Daftar Pustaka

Dennis, A., Wixom, B. H., Roth, R.M. (2012). *Systems analysis & design*. 5th Edition. United States of America: John Wiley & Sons, Inc.

Jogiyanto, H.M. (2005). *Sistem teknologi informasi*. Edisi 2. Yogyakarta: ANDI.

Nugroho, A. (2004). *Konsep pengembangan sistem basis data*. Bandung: INFORMATIKA.

**MSIM4302**  
**Edisi 1**

**MODUL 09**  
**Desain/Perancangan**  
**Sistem**

Bahar, S.T., M.Kom.

## Daftar Isi

<b>Modul 09</b>	
<b>9.1</b>	
Desain/Perancangan Sistem	
<b>Kegiatan Belajar 1</b>	9.5
Desain Arsitektur Sistem	
Latihan	9.20
Rangkuman	9.22
Tes Formatif 1	9.23
<b>Kegiatan Belajar 2</b>	9.26
Desain Antarmuka Pengguna	
Latihan	9.51
Rangkuman	9.52
Tes Formatif 2	9.53
Kunci Jawaban Tes Formatif	9.57
Glosarium	9.58
Daftar Pustaka	9.60
Riwayat Penulis	9.61



## Pendahuluan

**F**ase desain/perancangan dalam SDLC menggunakan kebutuhan yang dikumpulkan selama fase analisis untuk membuat cetak biru sistem yang dikembangkan. Desain yang sukses dibangun berdasarkan apa yang telah dikaji pada fase sebelumnya, dengan membuat rencana yang jelas dan akurat tentang apa yang perlu dilakukan.

Desain sistem adalah penentuan arsitektur sistem secara keseluruhan yang akan memenuhi kebutuhan esensial sistem (Dennis, 2012). Tim proyek dengan hati-hati mempertimbangkan kebutuhan bisnis nonfungsional yang diidentifikasi selama analisis. Kebutuhan bisnis nonfungsional memengaruhi kebutuhan sistem yang mendorong desain arsitektur sistem. Tim proyek perlu merencanakan kinerja sistem baru: seberapa cepat sistem akan beroperasi, berapa kapasitasnya, dan ketersediaan serta keandalannya. Tim perlu membuat sistem yang aman dengan menentukan batasan akses dan dengan mengidentifikasi kebutuhan untuk enkripsi, autentikasi, dan pengendalian virus. Kebutuhan nonfungsional diubah menjadi kebutuhan sistem yang dijelaskan dalam dokumen desain arsitektur.

Interaksi pengguna dengan sistem juga harus dirancang. Masukan (input) dan keluaran (output) sistem dirancang bersama dengan rencana atau peta jalan cara fitur sistem akan dinavigasi. Keputusan desain yang dibuat mengenai antarmuka dikomunikasikan melalui dokumen desain yang disebut desain antarmuka.

Komponen lain yang juga perlu dirancang sebelum diimplementasikan adalah sistem data yang dijelaskan dalam model data logis. Keputusan desain mengenai penyimpanan data ditulis dalam dokumen desain penyimpanan data.

Meskipun pada umumnya buku teks harus menyajikan informasi ini secara berurutan, banyak aktivitas dari fase desain yang saling berkaitan dengan fase sebelumnya. Seperti halnya langkah-langkah dalam fase analisis, analis sering bolak-balik di antara langkah-langkah tersebut. Misalnya, pada saat membuat desain antarmuka, sering kali mengungkap informasi tambahan yang diperlukan dalam sistem, yang mengarah ke revisi DFD fisik atau ERD.

Di akhir fase desain, tim proyek membuat hasil akhir untuk fase yang disebut spesifikasi sistem. Dokumen ini berisi semua dokumen desain berupa: model proses fisik, model data fisik, desain arsitektur, spesifikasi perangkat keras dan perangkat lunak, desain antarmuka, desain penyimpanan data, dan desain program. Secara kolektif, spesifikasi sistem menyampaikan dengan tepat sistem apa yang akan diterapkan oleh tim proyek selama fase implementasi SDLC.

Karena keterbatasan ruang dalam modul ini untuk membahas keseluruhan unsur desain sistem, maka pada tahap awal penyajian modul ini hanya akan fokus pada dua hal paling mendasar dalam desain sistem, yaitu desain arsitektur sistem dan desain antarmuka pengguna.

#### 9.4 Desain/Perancangan Sistem

---

Setelah mempelajari modul ini mahasiswa diharapkan secara lebih rinci mampu:

1. menjelaskan elemen/komponen utama desain arsitektur sistem informasi;
2. menjelaskan model-model arsitektur sistem informasi;
3. menjelaskan keterkaitan antara kebutuhan nonfungsional dan desain arsitektur sistem;
4. menjelaskan prinsip-prinsip desain antarmuka pengguna;
5. menjelaskan komponen-komponen utama antarmuka pengguna;
6. menjelaskan proses desain antarmuka pengguna;
7. membuat desain antarmuka pengguna.

## Kegiatan Belajar

1

# Desain Arsitektur Sistem

**K**omponen penting dari fase desain/perancangan adalah desain arsitektur, yang menggambarkan perangkat keras sistem, perangkat lunak, dan lingkungan jaringan. Desain arsitektur mengalir terutama dari kebutuhan nonfungsional, seperti kebutuhan operasional, kinerja, keamanan, budaya, dan politik. Hasil dari desain arsitektur berupa desain arsitektur dan spesifikasi perangkat keras dan perangkat lunak.

Saat ini, kebanyakan sistem informasi tersebar di dua atau lebih komputer. Sistem berbasis web, misalnya, dapat berjalan di browser pada komputer desktop, tetapi akan berinteraksi dengan server Web (dan mungkin komputer lain) melalui internet. Sistem yang beroperasi sepenuhnya di dalam jaringan perusahaan mungkin telah menginstal program Visual Basic di komputer desktop, tetapi berinteraksi dengan server database di tempat lain di jaringan perusahaan. Langkah penting dari tahap desain adalah pembuatan desain arsitektur untuk mendukung rencana bagaimana komponen sistem informasi akan didistribusikan ke beberapa komputer dan perangkat keras.

Merancang arsitektur sistem bisa jadi sangat sulit. Oleh karena itu, banyak organisasi menggunakan keterampilan dari pakar arsitektur sistem yang berpengalaman (konsultan atau karyawan) yang mengkhususkan diri dalam profesi tersebut. Dalam modul ini, kita akan meninjau faktor-faktor kunci dalam desain arsitektur. Kebutuhan nonfungsional yang dikembangkan pada tahap awal fase analisis memainkan peran kunci dalam desain arsitektur. Kebutuhan ini diperiksa ulang dan disempurnakan menjadi kebutuhan yang lebih detail yang memengaruhi arsitektur sistem.

Modul ini, pertama-tama menjelaskan bagaimana para desainer berpikir tentang arsitektur sistem aplikasi dan menjelaskan arsitektur yang paling umum digunakan saat ini, arsitektur klien-server. Selanjutnya secara singkat membahas beberapa arsitektur yang kurang umum (berbasis server dan berbasis klien), serta perkembangan baru dalam arsitektur sistem (seperti *cloud computing* dan virtualisasi).

### A. ELEMEN-ELEMEN DESAIN ARSITEKTUR

Tujuan dari desain arsitektur adalah untuk menentukan bagaimana komponen perangkat lunak dari sistem informasi akan *ditugaskan* ke perangkat keras sistem. Pada bagian ini, pertama-tama kita membahas fungsi-fungsi utama perangkat lunak untuk memahami bagaimana perangkat lunak dapat dibagi menjadi beberapa bagian.

Kemudian secara singkat membahas tipe utama dari perangkat keras yang menjadi tempat penempatan perangkat lunak. Meskipun ada banyak cara di mana komponen perangkat lunak dapat ditempatkan pada komponen perangkat keras, arsitektur yang paling umum adalah arsitektur klien-server, jadi pembahasan difokuskan pada arsitektur klien-server.

### 1. Komponen Arsitektur

Komponen utama arsitektur dari setiap sistem informasi adalah perangkat lunak dan perangkat keras. Komponen perangkat lunak utama dari sistem yang sedang dikembangkan harus diidentifikasi dan kemudian dialokasikan ke berbagai komponen perangkat keras tempat sistem akan beroperasi. Masing-masing komponen ini dapat digabungkan dengan berbagai cara.

Semua sistem perangkat lunak dapat dibagi menjadi empat fungsi dasar. Fungsi yang **pertama** adalah fungsi penyimpanan data; sebagian besar sistem informasi memerlukan data untuk disimpan dan diambil. Ini adalah entitas data yang didokumentasikan dalam ERD. Fungsi **kedua** adalah logika akses data; pemrosesan yang diperlukan untuk mengakses data, sering kali kueri database dalam *Structured Query Language* (SQL). Fungsi **ketiga** adalah logika aplikasi; logika yang didokumentasikan di DFD, *use case*, dan kebutuhan fungsional. Fungsi **keempat** adalah logika presentasi; tampilan informasi kepada pengguna dan penerimaan perintah pengguna (antarmuka pengguna). Keempat fungsi ini (penyimpanan data, logika akses data, logika aplikasi, dan logika presentasi) adalah blok bangunan dasar dari setiap sistem informasi.

Tiga komponen utama perangkat keras dari suatu sistem adalah komputer klien, server, dan jaringan yang menghubungkannya. Komputer klien adalah perangkat input-output yang digunakan oleh pengguna dan biasanya merupakan komputer desktop atau laptop, tetapi juga dapat berupa perangkat genggam, *smartphone*, dan terminal khusus lainnya. Server biasanya adalah komputer multi-user yang memiliki spesifikasi teknis lebih tinggi yang digunakan untuk menyimpan perangkat lunak dan data yang dapat diakses oleh siapa saja yang memiliki hak akses. Jaringan yang menghubungkan komputer, dengan kecepatan yang bervariasi; mulai dari yang berkecepatan lambat (ponsel atau koneksi modem), jaringan *frame relay* berkecepatan sedang, hingga koneksi *broadband* yang cepat, seperti modem kabel, DSL, sirkuit *ethernet*, dan sebagainya.

### 2. Arsitektur Klien-Server

Sebagian besar organisasi saat ini menggunakan model arsitektur klien-server, yang berusaha menyeimbangkan pemrosesan antara perangkat klien dan satu atau lebih perangkat server. Dalam arsitektur ini, klien bertanggung jawab atas logika presentasi, sedangkan server bertanggung jawab atas logika akses data dan penyimpanan data. Logika aplikasi mungkin berada di klien, berada di server, atau dibagi di antara

keduanya (Gambar 9.1). Saat ini, *thin client element*, yang hanya berisi sebagian kecil dari logika aplikasi, populer karena *overhead* yang lebih rendah dan perawatan yang lebih mudah. Misalnya, banyak sistem berbasis Web dirancang dengan browser Web melakukan presentasi dan hanya menggunakan logika aplikasi minimal yang menggunakan bahasa pemrograman seperti JavaScript, sedangkan sisi server memiliki sebagian besar logika aplikasi, semua logika akses data, dan semua penyimpanan data.

Arsitektur klien-server memiliki beberapa kelebihan. **Pertama** dan terpenting, mereka dapat diskalakan. Artinya, mudah untuk menambah atau mengurangi kapasitas penyimpanan dan pemrosesan server. Jika satu server kelebihan beban, cukup menambahkan server lain sehingga banyak server digunakan untuk menjalankan logika aplikasi, logika akses data, atau penyimpanan data. Biaya untuk memutakhirkannya bersifat bertahap, dan kita dapat meningkatkan kinerjanya sedikit demi sedikit.

**Kedua**, arsitektur klien-server dapat mendukung berbagai jenis klien dan server. Dimungkinkan untuk menyambungkan komputer yang menggunakan sistem operasi berbeda sehingga pengembang tidak terikat dalam satu vendor. Pengguna dapat memilih jenis komputer yang mereka sukai (misalnya menggabungkan komputer Windows dan Apple Macintosh di jaringan yang sama).

**Ketiga**, untuk arsitektur klient-server bertingkat yang menggunakan standar Internet, sangat mudah untuk memisahkan dengan jelas logika presentasi, logika aplikasi, dan logika akses data agar lebih independen.



Gambar 9.1  
Arsitektur Klien-Server Dua Tingkat

**Keempat**, jika server gagal dalam arsitektur klien-server, hanya aplikasi yang memerlukan server tersebut yang akan gagal. Server yang gagal dapat diganti dan aplikasi kemudian dapat dipulihkan.

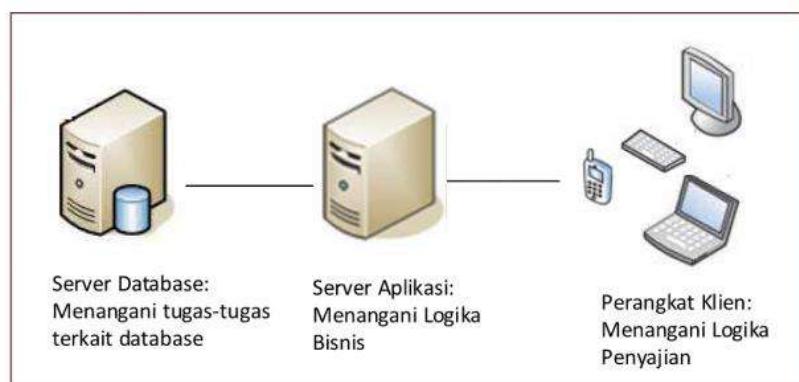
Arsitektur klien-server juga memiliki beberapa keterbatasan, yang paling penting adalah kompleksitasnya. Semua aplikasi dalam komputasi klien-server memiliki dua bagian, perangkat lunak di sisi klien dan perangkat lunak di sisi server. Penulisan kode program/perangkat lunak ini lebih rumit daripada penulisan kode program/perangkat lunak tradisional yang digunakan dalam arsitektur berbasis server. Memperbarui sistem secara keseluruhan dengan perangkat lunak versi baru juga lebih rumit. Dengan

arsitektur klien-server, kita harus memperbarui semua klien dan semua server dan kita harus memastikan bahwa pembaruan diterapkan pada semua perangkat.

### 3. Klien-Server Bertingkat

Ada banyak cara di mana logika aplikasi dapat dipartisi antara klien dan server. Susunan pada Gambar 9.1 adalah konfigurasi umum. Dalam hal ini, server bertanggung jawab atas data dan klien bertanggung jawab atas aplikasi dan presentasi (penyajian). Ini disebut arsitektur dua tingkat karena hanya menggunakan dua set komputer (klien dan server).

Arsitektur tiga tingkat menggunakan tiga set komputer, seperti yang ditunjukkan pada Gambar 9.2. Dalam hal ini, perangkat lunak pada komputer klien bertanggung jawab atas logika presentasi, server aplikasi bertanggung jawab atas logika aplikasi, dan server basis data terpisah bertanggung jawab atas logika akses data dan penyimpanan data. Biasanya, antarmuka pengguna berjalan pada PC desktop atau *workstation* dan menggunakan antarmuka pengguna grafis standar. Logika aplikasi dapat terdiri dari satu atau lebih modul terpisah yang berjalan di *workstation* atau *server* aplikasi. Terakhir, DBMS relasional yang berjalan pada server database berisi logika akses data dan penyimpanan data. Tingkat menengah dapat dibagi menjadi beberapa tingkatan lagi, menghasilkan arsitektur keseluruhan yang disebut “arsitektur *n-tier*” (Gambar 9.3).

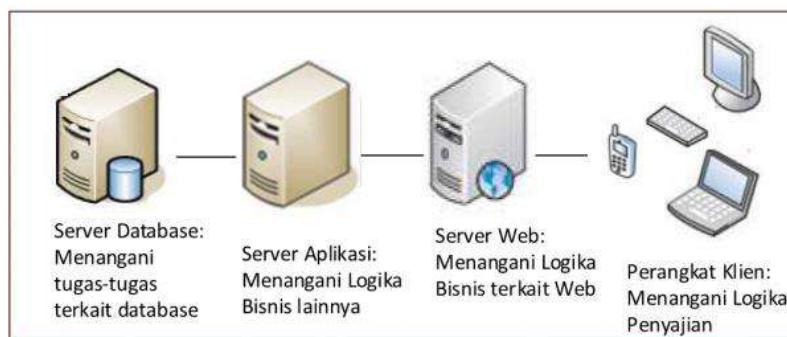


Gambar 9.2  
Arsitektur Klien-Server Tiga Tingkat

Arsitektur *n-tier* mendistribusikan tugas-tugas (pekerjaan) aplikasi di antara beberapa lapisan komputer server yang lebih terspesialisasi. Contoh: arsitektur yang umum digunakan dalam sistem *e-commerce* berbasis web saat ini (Gambar 9.3). Perangkat lunak browser pada komputer klien membuat permintaan HTTP untuk melihat halaman dari server Web, dan server Web memungkinkan pengguna untuk melihat barang dagangan yang akan dijual dengan menanggapi dengan dokumen HTML. Saat pengguna berbelanja, komponen pada server aplikasi dipanggil sesuai kebutuhan untuk memungkinkan pengguna memasukkan item ke dalam keranjang

belanja; menentukan harga dan ketersediaan barang; menghitung biaya pembelian, pajak penjualan, dan biaya pengiriman; mengotorisasi pembayaran, dan lain-lain. Elemen logika bisnis ini, atau pemrosesan terperinci, disimpan di server aplikasi dan dapat diakses oleh aplikasi apapun. Misalnya, aplikasi mesin kasir yang membutuhkan pencarian harga barang dapat menggunakan logika bisnis penentuan harga yang sama dengan yang digunakan oleh situs Web *e-commerce*. Logika bisnis modular dapat digunakan oleh beberapa aplikasi independen yang memerlukan logika bisnis tertentu. Server database mengelola komponen data sistem. Masing-masing dari empat komponen ini terpisah, yang memudahkan penyebaran komponen yang berbeda di server yang berbeda dan untuk mempartisi logika aplikasi pada server berorientasi Web dan server berorientasi bisnis.

Keuntungan utama dari arsitektur klien-server *n-tier* (tiga tingkat atau lebih) dibandingkan dengan arsitektur dua-tingkat adalah bahwa *n-tier* memisahkan pemrosesan yang terjadi untuk lebih menyeimbangkan beban pada server yang berbeda. Pada Gambar 9.3, kita memiliki tiga jenis server terpisah, konfigurasi yang menyediakan lebih banyak sumber daya daripada jika kita menggunakan arsitektur dua tingkat atau hanya satu server. Jika kita menemukan bahwa server aplikasi terlalu banyak beban, kita dapat menggantinya dengan server yang lebih kuat atau cukup memasukkan beberapa server aplikasi lagi untuk berbagi beban. Sebaliknya, jika kita menemukan bahwa server database kurang digunakan, kita dapat menyimpan data dari aplikasi lain di dalamnya.



Gambar 9.3  
Arsitektur Klien-Server *n-Tier*

Ada dua kelemahan utama arsitektur *n-tier* dibandingkan dengan arsitektur dua tingkat. **Pertama**, konfigurasi memberikan beban yang lebih besar pada jaringan. Jika kita membandingkan Gambar 9.1, 9.2, dan 9.3, kita akan melihat bahwa model *n-tier* membutuhkan lebih banyak komunikasi di antara server; ini membutuhkan lebih banyak lalu lintas jaringan, jadi kita memerlukan jaringan dengan kapasitas (*bandwidth*) lebih tinggi. **Kedua**, jauh lebih sulit untuk menulis program dan menguji perangkat lunak

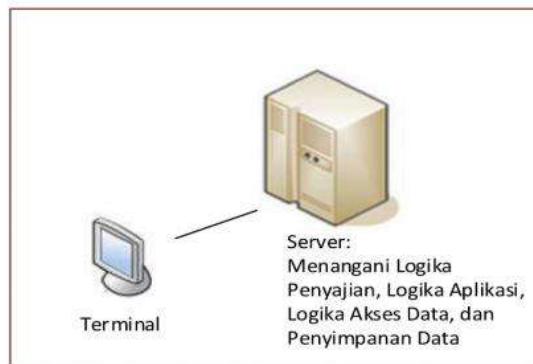
dalam arsitektur *n-tier* daripada dalam arsitektur dua-tingkat, karena lebih banyak perangkat harus berkomunikasi dengan benar untuk menyelesaikan transaksi pengguna.

#### 4. Arsitektur yang Tidak Umum

Arsitektur klien-server telah menjadi arsitektur utama yang digunakan saat ini. Dua arsitektur lain jarang ditemukan, tetapi masih digunakan dalam situasi tertentu.

##### a. Arsitektur Berbasis Server

Arsitektur komputasi pertama berbasis server, dengan server (biasanya berupa komputer *mainframe*) melakukan keempat fungsi dasar perangkat lunak. Klien (biasanya berupa terminal komputer) memungkinkan pengguna untuk mengirim dan menerima pesan ke dan dari komputer server. Klien hanya berfungsi sebagai alat masukan dan mengirimkannya ke server untuk diproses, selanjutnya menerima instruksi dari server tentang apa yang akan ditampilkan (Gambar 9.4).



**Gambar 9.4**  
Arsitektur Berbasis Server

Arsitektur yang sangat sederhana ini seringkali bekerja dengan sangat baik. Perangkat lunak aplikasi dikembangkan dan disimpan di server, dan semua data ada di komputer yang sama. Hanya ada satu titik kendali karena semua pesan mengalir melalui satu server pusat. Pengembangan perangkat lunak dan administrasi perangkat lunak disederhanakan karena satu komputer menampung seluruh sistem (sistem operasi dan perangkat lunak aplikasi).

Arsitektur berbasis server adalah arsitektur pertama yang digunakan dalam sistem informasi, tetapi tidak menjadi satu-satunya pilihan karena perangkat keras dan perangkat lunak berkembang. Masalah mendasar dengan sistem berbasis server adalah bahwa server memproses semua pekerjaan di sistem. Semakin banyak aplikasi dan jumlah pengguna bertambah, komputer server menjadi kelebihan beban dan tidak dapat memproses semua permintaan pengguna dengan cepat. Waktu respons menjadi lebih lambat, dan administrator mengeluarkan lebih banyak uang untuk meningkatkan kinerja

server komputer. Pada masa-masa awal, mengupgrade ke komputer server yang lebih besar (biasanya *mainframe*) membutuhkan komitmen finansial yang besar.

Saat ini, arsitektur berbasis server tetap menjadi pilihan arsitektur yang masih layak. *Zero client* atau *ultrathin client*, adalah model komputasi berbasis server yang sering digunakan saat ini dalam *Virtual Desktop Infrastructure/VDI*. Perangkat *zero client* khas adalah kotak kecil yang menghubungkan keyboard, mouse, monitor, dan koneksi *ethernet* ke server yang letaknya jauh. Server menghosting semuanya: sistem operasi klien dan semua aplikasi perangkat lunak. Server dapat diakses secara nirkabel atau dengan kabel. Komputasi *zero client* memiliki sejumlah keuntungan. Penggunaan daya dapat dikurangi secara signifikan dibandingkan dengan konfigurasi klien yang besar. Manfaat ini semakin penting karena semakin banyak perusahaan yang meneliti tentang *green computing*. Perangkat yang digunakan jauh lebih murah daripada PC atau bahkan perangkat klien kecil. Karena tidak ada perangkat lunak di perangkat klien, tidak ada kerentanan terhadap *malware*.

b. *Arsitektur Berbasis Klien*

Untuk arsitektur berbasis klien, klien adalah komputer mikro di jaringan area lokal, dan server adalah komputer server di jaringan yang sama. Perangkat lunak aplikasi pada komputer klien bertanggung jawab atas logika presentasi, logika aplikasi, dan logika akses data; server hanya menyediakan penyimpanan untuk data (Gambar 9.5).

Arsitektur sederhana ini sering kali berfungsi dengan baik dalam situasi jumlah pengguna yang sedikit atau persyaratan akses data yang terbatas. Masalah mendasar dalam arsitektur berbasis klien adalah bahwa semua data di server harus dikirim ke klien untuk diproses. Misalnya, pengguna ingin menampilkan daftar semua karyawan dengan asuransi jiwa ditanggung perusahaan. Semua data dalam database karyawan harus berpindah dari server, tempat database disimpan, melalui jaringan ke klien, yang kemudian mengeksekusi kueri untuk menemukan setiap *record* yang cocok dengan data yang diminta oleh pengguna. Dalam model komputasi lain yang telah kita bahas, logika akses data akan dieksekusi di server dan hanya hasil kueri yang dikirimkan ke klien. Dalam model komputasi berbasis klien, logika akses data dijalankan pada sistem klien. Oleh karena itu, seluruh database harus dikirimkan ke klien sebelum pemrosesan dapat dilakukan. Ini dapat membebani jaringan dan daya komputer klien.



Gambar 9.5  
Arsitektur Berbasis Klien

## 5. Perkembangan Konfigurasi Arsitektur

Kemajuan dalam perangkat keras, perangkat lunak, dan jaringan telah memunculkan sejumlah opsi arsitektur baru. Pembahasan mendetail tentang semua opsi tidak disajikan dalam modul ini. Dua kemajuan yang saat ini banyak mendapat perhatian, virtualisasi dan komputasi awan (*cloud computing*), akan dijelaskan secara singkat di sini.

### a. Virtualisasi

Istilah virtualisasi dalam domain komputasi, mengacu pada pembuatan perangkat atau sumber daya virtual, seperti virtualisasi server atau perangkat penyimpanan, virtualisasi jaringan, dan variasi virtualisasi lainnya.

Virtualisasi server melibatkan partisi server fisik menjadi server virtual yang lebih kecil. Perangkat lunak digunakan untuk membagi server fisik menjadi beberapa lingkungan virtual, yang disebut server virtual atau pribadi. Kemampuan ini mengatasi keterbatasan utama dari arsitektur berbasis server gaya lama yang didasarkan pada komputer tunggal, besar, mahal, dan monolitik. Saat ini, perangkat server fisik dapat digunakan untuk menyediakan banyak server virtual yang tidak bergantung satu sama lain, tetapi berada di satu server fisik yang sama. Setiap server virtual menjalankan sistem operasi dan dapat di-boot ulang secara independen dari server virtual lainnya. Lebih sedikit perangkat keras diperlukan untuk menyediakan satu set server virtual dibandingkan dengan server fisik yang setara, sehingga biaya berkurang. Pengaturan ini juga dapat mengoptimalkan pemanfaatan server fisik, sehingga menghemat biaya operasional.

Virtualisasi penyimpanan melibatkan penggabungan beberapa perangkat penyimpanan jaringan ke dalam apa yang tampak seperti unit penyimpanan tunggal. Jaringan area penyimpanan (*Storage Area Network/SAN*) menggunakan virtualisasi penyimpanan untuk membuat subjaringan berkecepatan tinggi dari perangkat penyimpanan bersama. Dalam lingkungan ini, tugas-tugas seperti pencadangan, pengarsipan, dan pemulihan menjadi lebih mudah dan lebih cepat.

b. *Komputasi Awan (Cloud Computing)*

Organisasi bisnis tidak lagi perlu memiliki dan mengelola infrastruktur komputasi mereka sendiri. Saat ini bisnis berada di lingkungan kebangkitan sistem *cloud computing*, di mana segala sesuatu, mulai dari daya komputasi hingga infrastruktur komputasi, aplikasi, proses bisnis hingga kolaborasi pribadi dapat dihadirkan sebagai sistem layanan "di manapun dan kapanpun diperlukan". "*Cloud*" dalam *cloud computing* dapat didefinisikan sebagai sekumpulan perangkat keras, jaringan, penyimpanan, layanan, dan antarmuka yang digabungkan untuk memberikan aspek komputasi sebagai layanan. Layanan *cloud* mencakup pengiriman perangkat lunak, infrastruktur, dan penyimpanan melalui Internet (baik sebagai komponen terpisah atau platform lengkap) berdasarkan permintaan pengguna.

*Cloud computing* dapat diimplementasikan dalam tiga cara: *cloud* pribadi, *cloud* publik, dan *cloud* hybrid. Dengan ***cloud* publik**, layanan disediakan "sebagai layanan" melalui Internet dengan sedikit atau tanpa kendali atas infrastruktur teknologi yang mendasarinya. ***Cloud* pribadi** menawarkan aktivitas dan fungsi "sebagai layanan", tetapi diterapkan melalui intranet perusahaan atau pusat data yang dihosting. ***Cloud* hybrid** menggabungkan kekuatan *cloud* publik dan *cloud* pribadi. Dalam skenario ini, aktivitas dan tugas dialokasikan ke *cloud* pribadi atau publik sesuai kebutuhan.

Kehadiran teknologi *cloud computing* sebagai teknologi pendukung sistem menunjukkan sejumlah keunggulan. **Pertama**, saat memanfaatkan *cloud*, sumber daya yang dialokasikan dapat ditingkatkan atau dikurangi berdasarkan permintaan. Kemampuan ini, disebut elastisitas, membuat *cloud* dapat diskalakan (misalnya ditingkatkan untuk periode permintaan puncak dan diturunkan saat permintaan lebih sedikit). Aplikasi di *cloud* dapat ditingkatkan seiring dengan penambahan pengguna dan saat kebutuhan aplikasi berubah. **Kedua**, pelanggan *cloud* dapat memperoleh sumber daya *cloud* dengan cara yang langsung. Pengaturan telah disediakan oleh penyedia layanan *cloud* untuk sejumlah komputasi, penyimpanan, perangkat lunak, proses, atau sumber daya tertentu. Setelah menggunakan sumber daya ini, mereka dapat dirilis jika tidak lagi diperlukan. **Ketiga**, layanan *cloud* biasanya memiliki standar antarmuka program aplikasi (*Application Programming Interface/API*). Ini berarti bahwa layanan telah menstandarkan cara program atau sumber data berkomunikasi satu sama lain. Kemampuan ini memungkinkan pelanggan lebih mudah membuat keterkaitan antar layanan *cloud*. **Keempat**, model *cloud computing* memungkinkan pelanggan ditagih untuk sumber daya saat digunakan. Penggunaan *cloud* diukur dan pelanggan hanya membayar sumber daya yang digunakan (mirip seperti penggunaan listrik di rumah kita). Fitur ini membuat *cloud computing* sangat menarik dari perspektif keuangan.

Vendor *cloud computing* memanfaatkan virtualisasi sebagai teknologi pendukung utama. Namun, bagi pelanggan *cloud computing*, intinya adalah mengalihdayakan teknologi, aplikasi, dan keterampilan Teknologi Informasi dengan model pembayaran per penggunaan. Konsep *cloud computing* telah menarik perhatian

dan imajinasi organisasi dari semua ukuran. Melalui model *cloud computing*, kekuatan virtualisasi diubah menjadi nilai bisnis yang dapat diukur.

### 6. Membandingkan Arsitektur

Sebagian besar sistem dibangun dengan menggunakan infrastruktur yang ada dalam organisasi, sehingga seringkali infrastruktur saat ini membatasi pilihan arsitektur. Misalnya, jika sistem baru akan dibangun untuk organisasi yang berpusat pada *mainframe*, arsitektur berbasis server mungkin merupakan pilihan terbaik. Faktor lain seperti standar perusahaan, perjanjian lisensi yang ada, dan hubungan produk/vendor juga dapat menjadi landasan bagi tim proyek memilih model arsitektur. Banyak organisasi bisnis sekarang memiliki beragam infrastruktur yang digunakan, atau secara aktif mencari proyek percontohan untuk menguji arsitektur dan infrastruktur baru, memungkinkan tim proyek untuk memilih arsitektur berdasarkan faktor penting lainnya.

Masing-masing arsitektur yang telah dibahas memiliki kelebihan dan kekurangannya masing-masing. Arsitektur klien-server sangat disukai berdasarkan biaya infrastruktur (perangkat keras, perangkat lunak, dan jaringan yang akan mendukung sistem aplikasi). Arsitektur klien-server sangat *scalable* karena server dapat ditambahkan ke (atau dihapus dari) infrastruktur saat pemrosesan perlu diubah. Alat pengembangan GUI yang digunakan untuk membuat aplikasi untuk arsitektur klien-server bisa intuitif dan mudah digunakan. Pengembangan aplikasi untuk arsitektur ini bisa cepat dan tidak menyulitkan. Perlu diingat, bagaimanapun, arsitektur klien-server memang melibatkan kompleksitas tambahan dari beberapa lapisan perangkat keras (misalnya server basis data, server Web, terminal kerja klien) yang harus berkomunikasi secara efektif satu sama lain. Tim proyek sering kali meremehkan kesulitan yang terkait dengan pembuatan aplikasi klien-server yang aman dan efisien.

## B. MEMBANGUN DESAIN ARSITEKTUR

Desain arsitektur menentukan arsitektur keseluruhan dan penempatan perangkat lunak dan perangkat keras yang akan digunakan. Desain arsitektur adalah proses yang sangat kompleks yang sering kali diserahkan kepada desainer dan konsultan arsitektur berpengalaman, namun kita akan memberikan gambaran prosesnya pada pembahasan ini.

Membuat desain arsitektur dimulai dengan kebutuhan nonfungsional. Langkah pertama adalah menyempurnakan kebutuhan nonfungsional menjadi kebutuhan yang lebih detail yang kemudian digunakan untuk membantu memilih arsitektur yang akan digunakan (berbasis server, berbasis klien, atau klien-server) dan komponen perangkat lunak yang akan ditempatkan pada setiap perangkat. Dalam arsitektur klien-server, seseorang juga harus memutuskan apakah akan menggunakan arsitektur dua tingkat, tiga tingkat, atau n-tingkat. Kebutuhan nonfungsional dan desain arsitektur akan digunakan untuk mengembangkan spesifikasi perangkat keras dan perangkat lunak.

Ada empat jenis kebutuhan nonfungsional utama yang dapat menjadi penting dalam merancang arsitektur: kebutuhan operasional, kebutuhan kinerja, kebutuhan keamanan, dan kebutuhan budaya dan politik (Dennis, 2012). Kita akan menjelaskan masing-masing dan kemudian menjelaskan bagaimana kebutuhan-kebutuhan ini dapat mempengaruhi desain arsitektur.

### 1. Kebutuhan Operasional

Kebutuhan operasional menentukan lingkungan operasi di mana sistem harus bekerja dan bagaimana hal itu dapat berubah seiring waktu. Ini biasanya mengacu pada sistem operasi, perangkat lunak sistem, dan sistem informasi yang harus berinteraksi dengan sistem, tetapi terkadang juga mencakup lingkungan fisik jika lingkungan itu penting untuk aplikasi. Gambar 9.6 merangkum empat bidang kebutuhan operasional utama dan memberikan beberapa contoh untuk masing-masing bidang.

Jenis Kebutuhan	Definisi	Contoh
Kebutuhan Lingkungan Teknis	Kebutuhan perangkat keras, perangkat lunak, dan jaringan khusus yang dipersyaratkan oleh kebutuhan bisnis.	<ul style="list-style-type: none"> <li>Sistem akan bekerja melalui lingkungan Web dengan Internet Explorer.</li> <li>• Semua lokasi kantor akan memiliki koneksi jaringan yang selalu aktif untuk memungkinkan update database secara real-time.</li> <li>• Sebuah versi sistem akan disediakan bagi pelanggan yang terhubung melalui Internet melalui <i>smartphone</i> (layar kecil).</li> </ul>
Kebutuhan Integrasi Sistem	Sejauh mana sistem tersebut akan beroperasi (terintegrasi) dengan sistem lain.	<ul style="list-style-type: none"> <li>Sistem harus dapat mengimpor dan mengekspor spreadsheet Excel.</li> <li>• Sistem akan mengambil dan menyimpan ke database inventaris utama di sistem inventaris.</li> </ul>
Kebutuhan Portabilitas	Sejauh mana sistem perlu beroperasi di lingkungan lain.	<ul style="list-style-type: none"> <li>Sistem harus dapat bekerja dalam lingkungan sistem operasi yang berbeda (misalnya: Linux, Windows 8).</li> <li>• Sistem mungkin perlu beroperasi pada perangkat genggam seperti iPad.</li> </ul>
Lingkungan Pemeliharaan	Perubahan bisnis yang diharapkan sehingga sistem harus dapat beradaptasi.	<ul style="list-style-type: none"> <li>Sistem akan dapat mendukung lebih dari satu pemasok dengan pemberitahuan enam bulan sebelumnya.</li> <li>• Versi baru sistem akan dirilis setiap enam bulan.</li> </ul>

Gambar 9.6  
Kebutuhan Operasional

## 2. Kebutuhan Kinerja

Kebutuhan kinerja berfokus pada masalah kinerja seperti waktu respons, kapasitas, dan keandalan. Sebagai analis yang mendefinisikan kebutuhan kinerja untuk sistem, uji kebutuhan tersebut adalah masalah utama. Setiap kebutuhan harus dapat diukur agar dapat dibuat perbandingan patokan. Hanya dengan cara itu pencapaian kebutuhan kinerja dapat diverifikasi. Gambar 9.7 merangkum tiga bidang kebutuhan kinerja utama dan memberikan beberapa contoh.

Jenis Kebutuhan	Definisi	Contoh
Kebutuhan Kecepatan	Waktu di mana sistem harus menjalankan fungsinya.	<ul style="list-style-type: none"> <li>• Waktu respons harus paling lambat 4 detik untuk setiap transaksi melalui jaringan.</li> <li>• Database inventaris harus dapat diperbarui secara <i>real time</i>.</li> <li>• Pesanan akan dikirim ke ruang pabrik setiap 3 menit.</li> </ul>
Kebutuhan Kapasitas	Jumlah beban puncak pengguna serta volume data yang diharapkan.	<ul style="list-style-type: none"> <li>• Maksimal 2000 pengguna secara bersamaan pada waktu penggunaan puncak.</li> <li>• Transaksi khusus akan membutuhkan transmisi 300 KB data.</li> <li>• Sistem akan menyimpan data sekitar 50.000 pelanggan dengan total sekitar 2 GB data.</li> </ul>
Kebutuhan Ketersediaan dan Keandalan	Sejauh mana sistem akan tersedia untuk pengguna dan tingkat kegagalan yang diizinkan karena kesalahan.	<ul style="list-style-type: none"> <li>• Sistem harus tersedia 24 jam sehari dan 7 hari dalam seminggu, dengan pengecualian pemeliharaan terjadwal.</li> <li>• Pemeliharaan terjadwal tidak boleh melebihi satu periode 6 jam setiap bulan.</li> <li>• Sistem akan memiliki kinerja waktu aktif 99%.</li> </ul>

Gambar 9.7  
Kebutuhan Kinerja

## 3. Kebutuhan Keamanan

Keamanan adalah kemampuan untuk melindungi sistem informasi dari gangguan dan kehilangan data, baik yang disebabkan oleh tindakan yang disengaja (misalnya peretas) atau peristiwa acak (misalnya kegagalan disk).

Pengembang sistem harus mengetahui standar keamanan yang berlaku untuk organisasi mereka. Beberapa industri tunduk pada berbagai batasan yang diberlakukan oleh hukum. Misalnya, peraturan perundang-undangan yang berlaku untuk penyedia layanan kesehatan, perusahaan asuransi kesehatan, dan pemroses data informasi kesehatan, sedangkan industri keuangan harus memenuhi persyaratan sesuai regulasi tertentu. Ada juga standar keamanan umum yang dapat dipilih perusahaan untuk

dijadikan tolok ukur, seperti standar yang diadopsi oleh *International Organization for Standardization* (ISO).

Keamanan dalam sistem biasanya berfokus pada menentukan siapa yang dapat mengakses data apa, mengidentifikasi kebutuhan untuk enkripsi dan autentikasi, dan memastikan bahwa aplikasi tersebut mencegah penyebaran virus (lihat Gambar 9.8.)

Jenis Kebutuhan	Definisi	Contoh
Estimasi Nilai Sistem	Estimasi nilai bisnis dari sistem dan datanya.	<ul style="list-style-type: none"> <li>Berhentinya operasi sistem diperkirakan menelan kerugian \$ 150.000 per jam.</li> <li>Hilangnya seluruh data sistem diperkirakan menelan biaya \$ 20 juta.</li> </ul>
Kebutuhan Pengendalian Akses	Batasan siapa yang dapat mengakses data apa.	<ul style="list-style-type: none"> <li>Hanya manajer departemen yang dapat mengubah item inventaris dalam departemen mereka sendiri.</li> <li>Personel layanan pelanggan akan dapat membaca dan membuat item di file pelanggan, tetapi tidak dapat mengubah atau menghapus item.</li> </ul>
Kebutuhan Enkripsi dan Autentikasi	Menentukan data apa yang akan dienkripsi di mana dan apakah autentikasi akan diperlukan untuk akses pengguna.	<ul style="list-style-type: none"> <li>Data akan dienkripsi dari komputer pengguna ke situs Web untuk menyediakan pemesanan yang aman.</li> <li>Pengguna yang masuk dari luar kantor akan diminta untuk melakukan autentikasi.</li> </ul>
Kebutuhan Pengendalian Virus	Mengontrol penyebaran virus.	<ul style="list-style-type: none"> <li>Semua file yang diunggah akan diperiksa virusnya sebelum disimpan di sistem.</li> </ul>

**Gambar 9.8**  
Kebutuhan Keamanan

#### 4. Kebutuhan Budaya dan Politik

Kebutuhan budaya dan politik terkait dengan negara atau lingkungan tempat sistem akan digunakan. Dalam lingkungan bisnis global saat ini, organisasi memperluas sistem mereka untuk menjangkau pengguna di seluruh dunia. Meskipun hal ini dari segi bisnis masuk akal, dampaknya pada pengembangan aplikasi tidak boleh diremehkan. Bagian penting lainnya dari desain arsitektur sistem adalah memahami kebutuhan budaya dan politik global untuk sistem tersebut (lihat Gambar 9.9).

Jenis Kebutuhan	Definisi	Contoh
Kebutuhan Multibahasa	Bahasa yang dibutuhkan sistem untuk beroperasi.	Sistem akan beroperasi dalam Bahasa Indonesia dan Bahasa Inggris.
Kebutuhan Kustomisasi	Spesifikasi dari aspek sistem apa yang dapat diubah oleh pengguna lokal.	Kebijakan negara mungkin akan dapat mengubah format nomor telepon di database pelanggan.
Membuat Eksplisit Aturan yang Tidak Disebutkan	Menyatakan secara eksplisit asumsi yang berbeda antar negara.	<ul style="list-style-type: none"> <li>• Semua isian tanggal akan menggunakan format bulan-hari-tahun.</li> <li>• Semua isian bobot akan dinyatakan dalam kilogram.</li> </ul>
Persyaratan Legalitas	Hukum dan peraturan yang memberlakukan persyaratan pada sistem.	<ul style="list-style-type: none"> <li>• Informasi pribadi tentang pelanggan tidak dapat ditransfer keluar dari negara Indonesia.</li> <li>• Pelanggaran hukum terkait pembocoran informasi seperti mengakses riwayat transaksi pelanggan hanya diizinkan untuk manajer area.</li> </ul>

**Gambar 9.9**  
Kebutuhan Budaya dan Politik

## 5. Merancang Arsitektur Sistem

Dalam banyak kasus, kebutuhan lingkungan teknis yang didorong oleh kebutuhan bisnis mungkin hanya mendefinisikan arsitektur aplikasi. Dalam kasus ini, pilihannya sederhana: kebutuhan bisnis mendominasi pertimbangan lain. Misalnya, kebutuhan bisnis dapat menentukan bahwa sistem perlu beroperasi melalui Web, menggunakan browser Web pelanggan. Dalam hal ini, arsitektur aplikasi harus berupa *thin client-server*. Kebutuhan bisnis seperti itu kemungkinan besar terjadi dalam sistem yang dirancang untuk mendukung pelanggan eksternal. Sistem internal juga dapat memberlakukan kebutuhan bisnis, tetapi biasanya tidak seketat itu (Dennis, 2012).

Berbeda dengan kebutuhan *lingkungan teknis* yang tidak memerlukan pilihan arsitektur tertentu, kebutuhan nonfungsional lainnya menjadi penting. Gambar 9.10 merangkum hubungan antara kebutuhan dan arsitektur yang direkomendasikan.

Kebutuhan	Berbasis Server	Berbasis Client	Thin Client	Thick Client
Kebutuhan Operasional: <ul style="list-style-type: none"> <li>• Kebutuhan Integrasi Sistem</li> <li>• Kebutuhan Portabilitas</li> <li>• Kebutuhan Pemeliharaan</li> </ul>	√		√ √	√
Kebutuhan Kinerja: <ul style="list-style-type: none"> <li>• Kebutuhan Kecepatan</li> </ul>			√	√

Kebutuhan	Berbasis Server	Berbasis Client	Thin Client	Thick Client
<ul style="list-style-type: none"> <li>• Kebutuhan Kapasitas</li> <li>• Kebutuhan Ketersediaan/Keandalan</li> </ul>	√		√ √	√ √
Kebutuhan Keamanan:			√	
<ul style="list-style-type: none"> <li>• Nilai Sistem Tinggi</li> <li>• Kebutuhan Kontrol Akses</li> <li>• Kebutuhan Enkripsi/Autentikasi</li> <li>• Kebutuhan Pengendalian Virus</li> </ul>	√ √ √		√ √	√
Persyaratan Budaya/Politik			√ √ √ √	√
<ul style="list-style-type: none"> <li>• Persyaratan Multibahasa</li> <li>• Persyaratan Kustomisasi</li> <li>• Membuat Eksplisit Aturan yang Tidak Disebutkan</li> <li>• Persyaratan Legalitas</li> </ul>	√		√ √ √ √	√

Gambar 9.10  
Kebutuhan Nonfungsional dan Implikasinya terhadap Desain Arsitektur

**Kebutuhan Operasional.** Kebutuhan integrasi sistem dapat mengarah ke satu arsitektur tertentu, tergantung pada arsitektur dan desain sistem yang perlu diintegrasikan oleh sistem. Misalnya, jika sistem harus berintegrasi dengan sistem desktop (misalnya Excel), maka ini mungkin menyarankan arsitektur klien-server tidak berlapis, sementara jika harus berintegrasi dengan sistem berbasis server, maka arsitektur berbasis server dapat dipilih. Sistem yang memiliki kebutuhan portabilitas yang luas cenderung cocok untuk arsitektur *thin client-server* karena lebih mudah menulis program untuk standar berbasis web (misalnya, HTML, XML) yang memperluas jangkauan sistem ke platform lain daripada menulis ulang logika penyajian ekstensif untuk platform berbeda dalam arsitektur klien-server berbasis server, berbasis klien, atau berlapis. Sistem dengan kebutuhan pemeliharaan yang ekstensif mungkin tidak cocok untuk arsitektur klien-server berbasis klien atau berlapis karena kebutuhan untuk menginstal ulang perangkat lunak pada desktop.

**Kebutuhan Kinerja.** Secara umum, sistem informasi yang memiliki kebutuhan kinerja tinggi paling cocok untuk arsitektur klien-server. Arsitektur klien-server lebih terukur, yang berarti bahwa mereka merespon secara lebih baik kapasitas kebutuhan yang berubah dan dengan demikian memungkinkan organisasi untuk menyesuaikan perangkat keras dengan lebih baik dengan kebutuhan kecepatan sistem. Arsitektur klien-server yang memiliki beberapa server di setiap tingkatan harus lebih andal dan memiliki ketersediaan yang lebih besar, karena jika ada satu server yang bermasalah, permintaan akan diteruskan ke server lain dan pengguna bahkan mungkin tidak menyadarinya. Namun, dalam praktiknya, keandalan dan ketersediaan sangat bergantung pada perangkat keras dan sistem operasi yang digunakan.

**Kebutuhan Keamanan.** Secara umum, arsitektur berbasis server cenderung lebih aman, dengan alasan: semua perangkat lunak berada di satu lokasi, serta sistem operasi mainframe lebih aman daripada sistem operasi komputer mikro. Untuk alasan ini, sistem bernilai tinggi lebih mungkin ditempatkan di komputer mainframe, bahkan jika mainframe digunakan sebagai server dalam arsitektur klien-server. Di dunia yang

didominasi Internet saat ini, alat autentikasi dan enkripsi untuk arsitektur klien-server berbasis Internet lebih maju daripada yang digunakan pada arsitektur berbasis server mainframe. Virus adalah masalah potensial di semua arsitektur karena ia menyebar dengan mudah di komputer desktop. Model komputasi *zero client* berbasis server tidak menggunakan perangkat lunak pada perangkat klien, sehingga terlindungi dari *malware*.

**Persyaratan Budaya dan Politik.** Karena persyaratan budaya dan politik menjadi lebih penting, kemampuan untuk memisahkan logika penyajian dari logika aplikasi dan data menjadi penting. Pemisahan seperti itu mempermudah pengembangan logika penyajian dalam berbagai bahasa sekaligus menjaga logika aplikasi dan data tetap sama. Ini juga membuatnya lebih mudah untuk menyesuaikan logika penyajian bagi pengguna yang berbeda dan mengubahnya agar memenuhi norma budaya dengan lebih baik. Sejauh logika penyajian menyediakan akses ke aplikasi dan data, ini dapat mempermudah penerapan versi berbeda yang mengaktifkan atau menonaktifkan fitur berbeda yang diwajibkan oleh undang-undang dan peraturan di suatu negara. Pemisahan ini paling mudah dilakukan dalam arsitektur *thin client-server*, sehingga sistem dengan banyak persyaratan budaya dan politik sering menggunakan arsitektur *thin client-server*. Seperti halnya persyaratan integrasi sistem, dampak persyaratan hukum bergantung pada sifat spesifik persyaratan, tetapi secara umum, sistem berbasis klien cenderung kurang fleksibel.



### Latihan

---

Untuk memperdalam pemahaman Anda mengenai materi di atas, kerjakanlah latihan berikut!

- 1) Sebutkan dan jelaskan empat fungsi dasar pada sistem perangkat lunak sebagai bagian dari arsitektur sistem informasi!
- 2) Sebutkan dan jelaskan model-model arsitektur sistem informasi!
- 3) Jelaskan beberapa kelebihan model arsitektur sistem klien-server!
- 4) Sebutkan dan jelaskan dua model arsitektur sistem informasi modern saat ini!
- 5) Jelaskan empat kebutuhan nonfungsional utama dalam mendesain arsitektur sistem!

#### *Petunjuk Jawaban Latihan*

- 1) Empat fungsi dasar pada sistem perangkat lunak sebagai bagian dari arsitektur sistem informasi:
  - a) Fungsi sebagai penyimpanan data, yaitu logika yang digunakan untuk menyimpan dan atau mengambil kembali data dari entitas data.
  - b) Fungsi logika akses data, yaitu logika pemrosesan yang diperlukan untuk mengakses data.

- c) Fungsi logika aplikasi, seperti logika yang didokumentasi pada DFD dan kebutuhan fungsional.
  - d) Logika presentasi, yaitu logika untuk menampilkan informasi kepada pengguna dan antarmuka pengguna.
- 2) Model-model arsitektur sistem informasi:
- a) Arsitektur berbasis klien-server, terdiri atas berikut ini.
    - (1) Klien-server 2 tingkat: menggunakan 2 set komputer, masing-masing terdiri atas 1 set komputer klien (bertanggung jawab atas logika presentasi) dan 1 set komputer server (bertanggung jawab atas logika akses data dan logika penyimpanan data). Logika aplikasi dapat berada di sisi server atau di sisi klien.
    - (2) Klien-server 3 tingkat: menggunakan 3 set komputer, masing-masing terdiri atas 1 set komputer klien (bertanggung jawab atas logika presentasi) dan 2 set komputer server. Server pertama (server aplikasi) bertanggung jawab atas logika aplikasi dan server kedua (server basis data) bertanggung jawab atas logika akses data dan penyimpanan data.
    - (3) Klien-server n-tingkat (*n-tier*): terdiri atas 1 set komputer klien (bertanggung jawab atas logika presentasi) dan beberapa lapisan komputer server yang lebih terspesialisasi, misalnya: server aplikasi (menangani logika aplikasi), server Web (menangani logika bisnis terkait Web), dan server database (bertanggung jawab atas logika akses data dan penyimpanan data).
  - b) Arsitektur berbasis server: terdiri atas 1 komputer server (biasanya berupa komputer mainframe) dan beberapa terminal komputer (sebagai klien) yang berfungsi sebagai alat masukan dan pengirim data ke server untuk diproses, selanjutnya menerima instruksi dari server mengenai apa yang akan ditampilkan. Semua logika pemrosesan data berpusat di server.
  - c) Arsitektur berbasis klien: terdiri atas 1 set komputer klien (bertanggung jawab atas logika presentasi, logika aplikasi, dan logika akses data) terhubung (melalui jaringan area lokal) ke 1 set komputer server (yang berfungsi hanya untuk menyediakan penyimpanan data). Semua pemrosesan data dilakukan di sisi klien, dengan terlebih dahulu menerima kiriman data dari server.
- 3) Kelebihan model arsitektur sistem informasi berbasis klien-server sebagai berikut.
- a) Mudah dalam pengaturan beban kerja (penyimpanan dan atau pemrosesan data).
  - b) Dimungkinkan terjadinya koneksi multiplatform dalam jaringan (misalnya, koneksi komputer dengan sistem operasi yang berbeda), sehingga pengembang tidak terikat dalam 1 vendor.

- c) Mudah dalam pemisahan atau pengaturan fungsi-fungsi logika dasar perangkat lunak, agar lebih independen.
  - d) Jika terjadi suatu kegagalan kerja sistem, dengan mudah dapat dilokalisir dan dipulihkan.
- 4) Dua model arsitektur sistem informasi modern yang berbasis jaringan global:
- a) Model virtualisasi, yaitu membagi lingkungan server fisik menjadi beberapa lingkungan virtual, baik sumber daya penyimpanan maupun sumber daya jaringan (disebut server virtual), untuk mengatasi keterbatasan model server fisik yang berbiaya mahal. Server virtual dapat diatur sebagaimana layaknya mengatur server fisik.
  - b) Model *cloud computing*, yaitu sekumpulan perangkat keras, jaringan, penyimpanan, layanan aplikasi, proses bisnis, kolaborasi, dan antarmuka yang digabungkan untuk memberikan aspek komputasi sebagai layanan berbasis jaringan global (internet) dari pihak ketiga.
- 5) Empat jenis kebutuhan nonfungsional utama dalam mendesain arsitektur sistem:
- a) Kebutuhan operasional: berkaitan dengan lingkungan operasi di mana sistem harus beroperasi (misalnya sistem operasi, perangkat lunak sistem, dan sistem informasi lain yang berinteraksi dengan sistem)
  - b) Kebutuhan kinerja: berkaitan dengan masalah kinerja sistem (misalnya waktu respon, kapasitas, dan keandalan)
  - c) Kebutuhan keamanan: berkaitan dengan kemampuan untuk melindungi sistem informasi dari gangguan dan kehilangan data, baik berupa tindakan yang disengaja maupun karena kegagalan sistem (misalnya peretasan, kegagalan disk)
  - d) Persyaratan budaya dan politik: berkaitan dengan negara atau lingkungan tempat sistem akan digunakan (misalnya multi bahasa, format data, dan legalitas).



## Rangkuman

Sistem perangkat lunak dapat dibagi menjadi empat fungsi dasar: penyimpanan data, logika akses data (misalnya SQL), logika aplikasi (yang merupakan logika yang didokumentasikan di DFD dan kebutuhan fungsional), dan logika presentasi (antarmuka pengguna).

Ada tiga arsitektur dasar yang menempatkan fungsi dasar perangkat lunak pada perangkat komputer yang berbeda. Dalam arsitektur berbasis server, server melakukan semua fungsi. Dalam arsitektur berbasis klien, komputer klien bertanggung jawab atas logika presentasi, logika aplikasi, dan logika akses data, dengan data yang disimpan di server file. Dalam arsitektur klien-server, klien bertanggung jawab atas logika

presentasi dan server bertanggung jawab atas logika akses data dan penyimpanan data. Dalam arsitektur *thin client-server*, server menjalankan logika aplikasi, sedangkan dalam arsitektur *thick client-server*, logika aplikasi dibagi antara server dan klien. Dalam arsitektur klien-server dua tingkat, terdapat dua kelompok komputer: satu klien dan satu set server. Dalam arsitektur klien-server tiga tingkat, terdapat tiga kelompok komputer: klien, satu set server aplikasi, dan satu set server database.

Virtualisasi dan *cloud computing* adalah dua jenis perkembangan teknologi yang memunculkan opsi-opsi baru arsitektur sistem.

Pembuatan desain arsitektur merujuk pada kebutuhan nonfungsional. Kebutuhan operasional menentukan lingkungan operasi di mana sistem harus bekerja dan bagaimana hal tersebut dapat berubah dari waktu ke waktu (yaitu lingkungan teknis, integrasi sistem, portabilitas, dan pemeliharaan). Kebutuhan kinerja berfokus pada masalah kinerja seperti kecepatan sistem, kapasitas, serta ketersediaan dan keandalan. Kebutuhan keamanan berupaya melindungi sistem informasi dari gangguan dan kehilangan data (misalnya nilai sistem, kontrol akses, enkripsi dan autentikasi, dan pengendalian virus). Kebutuhan budaya dan politik adalah kebutuhan yang terkait dengan wilayah tempat sistem akan digunakan (misalnya multibahasa, kustomisasi, norma yang tidak tertulis, dan hukum).



### Tes Formatif 1

Pilihlah satu jawaban yang paling tepat!

- 1) Berikut adalah elemen utama desain arsitektur sistem informasi, *kecuali* ....
  - A. perangkat lunak
  - B. perangkat keras
  - C. manusia (pengguna sistem)
  - D. jaringan sistem
- 2) Jenis kebutuhan sistem yang paling memengaruhi proses desain arsitektur sistem adalah ....
  - A. kebutuhan fungsional
  - B. kebutuhan nonfungsional
  - C. kebutuhan bisnis
  - D. Semua jawaban benar
- 3) Komponen utama perangkat keras sistem informasi adalah ....
  - A. komputer klien, server, administrator sistem
  - B. server, jaringan komputer, administrator sistem
  - C. jaringan komputer, administrator sistem, komputer klien
  - D. komputer klien, server, jaringan komputer

- 4) Pada model arsitektur klien-server (dua tingkat), fungsi logika presentasi data/informasi merupakan tanggung jawab ....
  - A. server
  - B. komputer klien
  - C. berbagi antara server dan komputer klien
  - D. pengguna sistem
- 5) Model arsitektur klien-server 3 tingkat terdiri atas komponen ....
  - A. komputer klien - server aplikasi - server web
  - B. komputer klien - server web - server aplikasi
  - C. komputer klien - server aplikasi - server database
  - D. komputer klien - server aplikasi - terminal
- 6) Keterbatasan model arsitektur sistem klien-server adalah ....
  - A. rumit dalam penulisan kode program/perangkat lunak
  - B. rumit dalam pembaruan sistem secara keseluruhan
  - C. waktu proses relatif lambat
  - D. jawaban A dan B benar
- 7) Bertanggung jawab atas empat fungsi dasar perangkat lunak adalah ciri model arsitektur ....
  - A. berbasis server
  - B. berbasis klien
  - C. berbasis klien-server
  - D. berbasis klien-server bertingkat
- 8) Model arsitektur sistem di mana fungsi logika presentasi, logika aplikasi, dan logika akses data menjadi tanggung jawab aplikasi pada komputer klien, sedangkan fungsi penyimpanan menjadi tanggungjawab server, merupakan ....
  - A. model arsitektur berbasis klien
  - B. model arsitektur berbasis server
  - C. model arsitektur berbasis klien-server
  - D. model arsitektur berbasis klien-server bertingkat
- 9) *Cloud computing* sebagai model arsitektur sistem informasi modern dapat diimplementasikan dalam beberapa cara, *kecuali* ....
  - A. *cloud* publik
  - B. *cloud* pribadi
  - C. *cloud* organisasi
  - D. *cloud* hibrid

- 10) Jenis kebutuhan operasional (tergolong dalam kebutuhan nonfungsional) yang tidak memerlukan pilihan arsitektur tertentu adalah ....
- A. kebutuhan integrasi sistem
  - B. kebutuhan portabilitas
  - C. kebutuhan lingkungan teknis
  - D. kebutuhan lingkungan pemeliharaan

Cocokkanlah jawaban Anda dengan Kunci Jawaban Tes Formatif 1 yang terdapat di bagian akhir modul ini. Hitunglah jawaban yang benar. Kemudian, gunakan rumus berikut untuk mengetahui tingkat penguasaan Anda terhadap materi Kegiatan Belajar 1.

$$\text{Tingkat Penguasaan} = \frac{\text{Jumlah Jawaban yang Benar}}{\text{Jumlah Soal}} \times 100$$

Arti tingkat penguasaan

<70%

70% - 79%

80% - 89%

90% - 100%

kurang

cukup

baik

baik sekali

Apabila mencapai tingkat penguasaan 80% atau lebih, Anda dapat meneruskan dengan Kegiatan Belajar 2. **Bagus!** Jika masih di bawah 80%, Anda harus mengulangi materi Kegiatan Belajar 1, terutama bagian yang belum dikuasai.

## Desain Antarmuka Pengguna

**A**ntarmuka pengguna (*user interface*) adalah bagian dari sistem tempat pengguna berinteraksi. Ini mencakup tampilan layar yang menyediakan navigasi melalui sistem, layar, dan formulir yang menangkap data, dan laporan yang dihasilkan sistem (baik di atas kertas, di Web, atau melalui beberapa media lain).

Desain/perancangan antarmuka adalah proses mendefinisikan bagaimana sistem akan berinteraksi dengan entitas eksternal (misalnya, pelanggan, pemasok, sistem lain). Antarmuka pengguna mencakup tiga bagian dasar. **Pertama** adalah mekanisme navigasi, cara pengguna memberikan instruksi ke sistem dan memberitahukan apa yang harus dilakukan (misalnya tombol, menu). **Kedua** adalah mekanisme masukan (*input*), cara sistem menangkap informasi (misalnya, formulir untuk menambah pelanggan baru). **Ketiga** adalah mekanisme keluaran, cara sistem memberikan informasi kepada pengguna atau sistem lain (misalnya laporan, halaman Web). Masing-masing secara konseptual berbeda, tetapi semuanya saling terkait erat: semua layar komputer berisi mekanisme navigasi, dan sebagian besar berisi mekanisme masukan dan keluaran. Oleh karena itu, rancangan navigasi, rancangan masukan, dan rancangan keluaran merupakan kesatuan yang saling terkait.

Bidang ilmu Interaksi Manusia dengan Komputer (*Human Computer Interaction/HCI*) berfokus pada peningkatan interaksi antara pengguna dan komputer dengan membuat komputer lebih berguna dan menerima kebutuhan pengguna. Beberapa organisasi mempekerjakan desainer HCI profesional, yang memiliki spesialisasi dalam menerapkan proses desain untuk pembuatan grafis antarmuka pengguna dan antarmuka Web.

Kegiatan belajar ini memperkenalkan beberapa prinsip desain antarmuka pengguna yang mendasar dan memberikan gambaran umum tentang proses mendesain antarmuka pengguna. Selanjutnya, memberikan gambaran umum tentang komponen navigasi, komponen masukan, dan komponen keluaran yang digunakan dalam desain antarmuka.

### A. PRINSIP-PRINSIP DESAIN ANTARMUKA PENGGUNA

Dalam banyak hal, desain/rancangan antarmuka pengguna adalah seni. Tujuannya adalah untuk membuat antarmuka menyenangkan bagi mata dan mudah

digunakan, sekaligus meminimalkan upaya yang dikeluarkan pengguna untuk menyelesaikan pekerjaan mereka.

Beberapa hasil temuan mengungkapkan bahwa masalah terbesar yang dihadapi desainer berpengalaman adalah menggunakan ruang secara efektif (Dennis, 2012). Sederhananya, ada lebih banyak informasi untuk disajikan daripada ruang yang tersedia untuk menyajikannya. Analis harus menyeimbangkan kebutuhan akan kesederhanaan dan tampilan yang menyenangkan dengan kebutuhan untuk menyajikan informasi di berbagai halaman atau layar, yang dapat mengurangi kesederhanaan. Prinsip dasar desain antarmuka yang umum untuk desain navigasi, desain masukan (input), dan desain keluaran (output) akan dibahas berikut ini.

### 1. Tata Letak

Prinsip pertama desain antarmuka pengguna berkaitan dengan tata letak layar, formulir, atau laporan. Tata letak mengacu pada pengaturan area layar atau dokumen untuk tujuan yang berbeda dan menggunakan area tersebut secara konsisten di seluruh antarmuka pengguna. Kebanyakan perangkat lunak yang dirancang untuk komputer pribadi mengikuti pendekatan standar Windows atau Macintosh untuk tata letak layar (Dennis, 2012). Pendekatan ini membagi layar menjadi tiga area utama: area atas menginformasikan pengguna cara untuk menavigasi melalui sistem; area tengah (dan terbesar) untuk memajang karya pengguna; dan bagian bawah berisi informasi status tentang apa yang dilakukan pengguna.

Dalam banyak kasus antarmuka pengguna untuk sistem berbasis Web, digunakan beberapa area tata letak. Gambar 9.11 menunjukkan layar dengan tiga area navigasi utama, yang masing-masing diatur untuk menyediakan fungsi dan navigasi yang berbeda dalam bagian sistem yang berbeda (<http://www.gurupintar.ut.ac.id/content/prodi-fkip-ut>). Area atas menyediakan navigasi browser web standar (navigasi sistem) dan kontrol perintah yang mengubah konten seluruh sistem. Area navigasi di tepi kiri (navigasi situs) bermanuver antar bagian dan mengubah semua konten di sebelah kanannya (biasanya terdiri atas beberapa area navigasi). Konten di tengah halaman menampilkan hasil (yaitu ulasan artikel) dan menyediakan navigasi tambahan di dalam halaman tentang ulasan ini (navigasi halaman).



Gambar 9.11  
Prinsip-Prinsip Desain Antarmuka Pengguna

Pengaturan tata letak area untuk navigasi juga berlaku untuk masukan (input) dan keluaran (output). Area data pada laporan dan formulir sering kali dibagi lagi menjadi subarea, yang masing-masing digunakan untuk jenis informasi yang berbeda. Area-area ini hampir selalu berbentuk persegi panjang, meskipun terkadang keterbatasan ruang menyebabkan bentuk yang aneh. Setiap area dalam laporan atau formulir dirancang untuk menampung informasi yang berbeda. Misalnya, pada formulir pemesanan (atau laporan pesanan), satu bagian dapat digunakan untuk informasi pelanggan (misalnya: nama, alamat), satu bagian untuk informasi tentang pesanan secara umum (misalnya: tanggal, informasi pembayaran), dan satu bagian untuk detail pesanan (misalnya: berapa unit item beserta harga masing-masing). Setiap area berdiri sendiri sehingga informasi di satu area tidak berpindah ke area lain.

Area dan informasi di dalam area harus memiliki aliran intuitif secara alami untuk meminimalkan pergerakan pengguna dari satu area ke area berikutnya. Orang-orang di negara-negara barat (misalnya Eropa, Amerika Utara) cenderung membaca dari atas ke bawah, dari kiri ke kanan, sehingga informasi terkait harus ditempatkan sedemikian rupa sehingga digunakan dalam urutan ini (misalnya: baris alamat, diikuti oleh kota, negara bagian/provinsi, dan kemudian kode pos). Kadang-kadang, urutannya dalam urutan kronologis, atau dari yang umum ke yang spesifik, atau dari yang paling sering ke yang paling jarang digunakan. Bagaimanapun, sebelum area ditempatkan pada formulir atau laporan, analis harus memiliki pemahaman yang jelas tentang pengaturan yang paling logis terkait bagaimana sebuah formulir atau laporan akan digunakan.

**Kalkulator Breakeven Point**

Informasi Produk

Nama Produk	<input type="text"/>	Harga Satuan	<input type="text"/>
Biaya Tetap	<input type="text"/>	Biaya Satuan	<input type="text"/>
Breakeven Point	<input type="text"/>	Total Pendapatan	<input type="text"/>
		Total Biaya	<input type="text"/>

**Aliran Antarmuka Antar Bagian (Horizontal)**

Hitung      Baru      Keluar

Gambar 9.12  
Aliran Antarmuka Antar Bagian (Horizontal)

**Kalkulator Breakeven Point**

Informasi Produk

Nama Produk	<input type="text"/>
Harga Satuan	<input type="text"/>
Biaya Tetap	<input type="text"/>
Biaya Satuan	<input type="text"/>

Breakeven Point

<input type="text"/>
<input type="text"/>
<input type="text"/>

Total Pendapatan

Total Biaya

Hitung      Baru      Keluar

Gambar 9.13  
Aliran Antarmuka Antar Bagian (Vertikal)

Aliran antar bagian juga harus konsisten, baik horizontal (Gambar 9.12) maupun vertikal (Gambar 9.13). Idealnya, area akan tetap konsisten dalam ukuran, bentuk, dan penempatan formulir yang digunakan untuk memasukkan informasi dan menyajikan laporan.

## 2. Penyadaran Konten

Penyadaran konten (*content awareness*) mengacu pada kemampuan sebuah antarmuka untuk membuat pengguna sadar akan informasi yang dikandungnya dengan sedikit usaha oleh pengguna (Dennis, 2012). Semua bagian antarmuka, baik navigasi, masukan, atau keluaran, harus memberikan *content awareness* sebanyak mungkin, namun yang paling memerlukan adalah formulir atau laporan yang digunakan dengan cepat atau tidak teratur (misalnya situs Web).

**Content awareness berlaku untuk antarmuka secara umum.** Semua antarmuka harus memiliki judul (misalnya pada bingkai layar). Menu harus menunjukkan di mana pengguna berada. Misalnya, pada Gambar 9.11, baris status paling bawah (bagian kiri) menunjukkan bahwa pengguna sedang berada di bagian *Informasi GPO* dari situs *gurupintar.ut.ac.id*.

**Content awareness berlaku untuk area di dalam formulir dan laporan.** Semua area harus jelas dan ditentukan dengan baik (dengan judul jika ruang memungkinkan) untuk mengurangi kemungkinan pengguna menjadi bingung tentang informasi di area manapun. Kemudian pengguna bisa dengan cepat menemukan bagian dari formulir atau laporan yang mungkin berisi informasi yang mereka butuhkan. Kadang-kadang area ditandai dengan garis, warna, atau tajuk; dalam kasus lain, area hanya tersirat (misalnya tautan di tengah halaman).

**Content awareness berlaku untuk bidang di setiap area.** *Field* adalah elemen data individu yang merupakan masukan atau keluaran. Label bidang yang mengidentifikasi bidang pada antarmuka harus pendek dan spesifik. Tidak boleh ada ketidakpastian tentang format informasi dalam bidang, baik untuk entri atau tampilan. Misalnya tanggal 10/5/12 memiliki arti berbeda, bergantung pada apakah seseorang berada di Amerika Serikat (5 Oktober 2012) atau di Kanada (10 Mei 2012). Setiap bidang yang memiliki kemungkinan ketidakpastian atau interpretasi ganda harus memberikan penjelasan secara eksplisit.

**Content awareness juga berlaku untuk informasi yang berisi formulir atau laporan.** Secara umum, semua formulir dan laporan harus berisi tanggal persiapan (yaitu tanggal dicetak atau tanggal data dilengkapi) sehingga usia informasi jelas. Demikian pula, semua formulir dan perangkat lunak tercetak harus memberikan nomor versi sehingga pengguna, analis, dan pemrogram dapat mengidentifikasi materi yang ketinggalan zaman.

### 3. Estetika

Estetika mengacu pada desain antarmuka yang enak dipandang. Antarmuka tidak harus berupa karya seni, tetapi harus fungsional dan menarik untuk digunakan. Dalam kebanyakan kasus, “*less is more*,” artinya desain minimalis dan sederhana adalah yang terbaik.

Ruang biasanya menjadi sangat berarti untuk formulir dan laporan, dan sering kali ada kecenderungan untuk menyajikan informasi sebanyak mungkin ke halaman atau layar. Sayangnya, hal ini dapat membuat formulir atau laporan menjadi tidak menyenangkan sehingga pengguna tidak ingin mengisinya. Secara umum, semua formulir dan laporan membutuhkan setidaknya jumlah minimum spasi kosong yang sengaja dikosongkan.

Sebuah formulir yang biasanya tidak menyenangkan adalah yang memiliki kepadatan terlalu tinggi; memiliki terlalu banyak informasi yang dikemas ke dalam ruang yang terlalu kecil dengan ruang kosong yang terlalu sedikit. Meskipun mungkin

efisien dalam menghemat kertas menjadi satu halaman, bukan dua, ini tidak efektif untuk kebanyakan pengguna.

Secara umum, pengguna baru atau pengguna yang jarang berinteraksi dengan antarmuka, baik di layar atau di atas kertas, lebih memilih antarmuka dengan kepadatan rendah, sering kali yang memiliki kepadatan kurang dari 50% (yaitu kurang dari 50% antarmuka diisi oleh informasi). Pengguna yang lebih berpengalaman lebih menyukai kepadatan yang lebih tinggi, terkadang mendekati 90% terisi, karena mereka tahu di mana informasi berada dan kepadatan tinggi mengurangi jumlah pergerakan fisik melalui antarmuka (Dennis, 2012).

Desain teks sama pentingnya dengan desain formulir. Secara umum, harus ada satu jenis *font* untuk seluruh formulir atau laporan dan tidak lebih dari dua ukuran *font* tersebut di formulir atau laporan. Ukuran *font* yang lebih besar dapat digunakan untuk judul, judul bagian, dan lain-lain, dan *font* yang lebih kecil untuk laporan atau konten formulir. Jika formulir atau laporan akan dicetak, *font* yang lebih kecil harus berukuran (*font size*) minimal 8. *Font size* minimal 10 lebih disukai jika pengguna adalah orang tua. Untuk formulir atau laporan yang ditampilkan di layar, pertimbangkan minimal *font size* 12 jika tampilan monitor diatur untuk resolusi layar tinggi. Huruf miring dan garis bawah harus dihindari karena membuat teks lebih sulit untuk dibaca.

*Font* seperti Times New Roman adalah yang paling mudah dibaca untuk laporan tercetak, terutama untuk huruf kecil. *Font* seperti Tahoma atau Arial adalah yang paling mudah dibaca untuk layar komputer dan sering digunakan sebagai *heading* dalam laporan tercetak. Jangan pernah menggunakan huruf kapital semua, kecuali mungkin untuk judul.

Warna dan pola harus digunakan dengan hati-hati dan hemat dan hanya jika memiliki tujuan (terdapat banyak pengguna yang buta warna, jadi penggunaan warna yang tidak tepat dapat mengganggu kemampuan mereka untuk membaca informasi). Ingat, tujuannya adalah kemudahan membaca, bukan seni; warna dan pola harus digunakan untuk memperkuat pesan, bukan membanjirinya. Warna paling baik digunakan untuk memisahkan dan mengkategorikan item, seperti menunjukkan perbedaan antara judul dan teks biasa, atau untuk menyoroti informasi penting. Oleh karena itu, warna dengan kontras tinggi harus digunakan (misalnya hitam dan putih). Secara umum, teks hitam dengan latar belakang putih adalah yang paling mudah dibaca, sedangkan warna biru di atas merah paling tidak dapat dibaca (kebanyakan ahli setuju bahwa pola latar belakang pada halaman Web harus dihindari). Warna telah terbukti mempengaruhi emosi, dengan warna merah yang memicu emosi yang intens (misalnya marah) dan biru yang memicu emosi yang diturunkan (misalnya mengantuk).

#### 4. Pengalaman Pengguna

Pengalaman pengguna (*user experience*) mengacu pada merancang antarmuka pengguna dengan mempertimbangkan tingkat pengalaman pengguna menggunakan komputer. Sistem komputer akan digunakan oleh orang-orang yang berpengalaman dan

oleh orang-orang yang tidak berpengalaman; antarmuka pengguna harus dirancang untuk kedua jenis pengguna ini. Pengguna pemula biasanya sangat mementingkan kemudahan belajar (seberapa cepat dan mudah mereka dapat belajar menggunakan sistem). Pengguna ahli biasanya lebih mementingkan kemudahan penggunaan (seberapa cepat dan mudah mereka dapat menyelesaikan tugas dengan sistem setelah mereka mempelajari cara menggunakannya). Seringkali, kedua tujuan ini saling melengkapi dan mengarah pada keputusan desain yang serupa, tetapi terkadang ada kompromi. Para pemula, misalnya, sering kali lebih menyukai menu yang menampilkan semua fungsi sistem yang tersedia, karena ini menawarkan kemudahan belajar. Para ahli, di sisi lain, terkadang lebih memilih menu yang lebih sedikit yang diatur berdasarkan fungsi yang paling umum digunakan.

Meskipun antarmuka harus mencoba menyeimbangkan antara kemudahan penggunaan dan kemudahan belajar, jenis sistem ini harus lebih menekankan pada kemudahan penggunaan daripada kemudahan belajar. Pengguna harus dapat mengakses fungsi yang umum digunakan dengan cepat, dengan sedikit penekanan tombol atau sedikit pilihan menu.

## 5. Konsistensi

Konsistensi dalam desain mungkin merupakan faktor terpenting dalam membuat sistem mudah digunakan, karena memungkinkan pengguna untuk memprediksi apa yang akan terjadi. Jika antarmuka konsisten, pengguna dapat berinteraksi dengan satu bagian sistem dan kemudian mengetahui cara berinteraksi dengan bagian lainnya (selain elemen unik yang terdapat pada bagian tersebut). Konsistensi biasanya mengacu pada antarmuka dalam satu sistem komputer, sehingga semua bagian sistem yang sama bekerja dengan cara yang sama. Idealnya, bagaimanapun, sistem tersebut juga harus konsisten dengan sistem komputer lain dalam organisasi dan dengan perangkat lunak komersial apa pun yang digunakan (misalnya Windows). Misalkan banyak pengguna yang akrab dengan Web, sehingga penggunaan antarmuka mirip Web dapat mengurangi jumlah pembelajaran yang dibutuhkan oleh pengguna. Dengan cara ini, pengguna dapat menggunakan kembali pengetahuan Web, sehingga secara signifikan mengurangi waktu pembelajaran untuk sistem baru.

Konsistensi terjadi di berbagai tingkatan. Konsistensi dalam kontrol navigasi menunjukkan bagaimana tindakan dalam sistem harus dilakukan. Misalnya, menggunakan ikon atau perintah yang sama untuk mengubah item dengan jelas mengkomunikasikan bagaimana perubahan dilakukan di seluruh sistem. Konsistensi dalam terminologi juga penting. Ini mengacu pada penggunaan kata yang sama untuk elemen pada formulir dan laporan (misalnya tidak menggunakan istilah "pelanggan" di satu tempat dan "klien" di tempat lain). Konsistensi dalam laporan dan desain formulir juga penting, meskipun satu studi menunjukkan bahwa terlalu konsisten dapat menyebabkan masalah. Ketika laporan dan formulir sangat mirip (kecuali perubahan kecil pada judul), pengguna terkadang menggunakan laporan atau formulir yang salah

dan memasukkan data yang salah atau salah menafsirkan informasinya. Implikasi bagi desain adalah membuat laporan dan formulir serupa, tetapi memberikan beberapa elemen khusus (misalnya warna, ukuran judul) yang memungkinkan pengguna untuk segera mendeteksi perbedaan.

## 6. Meminimalkan Upaya Pengguna

Pada akhirnya, antarmuka harus dirancang untuk meminimalkan jumlah upaya yang diperlukan pengguna untuk menyelesaikan tugas. Ini berarti menggunakan sesedikit mungkin *klik mouse* atau *penekanan tombol* untuk berpindah dari satu bagian sistem ke bagian lain. Sebagian besar perancang antarmuka mengikuti prinsip *tiga kali klik*, yaitu pengguna harus dapat beralih dari menu awal atau utama sistem ke informasi atau tindakan yang mereka inginkan tidak lebih dari tiga kali klik mouse atau tiga kali penekanan tombol.

## B. PROSES DESAIN ANTARMUKA PENGGUNA

Desain antarmuka pengguna dibangun berdasarkan informasi pada DFD yang dikembangkan dalam fase analisis (Pressman, 2002), dan mewawancara pengguna untuk **mengembangkan skenario penggunaan** yang menggambarkan pola tindakan yang umum dilakukan pengguna (Dennis, 2012).

Analis kemudian **mengembangkan struktur diagram antarmuka** (*interface structure diagram/ISD*) yang mendefinisikan struktur dasar antarmuka. Diagram ini menunjukkan semua antarmuka (misalnya layar, formulir, dan laporan) dalam sistem dan bagaimana mereka saling terhubung. Berdasarkan struktur diagram antarmuka, analis **mendesain standar dan template antarmuka dalam sistem**.

Tahap akhir, analis **membuat prototipe antarmuka** berdasarkan desain antarmuka, selanjutnya melakukan **evaluasi prototipe** untuk menentukan apakah desain antarmuka memuaskan dan bagaimana mereka dapat ditingkatkan. Evaluasi antarmuka selalu mengidentifikasi perbaikan, sehingga proses desain antarmuka diulangi dalam proses siklus sampai tidak ada perbaikan baru yang teridentifikasi. Dalam praktik yang baik, analis berinteraksi secara dekat dengan pengguna selama proses desain antarmuka, sehingga pengguna memiliki banyak kesempatan untuk melihat antarmuka saat dikembangkan, daripada menunggu tahap evaluasi antarmuka di akhir proses desain antarmuka untuk melakukan penyempurnaan. Akan lebih baik bagi semua pihak (baik analis maupun pengguna) untuk lebih cepat mengidentifikasi perubahan sebelum sebagian besar layar antarmuka standar dirancang.

### Contoh

Perhatikan Gambar 4.23 (diagram berjenjang) dan Gambar 4.24 (DFD level 0), serta Gambar 4.25 hingga Gambar 4.27 (DFD level 1) pada Modul 4 Kegiatan Belajar

1. Skenario pada diagram berjenjang dan DFD tersebut akan kita gunakan untuk mendesain antarmuka pengguna aplikasi Sistem Perpustakaan.

### 1. Menyusun Skenario Penggunaan

Skenario penggunaan adalah garis besar langkah-langkah yang dilakukan pengguna untuk menyelesaikan beberapa bagian dari pekerjaan mereka. DFD dapat mencakup berbagai cara untuk merespon penyelesaian sebuah peristiwa. Misalnya, skenario pada DFD level 0 (Gambar 4.24) aplikasi sistem perpustakaan dapat disusun sebagai berikut.

- a. Pada Proses 1 (Pendataan Awal), terdiri atas 2 kegiatan, yaitu pendataan koleksi pustaka dan registrasi anggota.
  - 1) Proses pendataan koleksi pustaka, sistem menerima masukan (input) berupa koleksi pustaka dari entitas luar petugas pengadaan (proses 1.1), memprosesnya dan menyimpan data (menghasilkan output) ke simpanan data D1 (tabel koleksi). Aliran data dari proses pendataan awal ke simpanan data D1 nampak 2 arah, menunjukkan bahwa proses 1 (pendataan awal) mengambil kembali (query) data dari simpanan data D1 untuk keperluan penyajian informasi koleksi pustaka ke entitas luar anggota (proses 3.1).
  - 2) Proses registrasi anggota, sistem menerima masukan (input) berupa identitas anggota perpustakaan dari entitas luar anggota (proses 1.2), memprosesnya dan menyimpan data. Pada saat yang sama, sistem juga memproses identitas anggota dan menghasilkan keluaran (output) berupa kartu anggota ke entitas luar anggota (luaran tercetak).
- b. Pada Proses 2 (Peminjaman Koleksi)  
Proses peminjaman terdiri atas 2 kegiatan utama, yaitu proses peminjaman dan proses pengembalian pustaka.
  - 1) Proses peminjaman, sistem menerima masukan (input) berupa koleksi pustaka dari simpanan data D1, identitas anggota dari simpanan data D2, serta data peminjaman (misalnya: tanggal peminjaman, tanggal harus dikembalikan) dari entitas luar petugas layanan (proses 2.1). Ketiga input tersebut merupakan komponen input untuk proses peminjaman. Proses peminjaman menghasilkan output berupa data peminjaman mengalir ke simpanan data D3.
  - 2) Pada proses pengembalian, sistem menerima masukan (input) berupa sebagian data peminjaman (id\_peminjaman) dari simpanan data D3 dengan aliran data 2 arah (pustaka yang dikembalikan adalah pustaka yang pernah dipinjam) dan data pengembalian (misalnya: tanggal pengembalian) dari entitas luar petugas layanan (proses 2.2). Proses pengembalian menghasilkan output berupa data pengembalian mengalir ke simpanan data D4.

c. Pada Proses 3 (Penyajian Informasi)

- 1) Proses penyajian informasi terdiri atas 2 kegiatan utama, yaitu proses penyajian informasi koleksi pustaka dan pelaporan aktivitas peminjaman koleksi pustaka.
- 2) Proses penyajian informasi koleksi pustaka, sistem menerima input berupa data koleksi pustaka dari simpanan data D1, memprosesnya dan menghasilkan luaran (output) berupa informasi koleksi pustaka ke entitas luar anggota dan kepala perpustakaan (proses 3.1).
- 3) Pada proses pelaporan aktivitas peminjaman koleksi pustaka, sistem menerima input berupa data peminjaman dari simpanan data D3 dan data pengembalian (id\_pengembalian) dari simpanan data D4, memprosesnya dan menghasilkan luaran (output) berupa laporan peminjaman koleksi pustaka ke entitas luar kepala perpustakaan (proses 3.2).

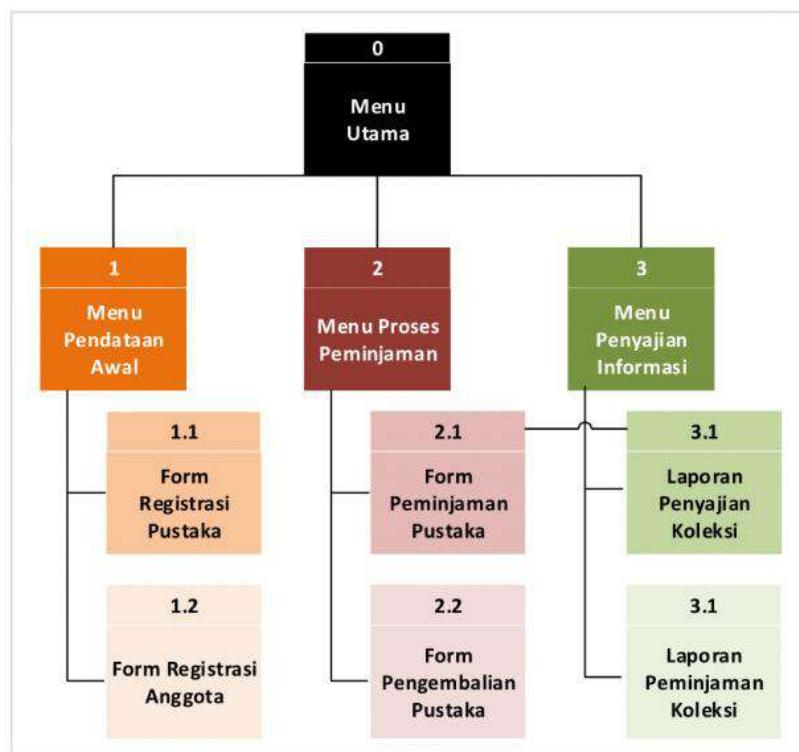
DFD dirancang untuk memodelkan semua jalur yang memungkinkan. Tapi skenario penggunaan hanyalah dipilih satu jalur melalui kasus penggunaan yang paling umum atau yang paling sering terjadi (Sommerville, 2003) dan yang paling singkat dan sederhana. Skenario banyak jalur ini umumnya terjadi pada sistem-sistem berbasis Web, di mana jalur tersebut juga sering dilalui secara bolak-balik. Misalnya: dalam satu skenario penggunaan *sistem pembelian lagu secara online*/berbasis Web (Dennis, 2012), pengguna akan menelusuri banyak lagu, seperti seseorang yang menjelajahi toko musik nyata untuk mencari lagu yang menarik. Dia akan mencari lagu, mendengarkan sampel, mungkin menambahkannya ke keranjang belanja, mencari lagu yang lain, dan seterusnya. Akhirnya, pengguna akan ingin membeli download, mungkin menghapus beberapa pilihan dari keranjang belanja sebelumnya.

Dalam *skenario penggunaan* lain, pengguna hanya ingin membeli satu lagu tertentu. Dia akan langsung memilih lagu tersebut, memberi harga, dan segera membelinya, seperti seseorang yang berlari ke toko, langsung menuju ke barang yang dia inginkan, dan segera membayar dan meninggalkan toko. Untuk skenario penggunaan ini, kita perlu memastikan bahwa jalur melalui DFD yang disajikan oleh antarmuka adalah singkat dan sederhana, dengan sedikit menu dan klik mouse.

## 2. Mendesain Struktur Antarmuka

Desain struktur antarmuka mendefinisikan komponen dasar antarmuka dan bagaimana mereka saling berkaitan untuk menyediakan fungsionalitas bagi pengguna. Diagram Desain Struktur Antarmuka (*Interface Structure Design/ISD*) digunakan untuk menunjukkan bagaimana semua layar, formulir, dan laporan yang digunakan oleh sistem saling terkait dan bagaimana pengguna berpindah dari satu antarmuka ke yang lainnya (Gambar 9.14). Tidak seperti DFD, tidak ada aturan atau standar yang umum digunakan untuk pengembangan ISD.

Untuk mengetahui jenis antarmuka yang mewakili setiap proses pada DFD atau diagram berjenjang, analis harus memeriksa *skenario penggunaan*. Misalnya, pada Proses 1 bagian (1) (pendataan awal) terdapat pernyataan "sistem menerima **masukan (input)** berupa *koleksi pustaka* dari entitas luar *petugas pengadaan*", mengindikasikan munculnya sebuah **antarmuka masukan** (elemen ISD berupa formulir masukan: proses 1.1, 1.2, 2.1, dan 2.2 pada Gambar 9.14). Pada Proses 3 bagian (1) (Penyajian Informasi) terdapat pernyataan "... menghasilkan **luaran (output)** berupa *informasi koleksi pustaka* ke entitas luar *anggota ...*", mengindikasikan munculnya sebuah **antarmuka keluaran** (elemen ISD berupa laporan atau informasi: proses 3.1 dan 3.2 pada Gambar 9.14). Pernyataan pada sebuah proses DFD yang tidak mengindikasikan adanya **masukan** atau **keluaran**, kemungkinan besar berupa *antarmuka layar* (elemen ISD berupa layar: proses 0, 1, 2, dan 3 pada Gambar 9.14).



Gambar 9.14  
Desain Struktur Antarmuka Aplikasi Sistem Perpustakaan

Struktur dasar antarmuka mengikuti struktur dasar dari proses bisnis itu sendiri sebagaimana didefinisikan dalam model proses. Setiap kotak di ISD menunjukkan (di bagian atas) proses DFD yang didukung oleh antarmuka (Gambar 9.14). Dalam kebanyakan kasus, antarmuka membentuk hierarki atau pohon; tapi terkadang, sebuah antarmuka ditautkan ke salah satu di luar hierarki (Dennis, 2012). Seperti yang

ditunjukkan oleh tautan dari formulir peminjaman pustaka ke laporan penyajian koleksi (misalnya pengguna menampilkan informasi koleksi pada saat mengisi peminjaman, seperti judul koleksi, penulis, dan informasi lainnya yang berkaitan dengan suatu koleksi). Tautan antarmuka di luar jalur hierarki umumnya terjadi pada sistem-sistem aplikasi berbasis Web (ditunjukkan dengan *link* atau *page navigation*, lihat Gambar 9.11).

Seringkali, skenario penggunaan mengidentifikasi jalur melalui ISD yang lebih rumit dari yang seharusnya. Analis kemudian mengerjakan ulang ISD untuk menyederhanakan kemampuan antarmuka untuk mendukung skenario penggunaan, terkadang dengan membuat perubahan besar pada struktur menu, atau dengan menambahkan pintasan.

### 3. Desain Standar Antarmuka

Contoh desain standar antarmuka disajikan pada Gambar 9.15.

Label/nama antarmuka: Formulir Registrasi Anggota Baru

#### **Antarmuka Objek**

- Nomor Anggota: Kode unik yang dimiliki setiap anggota baru saat registrasi sebagai anggota, dibangkitkan secara otomatis oleh sistem.
- Nomor Identitas & Jenisnya: Nomor Kartu Identitas Diri, dapat berupa Kartu Tanda Penduduk, Kartu Surat Izin Mengemudi, dan sebagainya. Jenis identitas dapat diprogram dan dipilih melalui Kotak Kombo.
- Nama Lengkap: Nama Lengkap Anggota (termasuk gelar) sesuai yang tertera pada Kartu Identitas Diri.
- Alamat: Alamat domisili anggota, sesuai yang tertera pada Kartu Identitas Diri.
- Status Anggota: Status sosial anggota, misalnya: Pelajar, Mahasiswa, Pegawai Negeri/Swasta, Umum.

#### **Antarmuka Aksi**

- Baru: Mengosongkan isian teks, untuk mengisi data baru
- Simpan: Menyimpan data baru atau data hasil perubahan
- Edit: Mengaktifkan kotak isian teks tertentu yang berisi data sebelum melakukan perubahan data
- Hapus: Menghapus record data yang sedang tampil di form
- Selesai: Mengakhiri proses (menutup form)
- Cetak Kartu Anggota: Dapat mencetak kartu anggota setelah mengisi data anggota secara lengkap. Hasil cetakan dapat berupa tampilan di layar atau berupa cetak melalui alat cetak (printer).

#### **Ikon Antarmuka**

- Logo Organisasi: Ditampilkan di semua layar.

Gambar 9.15

Contoh Standar Antarmuka

(Formulir Registrasi Anggota Baru Aplikasi Sistem Perpustakaan)

Desain standar antarmuka mengidentifikasi objek dan tindakan yang berkenaan dengannya. Keberadaan objek-objek ini diidentifikasi melalui diskusi dengan pengguna sistem (pada tahapan elisitasi kebutuhan), atau mengamati DFD (kamus data) dan ERD (atribut-atribut simpanan data).

#### 4. Desain Template Antarmuka

Contoh desain template antarmuka disajikan pada Gambar 9.16. Tata letak objek dalam template antarmuka (baik untuk aplikasi berbasis desktop maupun aplikasi berbasis Web) disusun mengacu pada prinsip-prinsip desain antarmuka pengguna (dibahas pada Modul 9, Kegiatan Belajar 3, bagian A).

Gambar 9.16  
Contoh Desain Template Antarmuka Pengguna  
(Formulir Registrasi Anggota Baru Aplikasi Sistem Perpustakaan)

Pada saat mendesain template antarmuka, analis dapat bertemu dengan sponsor proyek dan pengguna sistem, untuk membahas desain yang sedang dikembangkan. Membuat perubahan pada tahap ini akan jauh lebih sederhana daripada menunggu sampai setelah mengembangkan prototipe. Pengguna mungkin punya beberapa saran, sehingga, setelah pertemuan tersebut, analis melakukan perubahan dan selanjutnya berpindah ke tahap pembuatan desain prototipe.

#### 5. Desain Prototipe

Prototipe adalah model kerja yang belum sempurna dari produk sistem informasi, biasanya dibangun untuk keperluan demonstrasi atau sebagai bagian dari proses pengembangan sistem. Prototipe memberikan informasi lebih rinci dan dapat dikembangkan dengan cepat menggunakan *tools* pengembangan prototipe, misalnya aplikasi *Balsamiq Mockup* (Sulianta, 2017) untuk menguji antarmuka secara interaktif.

Setelah analis memiliki prototipe awal yang dirancang, analis dapat melakukan ujicoba secara *offline* kepada user, atau mempostingnya di intranet dan meminta komentar dari beberapa pengguna yang berpengalaman menggunakan aplikasi berbasis Web. Analis kemudian melakukan revisi prototipe sesuai dengan komentar yang diterima dari pengguna, hingga diperoleh sebuah prototipe desain antarmuka yang dinyatakan sempurna.

## C. DESAIN NAVIGASI

Komponen navigasi antarmuka memungkinkan pengguna memasukkan perintah untuk menavigasi sistem dan melakukan tindakan untuk memasukkan dan melihat informasi yang terkandung di dalamnya. Komponen navigasi juga menyajikan pesan kepada pengguna tentang keberhasilan atau kegagalan perintah yang diberikan. Tujuan dari sistem navigasi adalah membuat sistem dapat dioperasikan sesederhana mungkin.

### 1. Prinsip Dasar

Dalam mendesain navigasi, analis biasanya harus berasumsi bahwa pengguna belum membaca manual dan belum mengikuti pelatihan. Semua kontrol harus didesain jelas, dapat dimengerti, dan ditempatkan di lokasi yang intuitif di layar. Idealnya, kontrol harus mengantisipasi apa yang akan dilakukan pengguna dan menyederhanakan usahanya. Misalnya, banyak program *set-up* yang dirancang untuk pengguna pemula, di mana pengguna cukup menekan tombol "next" terus menerus.

**Prinsip pertama: mencegah pengguna melakukan kesalahan.** Kesalahan dapat dikurangi dengan memberi label perintah dan dengan membatasi pilihan. Terlalu banyak pilihan dapat membingungkan pengguna, terutama jika mereka serupa dan sulit untuk dijelaskan dalam ruang yang singkat di layar. Ketika ada banyak pilihan serupa pada menu, perlu dipertimbangkan untuk membuat menu tingkat kedua. Saat pengguna akan menjalankan fungsi penting yang sulit atau tidak mungkin dibatalkan (misalnya menghapus file), penting untuk mengkonfirmasi tindakan tersebut dengan pengguna (dan memastikan pemilihan tidak dilakukan karena kesalahan). Ini biasanya dilakukan dengan meminta pengguna menanggapi pesan konfirmasi yang menjelaskan apa yang diminta pengguna dan meminta pengguna untuk memastikan bahwa tindakan ini benar.

**Prinsip kedua: menyederhanakan pemulihannya dari kesalahan.** Apapun yang dilakukan perancang sistem, pengguna tidak akan luput dari membuat kesalahan. Sistem harus membuatnya semudah mungkin untuk memperbaiki kesalahan ini. Idealnya, sistem akan memiliki tombol "urungkan" yang membuat kesalahan mudah ditimpak; namun, menulis perangkat lunak untuk tombol semacam itu bisa jadi sangat rumit.

**Prinsip ketiga: menggunakan urutan tata bahasa yang konsisten.** Sebagian besar perintah mengharuskan pengguna untuk menentukan objek (misalnya file, simpan) dan tindakan yang akan dilakukan pada objek itu (misalnya salin, hapus). Antarmuka dapat meminta pengguna untuk pertama-tama memilih objek dan kemudian

tindakan (urutan *objek-tindakan*) atau pertama memilih tindakan dan kemudian objek (urutan *tindakan-objek*). Sebagian besar aplikasi Windows menggunakan urutan tata bahasa *objek-tindakan* (misalnya kegiatan menyalin teks yang terblok di aplikasi pengolah kata). Urutan tata bahasa harus konsisten di seluruh sistem, baik di tingkat elemen data maupun di tingkat menu secara keseluruhan.

## 2. Jenis Kontrol Navigasi

Ada tiga pendekatan dasar dalam mendefinisikan perintah bagi pengguna perangkat lunak: bahasa program, menu, dan manipulasi langsung.

**Bahasa program.** Dengan perintah bahasa, pengguna memasukkan perintah dalam bahasa khusus yang dikembangkan untuk sistem komputer (misalnya UNIX dan SQL). Perintah melalui bahasa program terkadang memberikan fleksibilitas yang lebih besar daripada pendekatan lainnya, karena pengguna dapat menggabungkan elemen bahasa dengan cara yang tidak ditentukan sebelumnya oleh pengembang. Namun, cara ini memberikan beban yang lebih besar pada pengguna karena pengguna harus mempelajari sintaks dan tipe perintah dibandingkan jika memilih dari daftar pilihan yang terbatas.

**Menu.** Jenis sistem navigasi yang paling umum saat ini adalah menu. Menu harus dirancang dengan hati-hati, karena submenu di belakang menu utama disembunyikan dari pengguna hingga mereka mengklik item menu. Lebih baik membuat menu menjadi luas dan dangkal (yaitu, dengan setiap menu berisi banyak item dan setiap item hanya berisi satu atau dua lapisan menu) daripada sempit dan dalam (yaitu, setiap menu hanya berisi beberapa item, tetapi setiap item mengarah ke tiga atau lebih lapisan menu).

Jenis Menu	Penggunaan	Keterangan
Menu Bar	Menu utama untuk sistem	Item menu selalu satu kata. Jangan pernah mengizinkan pengguna untuk memilih tindakan yang tidak dapat mereka lakukan (sebagai gantinya, gunakan item berwarna abu-abu).
Menu Drop-down	Menu tingkat kedua, seringkali menjadi bagian/kelanjutan dari Menu Bar	Item menu dapat terdiri dari banyak kata, hindari singkatan.
Menu Hyperlink	Menu utama untuk sistem berbasis web	Sebaiknya ditempatkan di tepi kiri layar, meskipun dapat ditempatkan di tepi manapun. Item menu terdiri dari satu atau dua kata.
Embedded Hyperlink	Sebagai link ke informasi opsional	Biasanya membuka jendela baru yang ditutup setelah tindakan selesai,

Jenis Menu	Penggunaan	Keterangan
		sehingga pengguna dapat kembali ke skenario penggunaan awal.
Menu Pop-up	Sebagai jalan pintas ke suatu perintah tertentu (untuk pengguna berpengalaman)	Seringkali terabaikan oleh pengguna pemula, sehingga biasanya fungsi duplikat harus disediakan di menu lain.
Menu Tab	Ketika pengguna perlu mengubah beberapa pengaturan atau melakukan beberapa perintah terkait	Hindari lebih dari satu baris tab karena mengklik tab untuk membukanya dapat mengubah urutan tab.
Tool Bar	Sebagai jalan pintas ke perintah untuk pengguna berpengalaman	Harus berukuran sama. Jika ukuran label sangat bervariasi, gunakan dua ukuran berbeda (kecil dan besar). Tombol dengan ikon harus memiliki <i>tool tip</i> .
Image Map	Hanya ketika menambahkan gambar grafis ke menu	Gambar harus menyiratkan makna untuk menunjukkan apa yang terjadi saat diklik.

**Gambar 9.17**  
**Tipe Menu**

Penelitian menunjukkan bahwa dalam sistem yang ideal, satu menu harus berisi tidak lebih dari delapan item dan tidak lebih dari dua klik mouse atau penekanan tombol dari menu manapun untuk melakukan suatu tindakan (tidak termasuk dari menu utama untuk memulai sistem). Seringkali, item menu memiliki *hot key* yang memungkinkan pengguna berpengalaman untuk dengan cepat menjalankan perintah dengan penekanan tombol sebagai pengganti pilihan menu (misalnya "Control-F" di Word untuk perintah *Find*).

Menu harus mengumpulkan kategori item yang serupa sehingga pengguna dapat menebak isi setiap menu secara intuitif. Kebanyakan perancang merekomendasikan pengelompokan item menu berdasarkan objek antarmuka (misalnya, Pelanggan, Pesanan Pembelian, Inventaris) daripada berdasarkan aksi interaksi (misalnya, Baru, Edit, Format), sehingga semua aksi yang berkaitan dengan satu objek ada dalam satu menu, semua aksi untuk objek lain ada di menu berbeda, dan seterusnya. Gambar 9.17 menyajikan berbagai tipe menu dan penggunaannya.

**Manipulasi Langsung.** Dalam manipulasi langsung, pengguna memasukkan perintah dengan bekerja secara langsung dengan objek antarmuka. Misalnya, pengguna dapat mengubah ukuran objek di Microsoft PowerPoint dengan mengklik objek dan memindahkan sisi, atau pengguna dapat memindahkan file di Windows Explorer dengan menyeret nama file dari satu folder ke folder lain. Manipulasi langsung bisa

sederhana, tetapi ada dua masalah. Pertama, pengguna yang terbiasa dengan antarmuka berbasis bahasa atau menu tidak selalu mengharapkannya. Kedua, tidak semua perintah bersifat intuitif (misalnya, bagaimana Anda menyalin [bukan memindahkan] file di Windows Explorer?).

### 3. Pesan

Pesan adalah cara sistem merespons pengguna dan memberi tahu tentang status interaksi. Ada banyak jenis pesan yang berbeda, seperti pesan kesalahan, pesan konfirmasi, pesan pengakuan, pesan penundaan, dan pesan bantuan (Gambar 9.18). Secara umum, pesan harus jelas, ringkas, dan lengkap.

Pesan harus meminta pengguna untuk mengeksekusinya (misalnya, dengan mengklik), daripada ditampilkan selama beberapa detik dan kemudian menghilang (kecuali pesan yang memberi tahu pengguna tentang penundaan/menunggu dalam pemrosesan, yang akan hilang setelah penundaan berlalu).

Pesan (terutama pesan kesalahan) harus selalu menjelaskan masalah dalam istilah yang singkat (misalnya, apa yang dilakukan pengguna secara tidak benar) dan menjelaskan tindakan korektif sejelas dan setegas mungkin sehingga pengguna tahu persis apa yang perlu dilakukan. Pesan kesalahan harus memberikan nomor pesan. Nomor pesan tidak ditujukan untuk pengguna, tetapi akan mempermudah staf *help desk* dan jaringan dukungan pelanggan untuk mengidentifikasi masalah dan membantu pengguna, karena banyak pesan menggunakan kata-kata yang serupa.

Jenis Pesan	Penggunaan	Keterangan
Pesan Eror ( <i>Error Message</i> )	Ketika pengguna melakukan sesuatu yang tidak diizinkan atau tidak memungkinkan	Selalu jelaskan alasannya dan sarankan tindakan korektif. Biasanya, pesan kesalahan disertai dengan bunyi bip, tetapi banyak aplikasi sekarang menghilangkannya atau mengizinkan pengguna untuk menonaktifkan.
Pesan Konfirmasi ( <i>Confirmation Message</i> )	Saat pengguna memilih pilihan yang berpotensi berbahaya, seperti menghapus file	Selalu jelaskan penyebabnya dan sarankan tindakan yang mungkin dilakukan. Sering kali menyertakan beberapa pilihan selain "Oke" dan "Batal".
Pesan Pengakuan ( <i>Acknowledgment Message</i> )	Jarang atau tidak pernah; pengguna dengan cepat menjadi kesal dengan semua klik mouse yang tidak perlu	Pesan pengakuan biasanya disertakan karena pengguna pemula sering kali ingin diyakinkan bahwa suatu tindakan telah dilakukan.

Jenis Pesan	Penggunaan	Keterangan
Pesan Tunda <i>(Delay Message)</i>	Ketika suatu aktivitas membutuhkan waktu lebih dari tujuh detik	Pesan ini harus mengizinkan pengguna untuk membatalkan operasi jika dia tidak ingin menunggu penyelesaiannya. Pesan tersebut harus memberikan indikasi berapa lama penundaan bisa berlangsung.
Pesan Bantuan <i>(Help Message)</i>	Di semua sistem	Informasi bantuan diatur menurut daftar isi dan/atau pencarian kata kunci.

Gambar 9.18  
Tipe Pesan

#### D. DESAIN MASUKAN

Desain masukan (input) berarti mendesain layar yang digunakan untuk memasukkan informasi, serta formulir apapun yang digunakan pengguna untuk menulis atau mengetik informasi.

##### 1. Prinsip dasar

Prinsip-prinsip dasar untuk desain masukan adalah memperhatikan sifat masukan (baik *batch* atau *online*) dan cara untuk menyederhanakan pengumpulannya.

**Menggunakan Sistem Pemrosesan *Online* dan Pemrosesan *Batch* dengan Tepat.** Dengan pemrosesan *online* (kadang-kadang disebut pemrosesan transaksi), setiap item masukan dimasukkan pada saat yang sama dengan peristiwa atau transaksi yang meminta masukan. Pemrosesan *online* paling sering digunakan karena penting untuk memiliki informasi *real time* tentang proses bisnis. Misalnya, saat seseorang memesan kursi maskapai penerbangan, kursi tersebut tidak lagi tersedia untuk digunakan orang lain, sehingga informasi tersebut harus segera dicatat.

Dengan pemrosesan *batch*, semua masukan yang dikumpulkan selama beberapa periode dikumpulkan dan dimasukkan ke dalam sistem sekaligus dalam satu *batch*. Beberapa proses bisnis secara alami menghasilkan informasi dalam kelompok. Misalnya, sebagian besar penggajian per jam dilakukan dengan pemrosesan batch karena kartu "waktu kerja" dikumpulkan bersama dalam beberapa *batch* dan diproses sekaligus. Pemrosesan batch juga digunakan untuk sistem pemrosesan transaksi yang tidak memerlukan informasi *real-time*.

**Menangkap Data Langsung pada Sumbernya.** Pada masa awal komputasi, sistem komputer digunakan menggantikan sistem manual tradisional yang didasarkan pada bentuk kertas. Misalnya, formulir data pelanggan diisi dengan tangan dan diserahkan ke bagian penjualan, menyetujuinya dan memasukkannya ke dalam sistem

secara berkelompok. Ada tiga masalah dengan pendekatan ini. **Pertama**, mahal karena duplikat pekerjaan (formulir diisi dua kali, sekali dengan tangan dan sekali dengan keyboard). **Kedua**, ini meningkatkan waktu pemrosesan karena formulir kertas harus dipindahkan secara fisik melalui proses tersebut. **Ketiga**, kemungkinan terjadi kesalahan; seseorang mungkin salah membaca tulisan tangan pada formulir masukan, data yang dimasukkan salah, atau masukan asli mungkin mengandung kesalahan yang membuat informasi tidak valid.

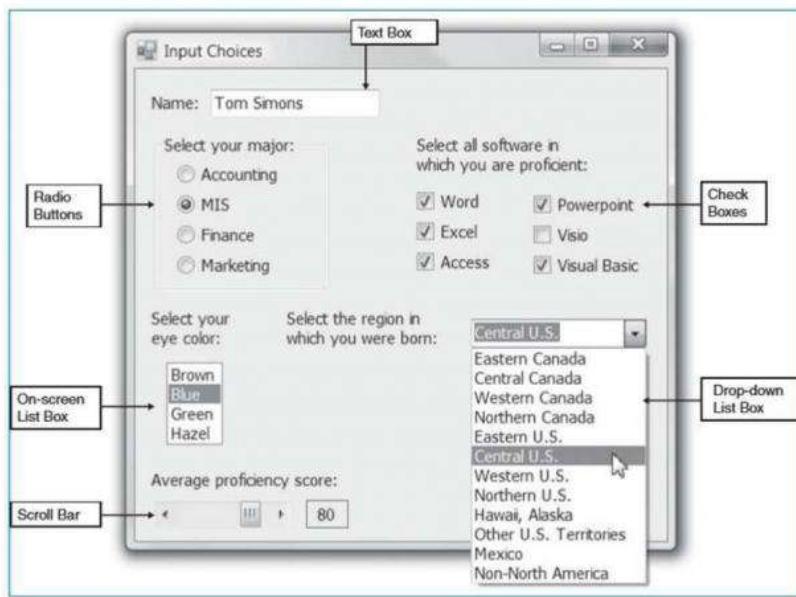
Sebagian besar sistem pemrosesan transaksi saat ini dirancang untuk menangkap data pada sumbernya, misalnya: menggunakan pembaca kode batang (*barcode*) yang secara otomatis memindai produk dan memasukkan data langsung ke sistem komputer; pembaca strip magnetik, yang dapat membaca informasi yang dikodekan pada strip bahan magnetis yang mirip dengan disket (misalnya, kartu kredit) dan kartu pintar yang berisi mikroprosesor. Perkembangan terbaru adalah tag RFID (*Radio Frequency Identification*), yang menggabungkan chip mikroprosesor dengan antena untuk mentransmisikan informasinya ke perangkat pembaca elektronik. Namun, banyak informasi yang tidak dapat dikumpulkan oleh sistem otomatis ini, sehingga sistem masukan langsung tetap digunakan. Dengan meluasnya penggunaan Web, banyak data diambil langsung dari pelanggan. Akibatnya, formulir untuk menangkap informasi di layar harus memberikan aliran logis dan harus memungkinkan pengguna untuk dengan mudah mengisi formulir dan memeriksa masukan mereka sebelum mengirimkannya. Karena data yang dimasukkan oleh pengguna rentan terhadap ketidakakuratan, pemeriksaan validasi (lihat Gambar 9.21) harus digunakan jika memungkinkan.

**Meminimalkan Keystroke.** Prinsip penting lainnya adalah meminimalkan penekanan tombol. Penekanan tombol membutuhkan waktu, baik dilakukan oleh pelanggan, pengguna, atau operator entri data yang terlatih. Sistem sebaiknya tidak mengharuskan pengguna untuk mengetik informasi yang dapat dipilih dari daftar; memilih mengurangi kesalahan dan mempercepat masuk.

## 2. Jenis Masukan

Setiap bidang masukan memiliki label bidang, yang merupakan teks di samping, di atas, atau di bawah bidang, yang memberi tahu pengguna jenis informasi apa yang termasuk dalam bidang tersebut. Seringkali, label bidang mirip dengan nama elemen data, tetapi keduanya tidak harus memiliki kata-kata yang identik. Dalam beberapa kasus, sebuah *field* akan menampilkan sebuah template di atas kotak masukan untuk menunjukkan kepada pengguna bagaimana data harus diketik. Ada banyak jenis masukan yang berbeda, seperti yang disajikan pada Gambar 9.19.

**Kotak Teks.** Kotak teks (*text box*) digunakan untuk memasukkan teks. Kotak teks dapat didefinisikan memiliki panjang tetap atau dapat menerima jumlah teks yang hampir tidak terbatas. Dalam kedua kasus tersebut, kotak dapat berisi satu atau beberapa baris informasi tekstual. Jangan pernah menggunakan kotak teks jika masih dapat menggunakan kotak pilihan.



Gambar 9.19  
Opsi Penggunaan Fitur Masukan (Dennis, 2012)

**Kotak Pilihan.** Kotak pilihan (*selection box*) memungkinkan pengguna untuk memilih nilai dari daftar yang telah ditentukan sebelumnya. Item dalam daftar harus diatur dalam beberapa urutan yang bermakna, seperti menurut abjad untuk daftar panjang, atau dalam urutan yang paling sering digunakan. Nilai pemilihan default harus disajikan dengan hati-hati.

Ada enam jenis kotak pemilihan yang umum digunakan: kotak centang (*check box*), tombol radio (*radio button*), kotak daftar di layar (*on-screen list box*), kotak daftar *drop-down* (*drop-down list box*), kotak kombo (*combo box*), dan batang gulir (*scroll bar*). Uraian setiap jenis kotak pilihan disajikan pada Gambar 9.20.

Jenis Kotak	Penggunaan	Keterangan
Kotak Centang ( <i>Check Box</i> )	Ketika beberapa item dapat dipilih dari daftar item	Kotak centang tidak saling eksklusif. Label kotak centang harus ditempatkan dalam beberapa urutan logis, seperti yang ditentukan oleh proses bisnis, menurut abjad, atau yang paling umum digunakan terlebih dahulu.
Tombol Radio ( <i>Radio Button</i> )	Ketika hanya satu item yang dapat dipilih dari satu set item yang saling eksklusif	Gunakan tidak lebih dari enam tombol radio dalam satu daftar. Jika membutuhkan lebih banyak, gunakan kotak daftar <i>drop-down</i> . Jika hanya ada dua opsi, satu kotak centang biasanya

		lebih disukai daripada dua tombol radio, kecuali opsi tidak jelas. Hindari menempatkan tombol radio di dekat kotak centang untuk mencegah kebingungan di antara daftar pilihan yang berbeda.
Kotak Daftar di Layar <i>(On-screen List Box)</i>	Digunakan hanya jika tidak ada cukup ruang untuk kotak centang atau tombol radio	Kotak ini hanya mengizinkan satu item untuk dipilih (dalam hal ini sama dengan tombol radio). Kotak ini memungkinkan daftar item untuk di- <i>scroll</i> , sehingga mengurangi jumlah ruang layar yang dibutuhkan.
Kotak Daftar <i>Drop-down</i> <i>(Drop-down List Box)</i>	Jika tidak ada cukup ruang untuk menampilkan semua pilihan	Kotak ini menyederhanakan desain jika jumlah pilihan tidak jelas, karena hanya membutuhkan satu baris saat ditutup.
Kotak Kombo <i>(Combo Box)</i>	Pintasan untuk pengguna berpengalaman	Kotak ini berfungsi seperti daftar <i>drop-down</i> , tetapi lebih cepat untuk pengguna berpengalaman jika daftar itemnya panjang.
Batang Gulir <i>(Scroll Bar)</i> vertikal atau horizontal	Saat memasukkan perkiraan nilai numerik dari skala kontinu besar	Penggeser mempersulit pengguna untuk memilih nomor yang tepat. Beberapa <i>slider</i> juga menyertakan kotak angka untuk memungkinkan pengguna memasukkan besaran angka tertentu.

Gambar 9.20  
Jenis Kotak Pilihan

Jika ruang layar terbatas dan hanya satu item yang dapat dipilih, maka kotak daftar *drop-down* adalah pilihan terbaik, karena tidak semua item daftar perlu ditampilkan di layar. Jika ruang layar terbatas, tetapi pengguna dapat memilih beberapa item, kotak daftar di layar yang hanya menampilkan beberapa item dapat digunakan. Kotak centang (untuk beberapa pilihan) dan tombol radio (untuk pilihan tunggal) keduanya membutuhkan semua item daftar untuk ditampilkan setiap saat, sehingga membutuhkan lebih banyak ruang layar, tetapi karena model ini menampilkan semua pilihan, seringkali lebih sederhana untuk pengguna pemula.

### 3. Validasi Masukan

Untuk mencegah informasi yang tidak valid memasuki sistem, sistem komputer tidak boleh menerima data penting yang gagal dalam pemeriksaan validasi. Jenis pemeriksaan validasi disajikan pada Gambar 9.21.

Jenis Validasi	Penggunaan	Keterangan
Pemeriksaan Kelengkapan (Completeness Check)	Ketika beberapa bidang isian harus dimasukkan sebelum formulir dapat diproses	Jika informasi yang diperlukan hilang, formulir tanpa proses dikembalikan ke pengguna.
Pemeriksaan Format (Format Check)	Jika bidang isian berupa angka atau berisi data berkode	Idealnya, bidang numerik seharusnya tidak mengizinkan pengguna untuk mengetik data teks, tetapi jika ini tidak memungkinkan, data yang dimasukkan harus diperiksa untuk memastikan bahwa itu adalah numerik.
Pemeriksaan rentang (Range Check)	Dengan semua data numerik, jika memungkinkan	Pemeriksaan rentang hanya mengizinkan angka di antara nilai yang benar. Sistem seperti itu juga dapat digunakan untuk menyaring data untuk "nilai dalam batas kewajaran"; misalnya, menolak tanggal lahir sebelum 1900 karena orang tidak hidup sampai lebih dari 100 tahun.
Pemeriksaan Konsistensi (Consistency Check)	Saat data saling terkait	Misalnya, tahun kelahiran seseorang harus mendahului tahun dia menikah. Meskipun tidak mungkin bagi sistem untuk mengetahui data mana yang salah, sistem dapat melaporkan kesalahan tersebut kepada pengguna untuk diperbaiki.
Pemeriksaan Database (Database Check)	Ketika data tersedia untuk diperiksa	Data dibandingkan dengan informasi dalam database (atau file) untuk memastikan kebenarannya.

Gambar 9.21  
Jenis Pemeriksaan Validasi

Hasil validasi dapat mengidentifikasi data yang tidak valid dan membuat perubahan atau memberi tahu seseorang yang dapat menyelesaikan masalah ketidakvalidan informasi.

## E. DESAIN KELUARAN

Keluaran (output) adalah laporan yang dihasilkan sistem, baik di layar, di atas kertas, atau di media lain, seperti Web.

### 1. Prinsip Dasar

Prinsip dasar untuk desain keluaran mencerminkan bagaimana keluaran digunakan dan cara untuk memudahkan pengguna dalam memahaminya.

#### a. Memahami Penggunaan Laporan

Prinsip pertama dalam mendesain laporan adalah memahami bagaimana laporan tersebut digunakan. Laporan dapat digunakan untuk berbagai tujuan. Dalam beberapa kasus (namun tidak terlalu sering) laporan dibaca dari awal sampai akhir karena semua informasi diperlukan. Dalam kebanyakan kasus, laporan digunakan untuk mengidentifikasi item tertentu atau digunakan sebagai referensi untuk menemukan informasi, sehingga item diurutkan pada laporan atau dikelompokkan dalam kategori sangat penting. Laporan web yang dimaksudkan untuk dibaca dari ujung ke ujung harus disajikan dalam satu halaman panjang yang dapat digulir, sedangkan laporan yang terutama digunakan untuk menemukan informasi tertentu harus dipecah menjadi beberapa halaman, masing-masing dengan tautan terpisah. Nomor halaman dan tanggal laporan juga penting disajikan untuk referensi laporan.

Frekuensi laporan juga berperan penting dalam desain dan distribusinya. Laporan *real time* memberikan data yang akurat hingga detik atau menit saat dibuat (misalnya, pasar saham). Laporan *batch* adalah laporan yang berisi informasi historis yang mungkin berumur berbulan-bulan, hari, atau jam, dan mereka sering memberikan informasi tambahan di luar informasi yang dilaporkan (misalnya, total, ringkasan, rata-rata historis).

#### b. Mengelola Beban Informasi

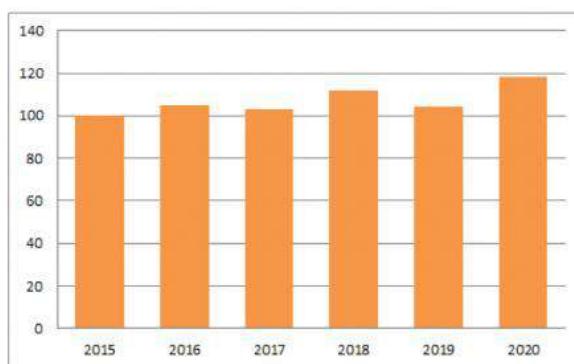
Tujuan dari laporan yang dirancang dengan baik adalah memberikan semua informasi yang diperlukan untuk mendukung tugas spesifik yang telah ditentukan. Ini tidak berarti bahwa laporan harus menyediakan semua informasi yang tersedia tentang suatu subjek, melainkan hanya apa yang menurut pengguna perlukan/spesifik). Dalam beberapa kasus, hal ini dapat mengakibatkan produksi beberapa laporan berbeda untuk topik yang sama dan untuk pengguna yang sama, karena laporan-laporan tersebut digunakan untuk keperluan yang berbeda dan dengan cara yang berbeda.

c. *Meminimalkan Bias*

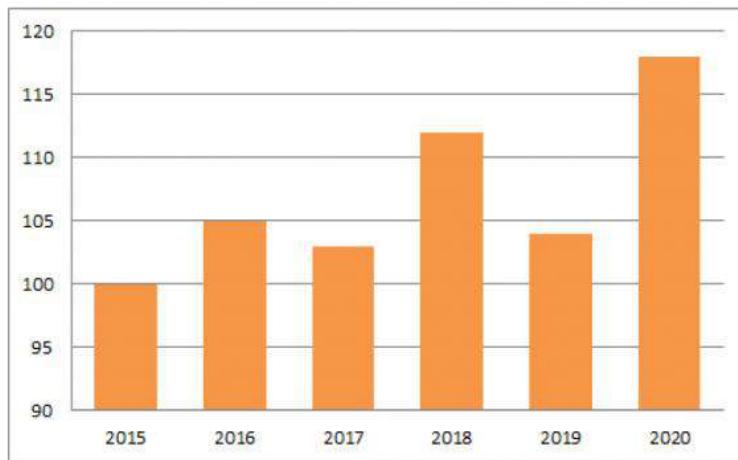
Masalah dengan bias adalah bahwa bias bisa muncul dengan sangat halus; analis mungkin secara tidak sengaja menyajikannya. Bias dapat muncul dari cara daftar data diurutkan. Data sering kali diurutkan dalam urutan abjad, membuat entri yang dimulai dengan huruf A lebih menonjol. Data dapat diurutkan dalam urutan kronologis (atau urutan kronologis terbalik), lebih menekankan pada entri yang lebih lama (atau terbaru). Data dapat diurutkan berdasarkan nilai numerik, lebih menekankan pada nilai yang lebih tinggi atau lebih rendah.

**Contoh**, pada sebuah laporan penjualan bulanan, haruskah laporan dicantumkan dalam urutan abjad menurut nama wilayah penjualan, dalam urutan menurun menurut jumlah yang terjual, atau dalam beberapa urutan lain? Tidak ada jawaban yang pasti untuk ini, kecuali bahwa urutan penyajian harus sesuai dengan cara penggunaan informasi tersebut.

**Contoh lain**, tampilan grafik dalam sebuah laporan dapat memunculkan masalah desain yang perlu perhatian. Skala pada sumbu dalam grafik sangat rentan terhadap bias. Untuk sebagian besar jenis grafik, skala harus selalu dimulai dari nol; jika tidak, perbandingan antar nilai bisa menyesatkan. Misalnya, pada Gambar 9.22 dan Gambar 9.23, apakah penjualan telah meningkat pesat sejak tahun 2015? Angka-angka di kedua grafik adalah sama, tetapi gambar visual yang ditampilkan keduanya sangat berbeda. Sekilas pada Gambar 9.22 hanya menunjukkan perubahan kecil, sedangkan sekilas pada Gambar 9.23 mungkin menunjukkan bahwa telah terjadi beberapa peningkatan yang signifikan. Faktanya, penjualan total meningkat hanya sekitar 15% selama lima tahun, atau 3% per tahun. Gambar 9.22 menyajikan gambaran yang paling akurat; Gambar 9.23 bias karena skala dimulai sangat dekat dengan nilai terendah dalam grafik dan menyesatkan mata untuk menyimpulkan bahwa telah terjadi perubahan besar (yaitu, menggandakan lebih dari dua kali lipat dari "dua garis" pada tahun 2015 menjadi "lima garis" di 2020). Jadi perhatikan betapa mudahnya untuk secara tidak sengaja memasukkan bias dalam grafik.



Gambar 9.22  
Grafik yang Tidak Bias dengan Skala Dimulai pada 0



Gambar 9.23  
Grafik yang Bias dengan Skala Dimulai pada 90

## 2. Jenis Keluaran

Ada banyak jenis laporan yang berbeda, seperti laporan detail, laporan ringkasan, laporan pengecualian, dan grafik. Penjelasan mengenai jenis-jenis laporan ini disajikan pada Gambar 9.24.

Jenis Laporan	Penggunaan	Keterangan
Laporan Detail <i>(Detail Report)</i>	Ketika pengguna membutuhkan informasi lengkap tentang item	Laporan menyajikan informasi dengan hanya sedikit atau tanpa dilakukan penyaringan atau pembatasan. Contoh: Daftar seluruh tagihan pelanggan
Laporan Rangkuman <i>(Summary Report)</i>	Ketika pengguna membutuhkan informasi singkat tentang banyak item	Laporan ini dibuat untuk membandingkan beberapa item satu sama lain. Laporan biasanya diurutkan berdasarkan kriteria tertentu. Contoh: Laporan total penjualan per bulan (dalam satu tahun)
Laporan Pengecualian <i>(Exception Report)</i>	Ketika ingin menyampaikan beberapa penyimpangan dari situasi yang normal	Data/informasi disaring sebelum disajikan sebagai sebuah informasi, sehingga dapat mengurangi informasi yang tidak dibutuhkan. Contoh: - Laporan persediaan barang yang hampir habis

Jenis Laporan	Penggunaan	Keterangan
		- Daftar pelanggan yang menunggak pembayaran
Grafik (Graph)	Saat pengguna perlu membandingkan data di antara beberapa item	Grafik membandingkan dua item atau lebih atau memahami bagaimana satu item berubah dari waktu ke waktu. Grafik batang tepat dalam hal membandingkan nilai antar item. Grafik garis memudahkan untuk membandingkan nilai dari waktu ke waktu, sedangkan grafik sebar ( <i>scatter</i> ) memudahkan untuk menemukan kelompok atau data yang tidak biasa. Grafik <i>pie</i> menunjukkan proporsi dari keseluruhan item.

Gambar 9.24  
Jenis Laporan



### Latihan

Untuk memperdalam pemahaman Anda mengenai materi di atas, kerjakanlah latihan berikut!

- 1) Jelaskan tiga dari beberapa prinsip utama desain antarmuka pengguna!
- 2) Sebutkan tiga bagian utama antarmuka pengguna berbasis Web pada umumnya!
- 3) Jelaskan proses dasar desain antarmuka pengguna!
- 4) Buat sebuah skenario penggunaan untuk sebuah situs Web yang menjual beberapa produk eceran (misalnya buku, pakaian, lainnya). Selanjutnya, buatlah desain template antarmuka dengan mengikuti tahapan-tahapan dalam proses desain antarmuka pengguna!

#### Petunjuk Jawaban Latihan

- 1) Beberapa prinsip utama dalam mendesain antarmuka pengguna, tiga di antaranya adalah berikut ini.
  - a) Memperhatikan tata letak objek. Pengaturan tata letak objek (layar, masukan, dan keluaran) mengacu pada penggunaan area yang berbeda untuk tujuan yang berbeda. Penetapan area diakukan secara konsisten di seluruh antarmuka pengguna.
  - b) Memperhatikan sistem *content awareness*. Semua bagian antarmuka, baik navigasi, masukan, atau keluaran, harus memberikan *content awareness*

sebanyak mungkin, dengan hanya sedikit usaha yang dilakukan oleh pengguna.

- c) Memperhatikan unsur estetika, misalnya: kepadatan objek sebaiknya tidak terlalu tinggi (kurang dari 50%) untuk pengguna yang jarang berinteraksi dengan antarmuka; keseragaman penggunaan font, baik tipe maupun ukuran; penggunaan warna hanya jika memiliki tujuan (misalnya memperkuat pesan).
- 2) Bagian-bagian utama antarmuka pengguna berbasis Web pada umumnya:
  - a) Area atas layar menyediakan navigasi browser web standar (disebut navigasi sistem) dan kontrol perintah untuk mengubah konten seluruh sistem.
  - b) Area navigasi di tepi kiri (navigasi situs), digunakan untuk bermanuver dan mengubah semua konten di sebelah kanannya.
  - c) Area di tengah halaman, digunakan untuk menampilkan hasil navigasi berupa ulasan. Di dalam halaman ulasan ini terkadang berisi navigasi tambahan (disebut navigasi halaman).
- 3) Proses desain antarmuka:
  - a) Mengembangkan skenario penggunaan yang menggambarkan pola tindakan yang umum dilakukan pengguna.
  - b) Menggambar struktur diagram antarmuka yang menunjukkan semua antarmuka (layar, formulir, dan laporan) dan hubungan antar antarmuka.
  - c) Mendesain template setiap antarmuka, didahului dengan menyusun standar antarmuka untuk setiap template.
  - d) Mendesain prototipe dan melakukan uji coba kepada user, hingga diperoleh desain prototipe antarmuka yang dinyatakan sempurna.
- 4) Dapat mengacu pada contoh kasus mendesain antarmuka pengguna aplikasi Sistem Perpustakaan.



## Rangkuman

Elemen utama dari desain antarmuka pengguna adalah tata letak layar, formulir, dan laporan, yang biasanya digambarkan dengan bentuk persegi panjang dengan area atas untuk navigasi, area tengah untuk input dan output, dan area status di bagian bawah. Desain harus membantu pengguna menyadari konten dan konteks, baik di antara berbagai bagian sistem saat mereka menavigasi maupun dalam satu formulir atau laporan. Semua antarmuka harus menyenangkan secara estetika dan perlu menyertakan area kosong yang cukup, menggunakan warna dengan hati-hati, dan font konsisten. Kebanyakan antarmuka harus dirancang untuk mendukung pengguna pemula serta pengguna berpengalaman. Konsistensi dalam desain (baik di dalam sistem dan di

seluruh sistem lain yang digunakan oleh pengguna) penting untuk setiap kontrol navigasi, terminologi, dan tata letak formulir dan laporan. Semua antarmuka harus berusaha meminimalkan upaya pengguna, misalnya, dengan tidak memerlukan lebih dari tiga klik dari menu utama untuk melakukan sebuah tindakan.

Prosedur pengembangan desain antarmuka: pertama, analis mengembangkan skenario penggunaan yang menggambarkan pola tindakan yang umum digunakan yang akan dilakukan pengguna. Kedua, analis merancang struktur antarmuka melalui ISD berdasarkan DFD. Ketiga, analis mendefinisikan standar antarmuka, yang terdiri atas objek, tindakan, dan ikon. Elemen-elemen ini digabungkan dengan desain template antarmuka dasar untuk setiap bagian utama dari sistem. Keempat, analis membuat dan menguji prototipe antarmuka, serta mengevaluasinya untuk mengidentifikasi peningkatan mengarah ke prototipe antarmuka yang lebih sempurna.

Desain navigasi ditujukan untuk membuat sistem sesederhana mungkin dengan mencegah pengguna dari membuat kesalahan. Bahasa perintah dan manipulasi langsung digunakan dalam navigasi, tetapi pendekatan yang paling umum adalah menu (dalam berbagai bentuk). Pesan kesalahan, pesan konfirmasi, pesan pengakuan, pesan penundaan, dan pesan bantuan adalah jenis pesan yang umum dalam menavigasi.

Tujuan dari desain masukan adalah untuk secara sederhana dan mudah menangkap informasi yang akurat untuk sistem, biasanya dengan menggunakan pemrosesan *online* atau *batch*, menangkap data pada sumbernya, dan meminimalkan penekanan tombol. Desain masukan mencakup desain layar masukan dan semua formulir pracetak yang digunakan untuk mengumpulkan data sebelum dimasukkan ke dalam sistem informasi. Ada banyak jenis masukan, seperti *text field*, *number field*, *check box*, *radio button*, *on-screen list box*, *drop-down list box*, dan *slider*. Sebagian besar masukan divalidasi dengan beberapa kombinasi pemeriksaan kelengkapan, pemeriksaan format, pemeriksaan rentang, pemeriksaan digit, pemeriksaan konsistensi, dan pemeriksaan database.

Tujuan desain keluaran adalah untuk menyajikan informasi kepada pengguna sehingga mereka dapat memahaminya secara akurat dengan sedikit usaha, biasanya dengan memahami bagaimana laporan akan digunakan dan merancangnya untuk meminimalkan informasi yang berlebihan dan bias. Desain keluaran berarti mendesain layar dan laporan di media lain, seperti kertas dan Web. Ada banyak jenis laporan, seperti laporan detail, laporan ringkasan, laporan pengecualian, dan grafik.



## Tes Formatif 2

Pilihlah satu jawaban yang paling tepat!

- 1) Antarmuka pengguna (*user interface*) mencakup tiga bagian dasar, *kecuali* ....
  - A. mekanisme proses
  - B. mekanisme navigasi
  - C. mekanisme masukan
  - D. mekanisme keluaran

- 2) Karakteristik sebuah antarmuka yang mampu membuat pengguna sadar akan informasi yang terkandung di dalamnya, dengan hanya sedikit usaha oleh pengguna, disebut ....
  - A. estetika
  - B. *content awareness*
  - C. *user experience*
  - D. konsistensi
- 3) Desain antarmuka dengan tingkat kepadatan konten yang rendah, cocok diperuntukkan bagi ....
  - A. pengguna usia remaja
  - B. pengguna usia dewasa
  - C. pengguna yang jarang berinteraksi dengan antarmuka
  - D. pengguna yang lebih berpengalaman
- 4) Hal-hal yang perlu diperhatikan dalam merancang antarmuka pengguna, yang terkait dengan pengalaman pengguna (*user experience*) adalah ....
  - A. pengguna pemula pada umumnya lebih mementingkan kemudahan mempelajari cara penggunaan sistem
  - B. pengguna ahli pada umumnya lebih mementingkan kemudahan penggunaan sistem
  - C. sistem ini harus lebih menekankan pada kemudahan penggunaan daripada kemudahan belajar
  - D. semua jawaban A, B, dan C benar
- 5) Berikut adalah bentuk-bentuk konsistensi dalam mendesain antarmuka pengguna, *kecuali* ....
  - A. konsistensi dalam penggunaan kontrol navigasi
  - B. konsistensi dalam penggunaan warna
  - C. konsistensi dalam terminologi
  - D. konsistensi dalam desain formulir dan desain laporan
- 6) Proses desain antarmuka pengguna merujuk pada ... dalam fase pemodelan analisis.
  - A. *use case diagram*
  - B. *entity relationship diagram*
  - C. DFD dan atau *hirarchy chart*
  - D. dokumen *flowchart*

- 7) Tiga pendekatan paling mendasar dalam mendefinisikan perintah bagi pengguna perangkat lunak, *kecuali* ....
- perintah melalui suara
  - perintah melalui bahasa program
  - perintah melalui menu
  - perintah melalui manipulasi langsung
- 8) Prinsip yang perlu diperhatikan dalam mendesain masukan (*input*) adalah ....
- meminimalkan penekanan tombol
  - menggunakan sistem pemrosesan *online* dan pemrosesan *batch* dengan tepat
  - menangkap data langsung pada sumbernya
  - semua jawaban A, B, dan C benar
- 9) Dalam mendesain validasi masukan, jika bidang isian berupa angka atau berisi data berkode, maka jenis validasi yang tepat berupa ....
- pemeriksaan kelengkapan
  - pemeriksaan format
  - pemeriksaan rentang
  - pemeriksaan digit
- 10) Jenis laporan berupa grafik tepat digunakan apabila ....
- pengguna membutuhkan informasi lengkap tentang suatu item
  - pengguna membutuhkan informasi singkat tentang banyak item
  - pengguna perlu membandingkan data antara beberapa item
  - jawaban A dan B benar

Cocokkanlah jawaban Anda dengan Kunci Jawaban Tes Formatif 2 yang terdapat di bagian akhir modul ini. Hitunglah jawaban yang benar. Kemudian, gunakan rumus berikut untuk mengetahui tingkat penguasaan Anda terhadap materi Kegiatan Belajar 2.

$$\text{Tingkat Penguasaan} = \frac{\text{Jumlah Jawaban yang Benar}}{\text{Jumlah Soal}} \times 100$$

Arti tingkat penguasaan

<70%

70% - 79%

80% - 89%

90% - 100%

kurang

cukup

baik

baik sekali

Apabila mencapai tingkat penguasaan 80% atau lebih, Anda dapat mengikuti Ujian Akhir Semester (UAS). **Selamat!** Jika masih di bawah 80%, Anda harus mengulangi materi Kegiatan Belajar 2, terutama bagian yang belum dikuasai.

## Kunci Jawaban Tes Formatif

### *Tes Formatif 1*

- 1) C
- 2) B
- 3) D
- 4) B
- 5) C
- 6) D
- 7) A
- 8) A
- 9) C
- 10) C

### *Tes Formatif 2*

- 1) A
- 2) B
- 3) C
- 4) D
- 5) B
- 6) C
- 7) A
- 8) D
- 9) B
- 10) C

## Glosarium

Aliran intuitif	: Aliran proses yang secara alami mudah dipahami tanpa terlebih dahulu memikirkan atau mempelajarinya secara mendalam
Batch processing	: Proses menghimpun data selama kurun waktu tertentu, untuk selanjutnya dimasukkan ke dalam sistem melalui antarmuka masukan
Broadband	: Koneksi Internet untuk transmisi data berkecepatan tinggi
Cloud Computing	: Paradigma di mana informasi secara permanen tersimpan di server internet dan tersimpan secara sementara di komputer pengguna
Content awareness	: Kemampuan sebuah antarmuka untuk membuat pengguna sadar akan informasi yang terkandung di dalamnya
Frame relay	: Cara pengiriman informasi melalui jaringan area yang luas, yang membagi informasi menjadi paket-paket
Klien-server	: Arsitektur jaringan yang memiliki server yang ditugaskan secara khusus untuk melayani komputer klien
Komputer klien	: Sistem komputer yang meminta/menggunakan jenis layanan tertentu dalam sebuah jaringan komputer
Komputer mainframe	: Istilah yang mengacu kepada kelas/spesifikasi tertinggi dari komputer yang terdiri dari komputer-komputer yang mampu melakukan banyak tugas komputasi yang besar dan rumit dalam waktu yang cepat
Komputer server	: Sistem komputer yang menyediakan jenis layanan tertentu dalam sebuah jaringan komputer
Logika akses data	: Suatu prosedur pemrosesan yang digunakan untuk mengakses data/informasi yang terdapat dalam sebuah perangkat komputasi

- Logika aplikasi** : Suatu prosedur pemrosesan yang digunakan untuk menjalankan fungsi-fungsi bisnis yang telah ditanamkan di dalam aplikasi komputer
- Logika presentasi** : Suatu prosedur pemrosesan yang digunakan untuk menyajikan data/informasi yang terdapat dalam sebuah perangkat komputasi
- Multi-user** : Suatu sistem di mana lebih dari satu pengguna menggunakan secara bersama satu atau lebih perangkat keras, perangkat lunak, data/informasi, dan prosedur melalui masing-masing komputer atau terminal lainnya
- Prototipe** : Model permulaan (yang menjadi contoh) dari desain antarmuka
- Real-time:** : Waktu nyata, adalah kondisi pengoperasian dari suatu sistem perangkat keras dan perangkat lunak yang dibatasi oleh rentang waktu dan memiliki tengat waktu yang jelas, relatif terhadap waktu suatu peristiwa atau operasi terjadi
- Skenario Penggunaan** : Garis besar langkah-langkah yang dilakukan pengguna untuk menyelesaikan beberapa bagian dari suatu pekerjaan, yang dituangkan dalam proses bisnis
- Thin client** : Perangkat komputer kecil atau ramping atau tipis tanpa fitur (seperti CD-ROM atau hardisk) yang dikelola secara terpusat di server
- Tooltip** : Area yang menampilkan frasa teks yang menjelaskan tombol saat pengguna menjeda penunjuk di atasnya
- User experience** : Proses meningkatkan kepuasan pengguna aplikasi dalam rangka meningkatkan kegunaan dan kesenangan yang diberikan dalam interaksi antara pengguna dan sistem informasi
- Workstation** : Stasiun kerja, terminal komputer yang digunakan untuk membantu user mengerjakan tugas lebih spesifik, biasanya membutuhkan spesifikasi yang tinggi dari komputer desktop
- Zero client  
(ultra thin client)** : Mirip thin client, namun fitur yang harus dikelola dan dipelihara di perangkat kliennya lebih sedikit

## Daftar Pustaka

- Dennis, A., Wixom, B. H., & Roth, R.M. (2012). *Systems analysis & design* (edisi kelima). United States of America: John Wiley & Sons, Inc.
- Pressman, R.S. (2002). *Rekayasa perangkat lunak: Pendekatan praktisi* (buku satu). Yogyakarta: ANDI.
- Sommerville, I. (2003). *Rekayasa perangkat lunak* (edisi keenam). Jakarta: Erlangga.
- Sulistia, F. (2017). *Teknik perancangan arsitektur sistem informasi*. Yogyakkarta: Andi.

## Riwayat Penulis



**Nama Lengkap :** Bahar, S.T., M.Kom.  
**Bidang Ilmu :** Teknik Informatika  
**NIDN :** 1107047001  
**Institusi :** STMIK Banjarbaru  
**Alamat Institusi :** Jl. A. Yani K.M. 33,5  
Loktabat - Banjarbaru  
**Telepon :** 0511 4782881  
**Email :** bahararahman@gmail.com

### Pendidikan:

- S1 Teknik Telekomunikasi  
Universitas Muslim Indonesia Makassar
- S2 Teknik Informatika  
Universitas Dian Nuswantoro Semarang
- S3 Teknologi Pendidikan (*on going*)  
Universitas Negeri Jakarta Jakarta

### Publikasi:

- Development of Instructional Media Based on Mobile Technology to Enriching Teaching Material for Primary School Students In Indonesia Post-Learning in the Classrooms. (*International Journal of Scientific & Technology Research – Scopus Indexed – 2020*)
- The Problem-Based Learning Procedural Model in the Software Modeling Course at the Information Technology College in Indonesia (*Universal Journal of Educational Research – Scopus Indexed – 2019*)
- Using Problem-Based Learning to Teach Software Modeling in Information Technology Colleges in Indonesia: A Concept (*Unnes Science Educational Journal*, SINTA 3 Indexed, 2019)
- Penggunaan Teknologi Mobile sebagai Pendukung Program Pembelajaran Pengendalian Berat Badan (*Progresif: Jurnal Ilmiah Komputer*, SINTA 4 Indexed, 2018)
- Rancangan Sistem Informasi Pemetaan Kompetensi pada Area TIK untuk Mendukung Sistem Instruksional di Perguruan Tinggi (*SOLITER*, 2017)

**9.62 Desain/Perancangan Sistem**

---

- Distance Learning Based E-Learning Operational Model for Student Non-Regular (*Proceeding in International Seminar on Electronic & Mobile Learning*, 2016)