BAB 2
LANDASAN TEORI

2.1 *Carpool*

Pool atau *carpool* adalah tempat untuk penyimpanan, pemeliharaan, dan perbaikan kendaraan dalam jumlah yang besar untuk suatu organisasi atau perorangan. Pada kasus penelitian ini *pool* merupakan parkir khusus untuk persediaan kendaraan dan bersifat steril terhadap orang yang tidak berkepentingan untuk menjaga keamanan dan kualitas kendaraan. Pada *pool* persediaan kendaraan terdapat beberapa area seperti berikut (PT Tunas Ridean Tbk, 2016):

2.1.1 Area Parkir

Pool mempunyai beberapa tipe area parkir seperti (PT Tunas Ridean Tbk, 2016):

1. Area *transit in*, area parkir sementara saat mobil antre untuk diperiksa sebelum masuk area utama parkir.
2. Area *transit out*, area parkir sementara setelah mobil selesai diperiksa dan sebelum mobil dikirim.
3. Area utama parkir, area parkir utama merupakan area paling steril untuk penyimpanan kendaraan.
4. Area *other* atau lainnya, area parkir yang dapat digunakan untuk beberapa kasus tertentu seperti mobil yang tidak dikenal atau mobil hasil salah pengiriman.
5. Area karantina, area parkir yang digunakan untuk meletakan sementara mobil yang keadaannya tidak memenuhi standar.
6. Area pemeriksaan, area tempat memeriksa kelengkapan dan keadaan mobil yang masuk *pool*.

2.1.2 *Stall*

Stall dapat diartikan sebagai sebuah meja besar, stan atau kedai kecil dengan bagian depan yang terbuka untuk menjual barang di area publik (PT

Tunas Ridean Tbk, 2016). *Stall* pada *carpool* berbentuk seperti bengkel-bengkel kecil dengan fungsi dan kategori yang berbeda. Pada *carpool* khusus logistik biasanya terdapat beberapa macam *stall* seperti *stall* diagnosa dan pemeriksaan, *stall body repair*, *stall* akssoris, serta *stall* cuci.

2.2 Agen Tunggal Pemegang Merk (ATPM).

Agen Tunggal Pemegang Merek *(*ATPM*)* ialah perusahaan nasional yang ditunjuk oleh perusahaan manufaktur pemilik merek, untuk secara ekslusif mengimpor, memasarkan, mendistribusikan, serta melayani layanan purna jual pada wilayah tertentu. Pada awalnya Pemerintah Indonesia merencanakan ATPM menjadi pelopor bagi perkembangan otomotif di Indonesia melalui transfer pengetahuan teknologi untuk menghasilkan produk yang berkualitas. Saat ini ada banyak ATPM yang memegang merek mobil yang beredar di Indonesia, seperti Toyota, Isuzu, Daihatsu, Honda, Nissan, BMW, Ford, Chevroler, dan merek lainnya (Saputra, 2013).

ATPM juga diawasi dan diikat dengan ketentuan-ketentuan dari pemerintah, seperti peraturan pajak PMK-16/PMK.010./2016 atau peraturan dari Komisi Pengawas Persaingan Usaha (KPPU) Repulbik Indonesia tentang keagenan pada Pedoman Pasal 50 d (Komisi Pengawas Persaingan Usaha, 2016).

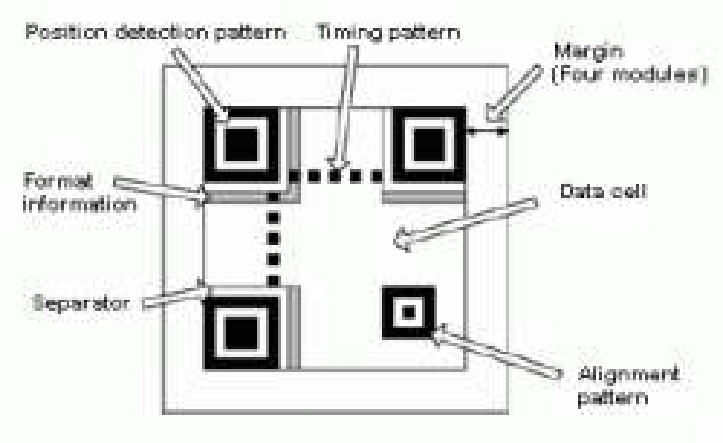
2.3 *QR Code*

QR Code atau *Quick Response Code* adalah jenis barcode yang berisi matriks titik-titik. *QR Code* dapat dipindai dengan *QR Scanner* atau smartphone. Setelah dipindai, perangkat lunak pada *device* akan mengubah titik-titik kode menjadi angka atau karakter.

Awalnya kode QR digunakan untuk pelacakan kendaraan bagian di manufaktur, namun kini kode QR digunakan dalam konteks yang lebih luas, termasuk aplikasi komersial dan kemudahan pelacakan aplikasi berorientasi yang ditujukan untuk pengguna telepon seluler. Di Jepang, penggunaan kode QR sangat populer, hampir semua jenis ponsel di Jepang bisa membaca kode QR sebab sebagian besar pengusaha di sana telah memilih kode QR sebagai alat tambahan dalam program promosi produknya, baik yang bergerak dalam perdagangan maupun dalam bidang jasa (Christensson, 2014).

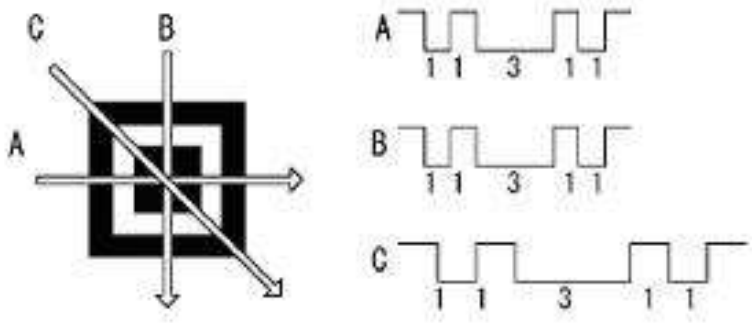
2.3.1. Prinsip Kerja

Fungsi dari *Quick Response Code* (*QR Code)* hampir sama dengan sistim *barcode* satu dimensi yang kita kenal selama ini yaitu digunakan untuk mengidentifikasi suatu barang secara cepat dan mudah, tetapi di era modern saat ini *QR Code* ini bisa digunakan lebih luas untuk segala macam kebutuhan seperti tiketing pesawat, tiket bioskop, iklan, MMS, kartu nama, dalam bidang post digunakan sebagai perangko *online*, dan dalam bidang industri digunakan sebagai kode informasi untuk komponen elektronika, perhiasan dan lainnya. Sehingga dapat disimpulkan keuntungan yang paling utama dari penggunaan 2D adalah efisiensi, kecepatan, ketepatan dan keamanan data serta mengalokasian waktu yang ada. Struktur dan prinsip kerja dari *barcode 2D* ini adalah sebagai berikut:



Gambar 2.1 Struktur *QR code*
(Sumber: Yudhistira, 2012)

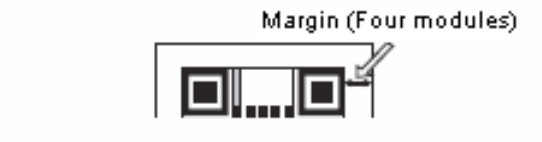
1. *Position detection patterns*



Gambar 2.2 *Position detection patterns QR code*
(Sumber: Yudhistira ,2012)

Posisi pola Deteksi diatur pada tiga sudut kode QR. Dari posisi A, B dan C, laju modul hitam dan putih 1:1:3:1:1 untuk menentukan sudut rotasi/perpindahan kode. Hal ini dapat dibaca dari segala arah, secara signifikan meningkatkan efisiensi kerja.

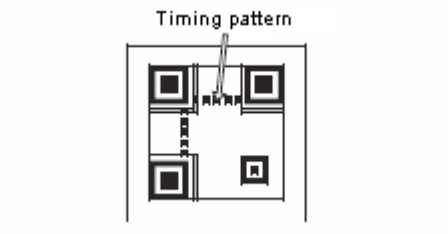
2. *Margin*



Gambar 2.3 *Margin QR code*
(Sumber: Yudhistira, 2012)

Area kosong di sekitar kode QR. Model 1 dan 2 membutuhkan margin sebesar empat modul dan Mikro kode QR membutuhkan dua modul.

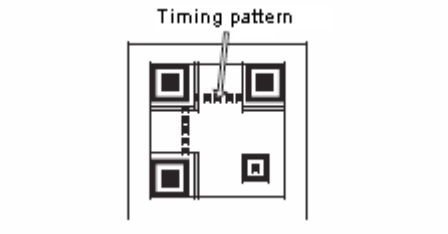
3. *Timing pattern*



Gambar 2.4 *Timing pattern QR code*
(Sumber: Yudhistira, 2012)

Putih dan modul hitam diatur secara bergantian untuk menentukan koordinat. Pola waktu ditempatkan di antara dua pola deteksi posisi dalam kode QR.

4. *Format information*



Gambar 2.5 *Format information QR code*
(Sumber: Yudhistira, 2012)

Berisi tingkat kesalahan koreksi dan pola topeng kode QR. Informasi format dibaca pertama ketika kode tersebut diterjemahkan (Yudhistira, 2012).

2.3.2. Kelebihan *QR Code*

Quick Response Code (kode QR) memiliki kapasitas tinggi dalam data pengkodean, yaitu mampu menyimpan semua jenis data, seperti data numerik, data alphabetis, kanji, kana, hiragana, simbol, dan kode biner. Secara spesifik, kode QR mampu menyimpan data jenis numerik sampai dengan 7.089 karakter, data alphanumerik sampai dengan 4.296 karakter, kode binari sampai dengan 2.844 byte, dan huruf kanji sampai dengan 1.817 karakter. Selain itu kode QR memiliki tampilan yang lebih kecil daripada kode batang. Hal ini dikarenakan kode QR mampu menampung data secara horisontal dan vertikal, oleh karena itu secara otomatis ukuran dari tampilannya gambar kode QR bisa hanya seperspuluh dari ukuran sebuah kode batang. Tidak hanya itu kode QR juga tahan terhadap kerusakan, sebab kode QR mampu memperbaiki kesalahan sampai dengan 30%. Oleh karena itu, walaupun sebagian simbol kode QR kotor ataupun rusak, data tetap dapat disimpan dan dibaca. Tiga tanda berbentuk persegi di tiga sudut memiliki fungsi agar simbol dapat dibaca dengan hasil yang sama dari sudut manapun sepanjang 360 derajat (Yudhistira, 2012).

2.4 Sistem Informasi

Sistem merupakan kumpulan dari komponen yang saling berhubungan yang berfungsi secara bersama-sama untuk mencapai sejumlah hasil (Satzinger et al., 2009). Informasi merupakan data yang sudah diolah sehingga data tersebut mempunyai makna dan nilai untuk penerima (Rainer et al., 2015).

Sistem informasi merupakan suatu set dari komponen computer yang saling berhubungan yang mengumpulkan, memproses, menyimpan, dan menyajikan pengeluaran berupa informasi yang dibutuhkan untuk menyelesaikan suatu tugas bisnis (Satzinger et al., 2009).

2.4.1. Komponen Sistem Informasi

Berikut komponen sistem informasi (Rainer et al., 2015):

1. *People*, merupakan seseorang yang menggunakan serta berinteraksi dengan *hardware* dan *software* serta menggunakan hasil keluarannya.
2. *Hardware*, perangkat keras yang menerima data dan informasi, memprosesnya, kemudian menghasilkan keluaran. Perangkat keras dapat berupa unit proses seperti *processor*, unit *input* seperti *mouse* dan *keyboard*, serta unit *output* seperti *monitor* dan *spreaker*.
3. *Software*, perangkat lunak yang memproses data yang telah masuk.
4. *Database*, sekumpulan file yang saling berhubungan atau sekumpulan tabel yang berisi data.
5. *Network*, sistem yang menghubungkan antar computer yang berbeda untuk berbagai *resource*.
6. *Procedures*, sekumpulan intruksi yang berisi tentang bagaimana cara mengkombinasikan komponen-komponen sistem informasi lainnya untuk memproses data dan menghasilkan keluaran yang diingikan.

2.5 *System Development Life Cycle*

Membangun sebuah sistem informasi menggunakan *System Development Life Cycle* (SDLC) mengikuti empat fase fundamental yaitu *planning, analysis*, *design*, dan *implementasi*. Terdapat dua poin penting tentang SDLC. Pertama, pengetahuan umum tentang teknik-teknik yang menghasilkan *deliverable* atau menghasilkan nilai tertentu. Kedua, SDLC merupakan proses yang runtut.

Deliverable dihasilkan dalam fase analisis yang menyediakan pemikirian-pemikiran bagaimana sistem baru akan berjalan. *Deliverable* digunakan sebagai input pada fase desain, kemudian hal tersebut menentukan pengembangan untuk

menghasilkan *deliverable* lain yang lebih detail tentang bagaimana sistem seharusnya dibangun. Pada fase implementasi *deliverable* digunakan sebagai acuan pembuatan sistem. Berikut fase SDLC menurut Alan Dennis (Dennis et al., 2000):

1. *Planning*

   Fase *planning* merupakan proses fundamental tentang memahami kenapa sebuah sistem informasi seharusnya dibangun dan ditentukan bagaimana tim proyek akan berjalan untuk membangun sistem informasi. Fase *planning* mempunyai dua langkah:

   - *Project initiation*, mengidentifikasi nilai bisnis sistem terhadap organisasi. Sebuah sistem *request* mewakili ringkasan singkat tentang kebutuhan bisnis, sistem *request* juga akan menjelaskan bagaimana sistem akan membantu memenuhi kebutuhan bisnis dan membuat nilai bisnis.
   - Setelah proyek disetujui, akan memasuki langkah *project management* yang akan menghasilkan *work plan* dan tim proyek. *Deliverable* dari *project management* adalah *project plan* yang mendiskripsikan bagaimana tim *project* akan menjalankan pengembangan sistem.

2. *Analysis*

   Fase analisis menjawab pertanyaan siapa yang akan menggunakan sistem, apa yang akan sistem lakukan, dimana dan kapan sistem digunakan. Pada fase ini tim proyek melakukan investigasi pada sistem berjalan, indentifikasi peluang pengembangan, mengembangkan konsep dari sistem baru. Fase ini mempunyai tiga langkah:

   - Pengembangan strategi analisis untuk memandu kinerja tim proyek.
   - *Requirements gathering* seperti wawancara, diskusi, atau kuisoner. Analisa dari informasi dini dan masukan dari sponsor proyek dan orang lain akan dijadikan sebagai *system concept*. Kemudian *system concept* akan digunakan untuk mengembangakan *analysis models* yang mendiskripsikan bisnis akan berjalan jika sistem baru sudah berjalan.
   - Analisis, *system concept*, dan *analysis models* dikombinasikan menjadi sebuah dokumen disebut *system proposal*.

3. *Design*

Fase desain menentukan bagaimana sistem akan beroperasi pada *hardware*, *software*, infrastruktur jaringan, *user interface*, form, laporan, spesifik program, *database*, dan file yang dibutuhkan. Fase *design* mempunyai empat langkah:

- *Design strategy* harus ditentukan. Hal ini mengklarifikasi apakah sistem akan dikembangkan oleh *programer* perusahaan atau menggunakan tenaga kerja luar, apakah perusahaan akan membeli sebuah paket perangkat lunak yang sudah ada.
- Mengarahkan pengembangan dari dasar desain arsitektur untuk sistem, berisi penjelasan infrastruktur *hardware*, *software*, dan jaringan yang akan digunakan.
- Database dan spesifikasi file dikembangkan, mendefinisikan data apa yang akan disimpan dan dimanadata akan disimpan.
- Tim analis mengembangkan desain program, mendefinisikan program yang butuh dituliskan dan bagaimana program akan berjalan.

4. *Implementation*

Fase implementasi merupakan fase terakhir. Fase ini kerap menjadi perhatian yang paling besar, karena merupakan fase yang memakan banyak waktu dan biaya karna melibatkan banyak stakeholder. Fase implementasi memiliki tiga langkah:

- Pembangunan sistem. Sistem ini dibangun dan diuji untuk memastikan bahwa sistem melakukan seperti yang dirancang. Karena biaya memperbaiki bug bisa sangat besar, pengujian adalah salah satu langkah paling penting dalam implementasi. Sebagian besar organisasi menghabiskan lebih banyak waktu dan perhatian pada pengujian daripada menulis program.
- Instalasi sistem. Proses dimana sistem yang lama dimatikan dan yang baru dihidupkan.
- Tim analis menetapkan rencana pasca-implementasi formal atau informal terhadap sistem untuk mengidentifikasi perubahan besar dan kecil yang kemudian diperlukan untuk sistem.

Semakin berkembangnya pendekatan ke SDLC dan banyak variasi untuk proyek yang memiliki berbagai kebutuhan. Satzinger dalam bukunya menjelaskan bahwa ada serangkaian proses inti yang selalu diperlukan, berikut ini enam proses inti yang diperlukan dalam pengembangan aplikasi baru (Satzinger et al., 2009):

1. Identifikasi masalah atau kebutuhan dan dapatkan persetujuan untuk melanjutkan.
2. Rencanakan dan pantau proyek mengenai apa yang harus dilakukan, bagaimana melakukannya dan siapa yang melakukannya.
3. Temukan dan pahami detail masalah atau kebutuhannya.
4. Desain komponen sistem yang memecahkan masalah atau memenuhi kebutuhan.
5. Membangun, menguji, dan mengintegrasikan komponen sistem.
6. Lakukan pengujian sistem dan *publish* di *production environment*.

2.6 *Object-Oriented Analyst And Design* (OOAD)

Object-Oriented Analysis (OOA) adalah semua jenis objek yang melakukan pekerjaan dalam sistem dan menunjukkan interaksi pengguna apa yang dibutuhkan untuk menyelesaikan tugas tersebut. Objek diartikan suatu hal dalam sistem komputer yang dapat merespon pesan (Satzinger et al., 2009).

Object-Oriented Design (OOD) adalah semua jenis objek yang diperlukan untuk berkomunikasi dengan orang dan perangkat dalam sistem, menunjukkan bagaimana objek berinteraksi untuk menyelesaikan tugas, dan menyempurnakan definisi dari masing-masing jenis objek sehingga dapat diimplementasikan dengan bahasa tertentu atau lingkungan (Satzinger et al., 2009).

Object-Oriented Programming (OOP) menuliskan laporan dalam bahasa pemrograman untuk mendefinisikan apa yang setiap jenis objek ini termasuk pesan bahwa pengirim satu sama lain (Satzinger et al., 2009).

Object-Oriented Analyst and Design (OOAD) adalah Teknik pendekatan yang digunakan dalam analisis dan desain dari sebuah aplikasi atau sistem melalui penerapan paradigma dan konsep yang berorientasi objek termasuk pemodelan visual. Ini diterapkan di sepanjang siklus pengembangan aplikasi atau sistem,

mendorong kualitas produk menjadi lebih baik dan mendorong *stakeholder* ikut berperan dan berkomunikasi (Janssen, 2018).

2.6.1. Objek dan *Class*

Objek merupakan sebuah entitas yang memiliki identitas, status, dan perilaku. Contoh dari objek misalnya pelanggan yang merupakan entitas dengan identitas yang spesifik, dan memiliki status dan perilaku tertentu yang berbeda antara satu pelanggan dengan pelanggan yang lain. Sedangkan *class* merupakan deskripsi dari kumpulan objek yang memiliki struktur, pola perilaku, dan atribut yang sama. Untuk dapat lebih memahami objek, biasanya objek-objek tersebut sering digambarkan dalam bentuk class (Mathiassen et al., 2000).

2.6.2. Konsep *Object Oriented Analysis and Design*

Terdapat tiga buah konsep atau teknik dasar dalam proses analisis dan perancangan berorientasi objek, yaitu (Mathiassen et al., 2000):

1. *Encapsulation*

   Encapsulation dalam bahasa pemrograman berorientasi objek secara sederhana berarti pengelompokkan fungsi. Pengelompokkan ini bertujuan agar developer tidak perlu membuat coding untuk fungsi yang sama, melainkan hanya perlu memanggil fungsi yang telah dibuat sebelumnya.

2. *Inheritance*

   Inheritance dalam bahasa pemrograman berorientasi objek secara sederhana berarti menciptakan sebuah class baru yang memiliki sifat-sifat dan karakteristik-karakteristik sama dengan yang dimiliki *class* induknya disamping sifat-sifat dan karakteristik-karakteristk individualnya.

3. *Polymorphism*

   Polymorphism berarti kemampuan dari tipe objek yang berbeda untuk menyediakan atribut dan operasi yang sama dalam hal yang berbeda. *Polymorphism* adalah hasil natural dari fakta bahwa objek dari tipe yang berbeda atau bahkan dari sub-tipe yang berbeda dapat menggunakan atribut dan operasi yang sama.

2.6.3. Keuntungan dan Kelemahan *Object Oriented Analysis and Design*

Terdapat dua kemampuan sistem berorientasi objek, yaitu (McLeod, 2015):

1. *Reusability*

   Kemampuan untuk menggunakan kembali pengetahuan dan kode program yang ada, dapat menghasilkan keunggulan saat suatu sistem baru dikembangkan atau sistem yang ada dipelihara atau direkayasa ulang. Setelah suatu objek diciptakan, ia dapat digunakan kembali, mungkin hanya dengan modifikasi kecil di sistem lain. Ini berarti biaya pengembangan yang ditanamkan di satu proyek dapat memberikan keuntungan bagi proyek-proyek lain.

2. *Interoperability*

   Kemampuan untuk mengintegrasikan berbagai aplikasi dari beberapa sumber, seperti program yang dikembangkan sendiri dan perangkat lunak jadi, serta menjalankan aplikasi-aplikasi ini di berbagai *platform* perangkat keras.

Reusability dan interoperability menghasilkan empat keunggulan kuat yaitu:

1. Peningkatan kecepatan pembangunan, karena sistem dirancang seperti dunia nyata melihatnya.
2. Pengurangan biaya pengembangan, karena pengembangan lebih cepat.
3. kode berkualitas tinggi memberikan keandalan lebih besar dan ketangguhan yang lebih dibandingkan yang biasa ditemukan dalam sistem berorientasi proses.
4. Pengurangan biaya pemeliharaan dan rekayasa ulang sistem, karena kode yang berkualitas tinggi dan kemampuan pemakaian kembali.

Berikut keuntungan lain menggunakan OOAD diantaranya adalah (Mathiassen et al., 2000):

1. OOAD memberikan informasi yang jelas mengenai konteks sistem.

2. Dapat menangani data yang seragam dalam jumlah yang besar dan mendistribusikannya ke seluruh bagian organisasi.

3. Berhubungan erat dengan analisis berorientasi objek, perancangan berorientasi objek, user interface berorientasi objek, dan pemrograman berorientasi objek.

Selain keuntungan yang diperoleh dalam menggunakan OOAD seperti yang telah disebutkan di atas, ternyata juga terdapat beberapa kelemahan yaitu (McLeod, 2015):

1. Diperlukan waktu lama untuk memperoleh pengalaman pengembangan.

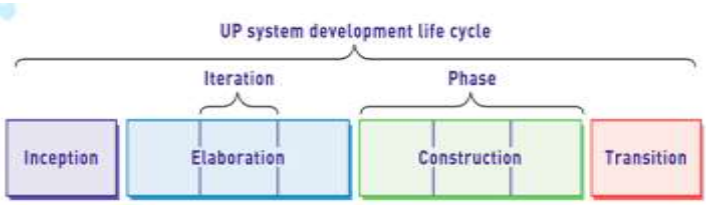
2. Kesulitan metodologi untuk menjelaskan sistem bisnis yang rumit.

3. Kurangnya pilihan peralatan pengembangan yang khusus disesuaikan untuk sistem bisnis.

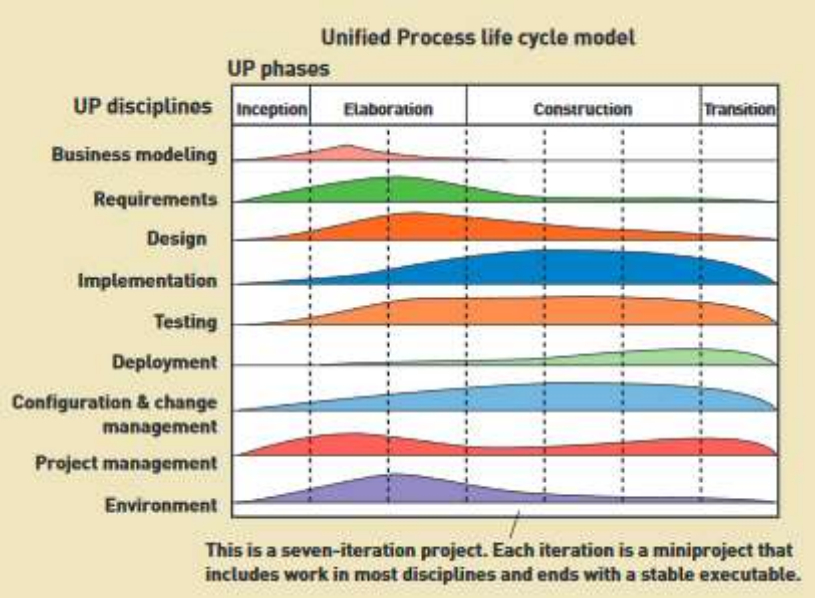
2.7 *Unified Process* (UP).

Unified Process (UP) adalah metodologi pengembangan sistem berorientasi objek yang ditawarkan oleh *Rational Software* yang merupakan bagian dari IBM. Dikembangkan oleh Grady Booch, James Rumbaugh, dan Ivar Jacobson yang merupakan tiga pendiri dibalik keberhasilan *Unified Modelling Laguage* (UML), UP menjelaskan metodologi yang lengkap menggunakan UML sebagai pemodelan sistem yang baru dan lengkap dalam siklus hidup pengembangan sistem adaptif (Satzinger et al., 2009).

2.7.1. Fase Unified Process



Gambar 2.6 *UP system development life cycle*
(Sumber: Satzinger et al., 2009)



Gambar 2.7 *UP system life cycle model*
(Sumber: Satzinger et al., 2009)

Berikut fase-fase pada *unified process* (Satzinger et al., 2009):

1. *Inception Phase*

   Mengembangkan perkiraan visi dari sistem, buat kasus bisnis, menentukan ruang lingkup, dan membuat perkiraan kasar untuk biaya dan jadwal.

2. *Elaboration Phase*

   Mendefinisikan visi, mengidentifikasi dan menggambarkan semua keperluan, menyelesaikan penentuan ruang lingkup, merancang dan mendesain arsitektur inti dan fungsinya, menentuka berbagai risiko yang tinggi, dan membuat perkiraan realistis untuk biaya dan jadwal

3. *Construction Phase*

   Secara berulang menerapkan risiko lebih rendah yang tersisa dan mudah diprediksi, serta mempersiapkan *deployment*.

4. *Transition Phase*

   Menyelesaikan pengujian dan *deployment* sehingga pengguna memiliki sistem yang berfungsi dan siap untuk mendapatkan manfaat seperti yang diharapkan.

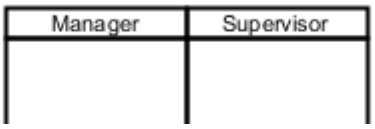
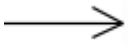
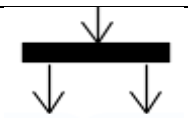
2.8 *Unified Modeling Language* (UML)

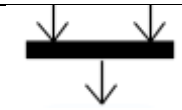
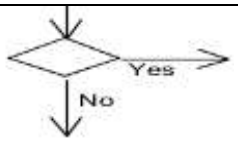
Unified Modeling Language (UML) merupakan set standar dari kronstruksi model dan notasi yang didefinisikan oleh *Object Management Group.* Menggunakan UML memungkinkan analis dan pengguna dapat memahami dan membaca isi diagram. Adapun beberapa contoh digramnya adalah *Activity Diagram, Use Case, Class Diagram, Sequence Diagram,* dan *Package Diagram* (Satzinger et al., 2009).

2.8.1. *Activity Diagram*

Activity Diagram mendiskripsikan segala aktifitas pengguna atau sebuah sistem, orang yang melakukan aktifitas, dan runtutan dari semua aktifitas tersebut (Satzinger et al., 2009). Berikut sismbol-simbol yang digunakan pada *activity diagram*:

Tabel 2. 1 Simbol-simbol *activity diagram* (Sumber: Satzinger et al., 2009)

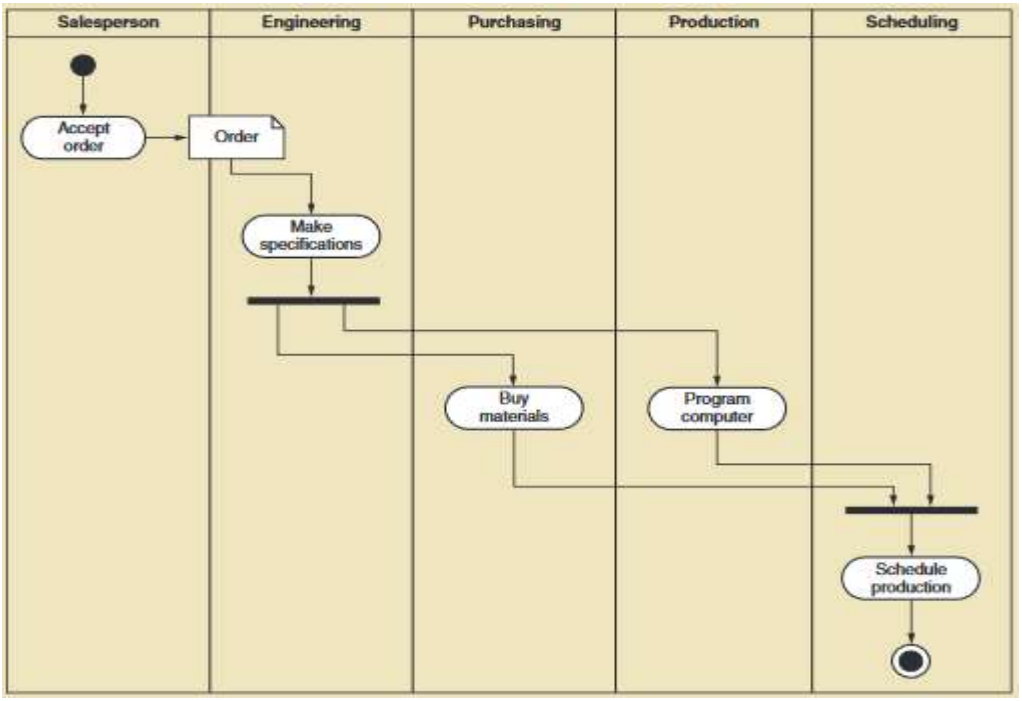
Notasi	Deskripsi
 *Swimlane heading*	Menunjukan aktor atau sistem yang melakukan aktivitas.
 *Start node*	Notasi memulai *activity diagram*.
 *Activity*	Gambaran sebuah aktivitas.
 *Control flow*	Menunjukan alur dari aktivitas.
 *Split*	Sebuah aktivitas dapat dipecah menjadi beberapa aktivitas yang berjalan secara parallel.

Notasi	Deskripsi
 *Join*	Beberapa aktivitas akan bergabung untuk melaksanankan aktivitas selajutnya.
 *Decision*	Sebuah aktivitas dapat mempunyai alternatif pilihan.
 *End node*	Notasi mengakhiri *activity diagram*.

Berikut langkah-langkah membuat *activity diagram* yaitu:

1. Dimulai dari notasi *start node.*
2. Mengidentifikasi pengguna atau sistem sebagai *swimlane*.
3. Menuliskan semua runtutan aktifitas dalam symbol oval.
4. Menghubungkan aktifitas dengan arah panah sehingga terlihat runtutannya.
5. Menggunakan simbol *decision* jika ada pilihan kondisi.
6. Menggunakan *synchronization bar join* atau *split* untuk aktifitas yang bergabung atau bercabang. *Synchronization bar* juga dapat mewakili aktifitas berulang
7. Berakhir dengan notasi *end node*.

Berikut contoh dari *activity diagram* pada proses produksi barang yaitu:

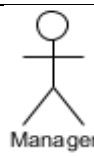


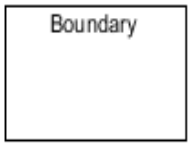
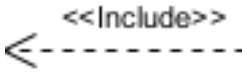
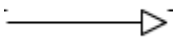
Gambar 2.8 Contoh *activity diagram*
(Sumber: Satzinger et al., 2009)

2.8.2. *Use Case Diagram*

Use case diagram merupakan sebuah aktifitas yang dilakukan sistem, biasanya dalam bentuk respon terhadap *request* pengguna. Teknik yang digunakan untuk identifikasi *use case* adalah teknik event decomposition, dimulai dari identifikasi semua kegiatan bisnis yang mengharuskan sistem mengembalikan respon, setiap kegiatan tersebut akan membentuk *use case* (Satzinger et al., 2009). Berikut sismbol-simbol yang digunakan pada *use case diagram*:

Tabel 2. 2 Simbol-simbol *use case diagram* (Sumber: Satzinger et al., 2009)

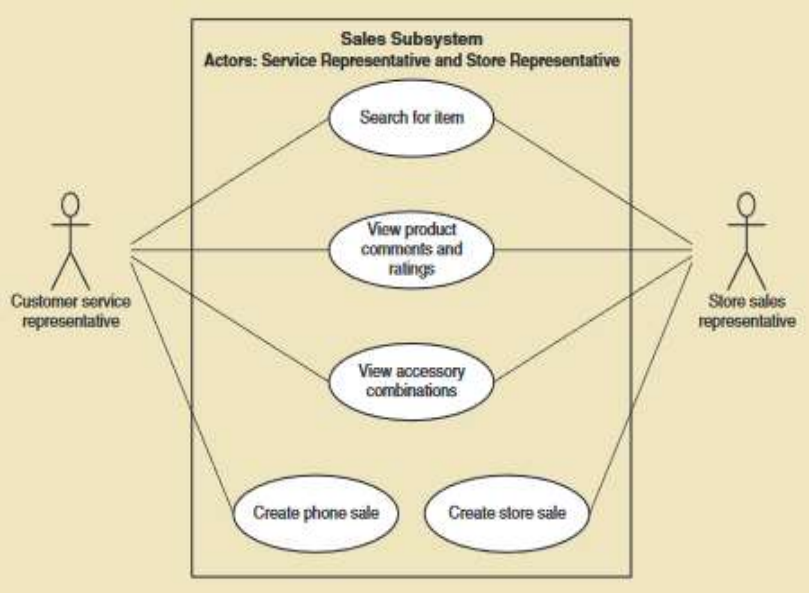
Notasi	Deskripsi
 *Actor*	Orang yang menggunakan sistem.

Notasi	Deskripsi
 *Boundary*	Ruang lingkup otomasi.
 *Usecase*	Kegiatan atau aktifitas yang dapat dilakukan *actor.*
Association relationship	Menghubungkan *actor* dengan *use case.*
 *Include*	Menunjukan sebuah *use case* memerlukan *use case* lain untuk menjalankan fungsinya.
 *Extend*	Menunjukan sebuah *use case* dapat berdiri sendiri walaupun tanpa *use case* tambahan.
 *Generalization*	Hubungan child *use case* ke parent *use case*. Menentukan *child use case* mendapat turunan perilaku dan karakteristik dari *parent use case*

Berikut langkah-langkah membuat *use case diagram* yaitu:

1. Mengidentifikasi semua *stakeholder* dan pengguna yang terlibat sebagai aktor.
2. Menentukan kebutuhan setiap aktor. *Use case* diagram memungkinkan menghasilkan subsistem dengan menggunakan notasi yang diperlukan.
3. Menghubungkan actor dengan *use case* atau antar *use case.*

Berikut contoh dari *use case diagram* pada sub sistem penjualan yaitu:



Gambar 2.9 Contoh *use case diagram*
(Sumber: Satzinger et al., 2009)

2.8.2.1. Use Case Brief Description

Use case brief description merupakan deskripsi singkat tentang aktifitas aktor yang terjadi pada suatu *use case* yang kecil, mudah dipahami, serta dengan *flow* yang normal dan jarang ada kondisi opsional (Satzinger et al., 2009). Cara membuat *use case brief description* cukup dengan mendiskripsikan aktivitas apa saja yang dapat terjadi pada suatu sistem. Berikut contoh dari *use case brief description* yang berkaitan dengan pelanggan:

Use case	Brief use case description
Create customer account	User/actor enters new customer account data, and the system assigns account number, creates a customer record, and creates an account record.
Look up customer	User/actor enters customer account number, and the system retrieves and displays customer and account data.
Process account adjustment	User/actor enters order number, and the system retrieves customer and order data; actor enters adjustment amount, and the system creates a transaction record for the adjustment.

Gambar 2.10 Contoh *use case brief description*
(Sumber: Satzinger et al., 2009)

2.8.2.2. Fully Developed Description

Fully developed description meerupakan metode deskripsi yang paling komplit untu menjelaskan sebuah *use case*. U*se case description* ini dapat memudahkan pemahaman mengenai proses bisnis (Satzinger et al., 2009). Cara membuatnya dengan mendisripsikan secara detail tentang *use case* yang bersangkutan mulai dari nama *use case* hingga kondisi eksepsional. Berikut contoh dari *use case fully developed description* penambahan akun pelanggan:

Use case name:	*Create customer account.*	
Scenario:	Create online customer account.	
Triggering event:	New customer wants to set up account online.	
Brief description:	Online customer creates customer account by entering basic information and then following up with one or more addresses and a credit or debit card.	
Actors:	Customer.	
Related use cases:	Might be invoked by the *Check out shopping cart* use case.	
Stakeholders:	Accounting, Marketing, Sales.	
Preconditions:	Customer account subsystem must be available. Credit/debit authorization services must be available.	
Postconditions:	Customer must be created and saved. One or more Addresses must be created and saved. Credit/debit card information must be validated. Account must be created and saved. Address and Account must be associated with Customer.	
Flow of activities:	**Actor**	**System**
	1. Customer indicates desire to create customer account and enters basic customer information.	1.1 System creates a new customer. 1.2 System prompts for customer addresses.
	2. Customer enters one or more addresses.	2.1 System creates addresses. 2.2 System prompts for credit/debit card.
	3. Customer enters credit/debit card information.	3.1 System creates account. 3.2 System verifies authorization for credit/debit card. 3.3 System associates customer, address, and account. 3.4 System returns valid customer account details.
Exception conditions:	1.1 Basic customer data are incomplete. 2.1 The address isn't valid. 3.2 Credit/debit information isn't valid.	

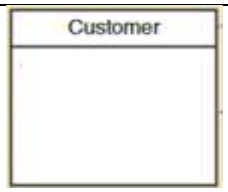
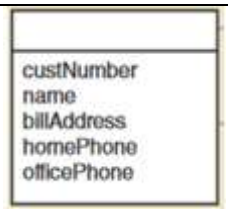
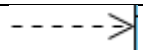
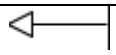
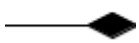
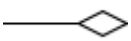
Gambar 2.11 Contoh *fully developed description*
(Sumber: Satzinger et al., 2009)

2.8.3. *Domain Model Class Diagram*

Class merupakan kategori atau klasisfikasi dai kumpulan objek atau benda. Sedangkan *domain class* merupakan *class* yang mendiskipsikan objek dari problem domain. Pada UML *class diagram* digunakan untuk menunjukan *class*

dari objek-objek pada sistem, sehingga *domain model class diagram* adalah *class diagram* yang menunjukan problem domain dari pengguna (Satzinger et al., 2009). Berikut sismbol-simbol yang digunakan pada *domain model class diagram*:

Tabel 2. 3 Simbol-simbol *domain model class diagram*
(Sumber: Satzinger et al., 2009)

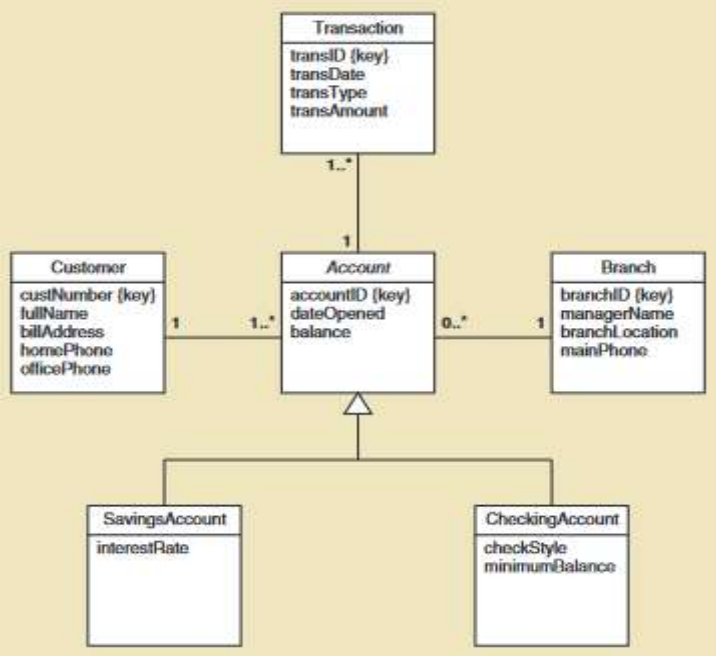
Notasi	**Deskripsi**
 *Class*	Nama dari *class*
 *Attributes*	Nilai atau atribut dari suatu objek dalam *class*
 *Asosiasi*	Hubungan statis antara dua *class* bersama dengan multiplisitas
 *Dependency*	Relasi dimana sebuah *class* membutuhkan *class* lainnya untuk dapat berjalan dalam hal ini dapat berbentuk parameter *object* yang dieksekusi dalam *method class* lainnya
 *Generalization*	Fitur warisan dari konsep berorientasi objek. Dimana child mewarisi atribut dan *method* dari parentnya
 *Composition*	Suatu *class* merupakan bagian utuh dari *class* lainya namun pada hal ini satu bagian *class* tersebut akan sangat bergantung pada keberadaan *class* lainya
 *Aggregation*	Relasi dimana sebuah *class* merupakan bagian utuh dari *class* lainya sering digambarkan dengan kata "*has a*" berarti memiliki
0..1 *Zero or one*	*Optional*, nol atau satu objek
0..* *Zero or more*	*Optional*, nol atau lebih banyak objek
1 *One and one only*	*Mandatory*, tepat satu objek
* *Zero or more alternate*	*Optional*, banyak objek
1..1	*Mandatory*, tepat satu objek

Notasi	Deskripsi
One and one only alternate	
1..* *One or more*	*Mandatory*, satu atau lebih banyak objek

Berikut langkah-langkah membuat *domain model class diagram* yaitu:

1. Mengidentifikasi semua *class* yang muncul.
2. Menentukan atribut dari setiap *class*.
3. Membuat dan menentukan hubungan dan tipe relasi antar *class.*
4. Menambahkan multisiplitas pada relasi *class*.

Berikut contoh *domain model class diagram* pada sistem perbankan yaitu:



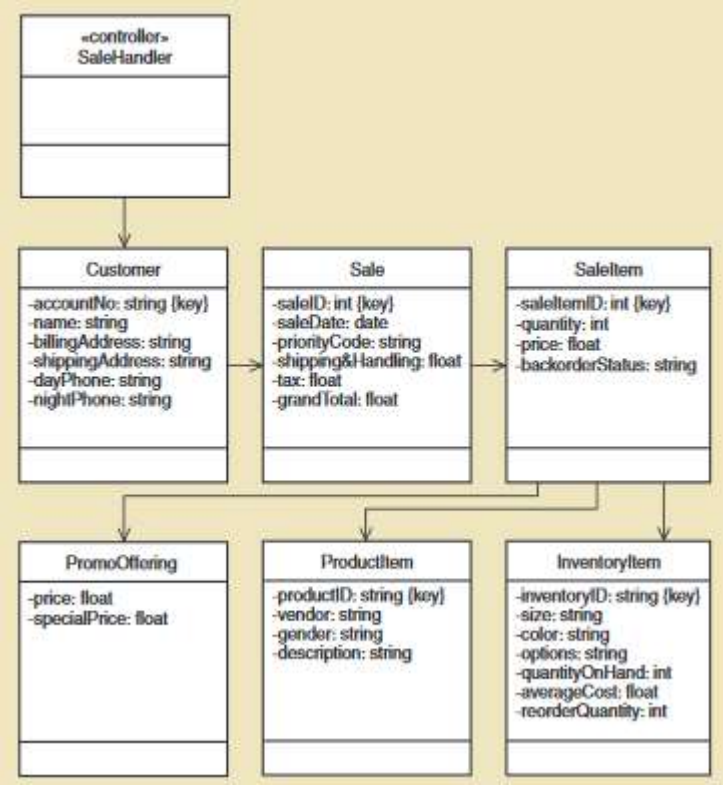
Gambar 2.12 Contoh domain model class diagram
(Sumber: Satzinger et al., 2009)

2.8.3.1. *First-Cut Design Class Diagram*

First Cut Design Class Diagram merupakan perluasan dari *model domain class diagram* dengan dua langkah yaitu (Satzinger et al., 2009):

1. Mengelaborasi atribut-atribut dengan tipe dan nilai informasi inisial.
2. Menambahkan panah navigasi visibilitas.

Berikut contoh *first cut design class diagram* pada sistem penjualan yaitu:



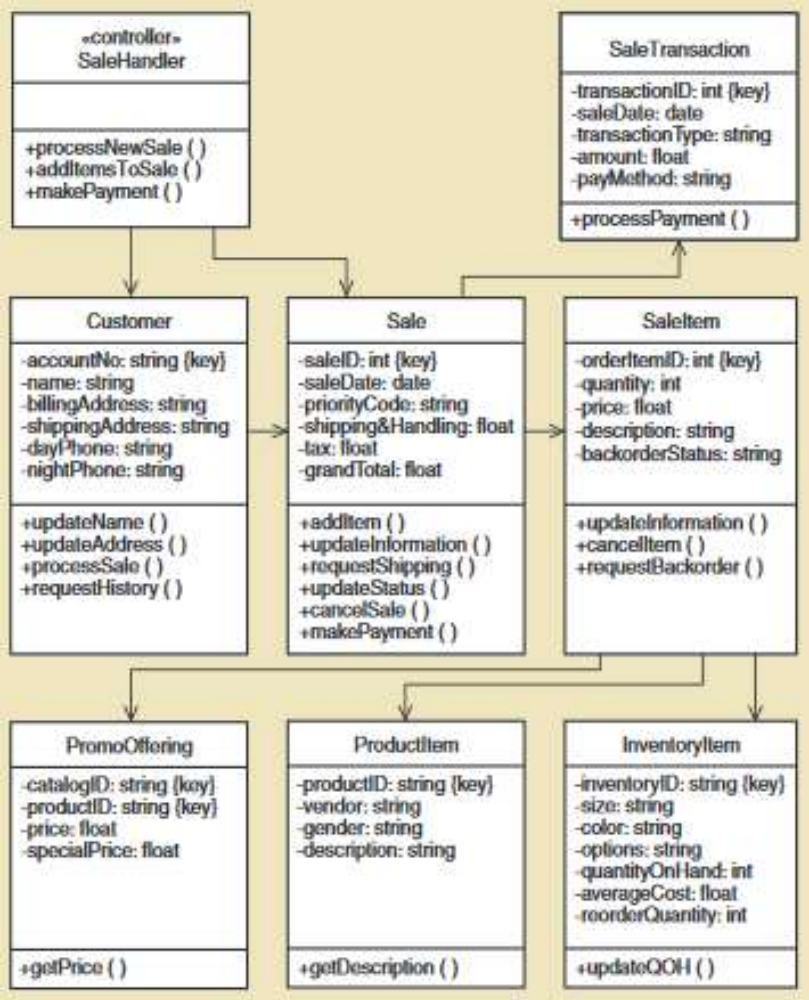
Gambar 2.13 Contoh first-cut design class diagram
(Sumber: Satzinger et al., 2009)

2.8.3.2. *Updated Design Class Diagram*

Updated design class diagram adalah sebuah *class diagram* lanjutan dari *first-cut design class diagram* yang lebih *detail* dalam menjelaskan *input message* yang terdapat pada *first cut sequence diagram*, alur data beserta tipe datanya, dan *input message* yang akan dilaksanakan oleh *use case controller* (Satzinger et al., 2009).

1. Membuat *controller* yang akan menjadi *method* untuk merubah data *class*.
2. Menuliskan detail atribut class *controller* beserta data tipenya.
3. Menghubungkan *class* tersebut dengan arah apanh sesuai *first cut sequence diagram.*

Berikut contoh *updated design class diagram* pada sistem penjualan yaitu:



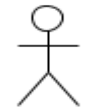
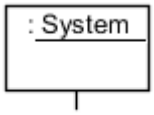
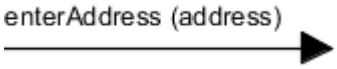
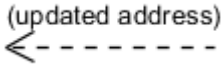
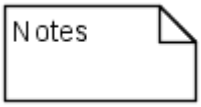
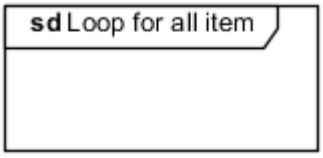
Gambar 2.14 Contoh updated design class diagram
(Sumber: Satzinger et al., 2009)

2.8.4. *System Sequence Diagram*

System sequence diagram digunakan untuk menggambarkan hubungan aktor dan sistem yang menyajikan *flow* dari informasi *input* dan *output* dari sebuah sistem otomasi (Satzinger et al., 2009).

Berikut sismbol-simbol yang digunakan pada *system sequence diagram*:

Tabel 2. 4 Simbol-simbol *system sequence diagram*
(Sumber: Satzinger et al., 2009)

Notasi	Deskripsi
 *Actor*	Aktor eksternal yang berinteraksi dengan sistem.
 *System object*	Objek yang mewakili sistem otomasi.
Lifeline	Menunjukan alur dari *message* dari atas ke bawah.
 *Input message*	Message input dari aktor.
 *Output message*	Hasil *output* atau *return value* dari sistem.
 *Optional note*	Keterangan tambahan untuk menjelaskan sesuatu pada diagram.
 *Repeatability*	Pengulangan untuk suatu kondisi dalam kotak.

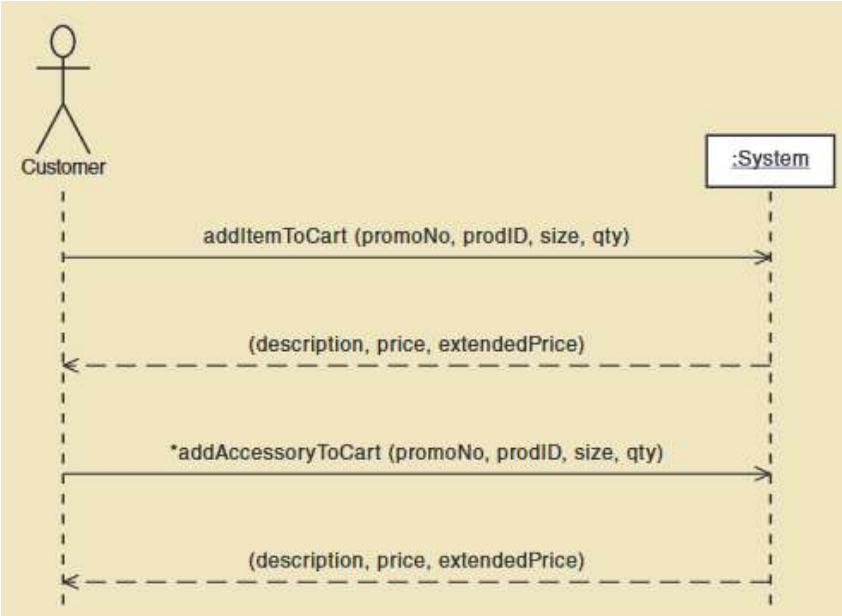
Berikut langkah-langkah membuat *system sequence diagram* yaitu:

1. Aktor dan alur yang digunakan berdasarkan *activity diagram*.
2. Mengidentifikasi pesan *input*, dapat dilihat pada arah panah *activity diagram*.
3. Mendeskripsikan pesan dari aktor eksternal ke sistem menggunakan pesan notasi di atas.

4. Mengidentifikasi dan menambah kondisi input pesan, termasuk iterasi dan kondisi benar atau salah.

5. Setelah *input* diketahui, identifikasi *output* pesan kembalian dari sistem.

Berikut contoh dari *system sequence diagram* pada proses pembelian oleh pelanggan yaitu:



Gambar 2.15 Contoh *system sequence diagram*
(Sumber: Satzinger et al., 2009)

Multi layer digunakan untuk mendukung jaringan *multitier* yang mana *database server* berada dalam satu mesin, dan *busines logic* nya ada pada server lain, dan *user interface* ada di mesin desktop (Satzinger et al., 2009). Adapun tiga lapisan tersebut adalah sebagai berikut:

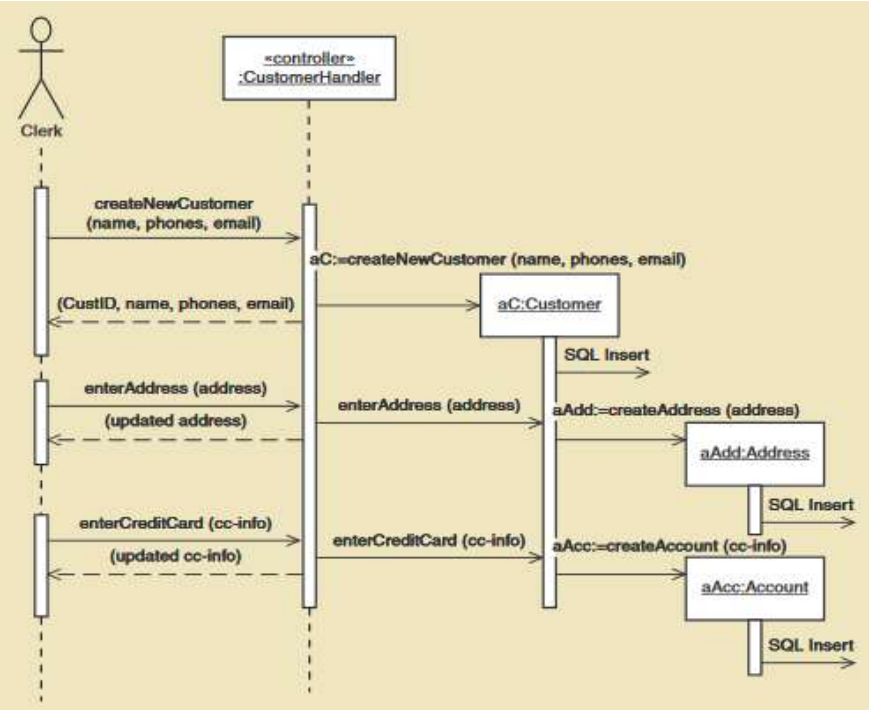
2.8.4.1. *First Cut Sequence Diagram*

First cut diagram merupakan *sequence digram* yang *detail*, menggunakan semua elemen pada sy*stem sequence diagram*. Perbedaannya objek sistem diganti dengan objek internal dan pesan antar sistem (Satzinger et al., 2009).

Berikut langkah-langkah tambahan dalam membuat f*irst cut sequence diagram* yaitu:

1. Mengambil semua pesan *input* dan menentukan pesan internal yang dihasilkan dari *input*
2. Mengidentifikasi kumpulan class yang dipengaruhi oleh pesan.
3. Menyempurnakan komponen tiap pesan, menambahkan iterasi, kondisi benar atau salah, nilai kembalian, dan parameter.

Berikut contoh dari f*irst cut sequence diagram* pada proses penambahan akun pelanggan oleh pegawai yaitu:



Gambar 2.16 Contoh first cut sequence diagram
(Sumber: Satzinger et al., 2009)

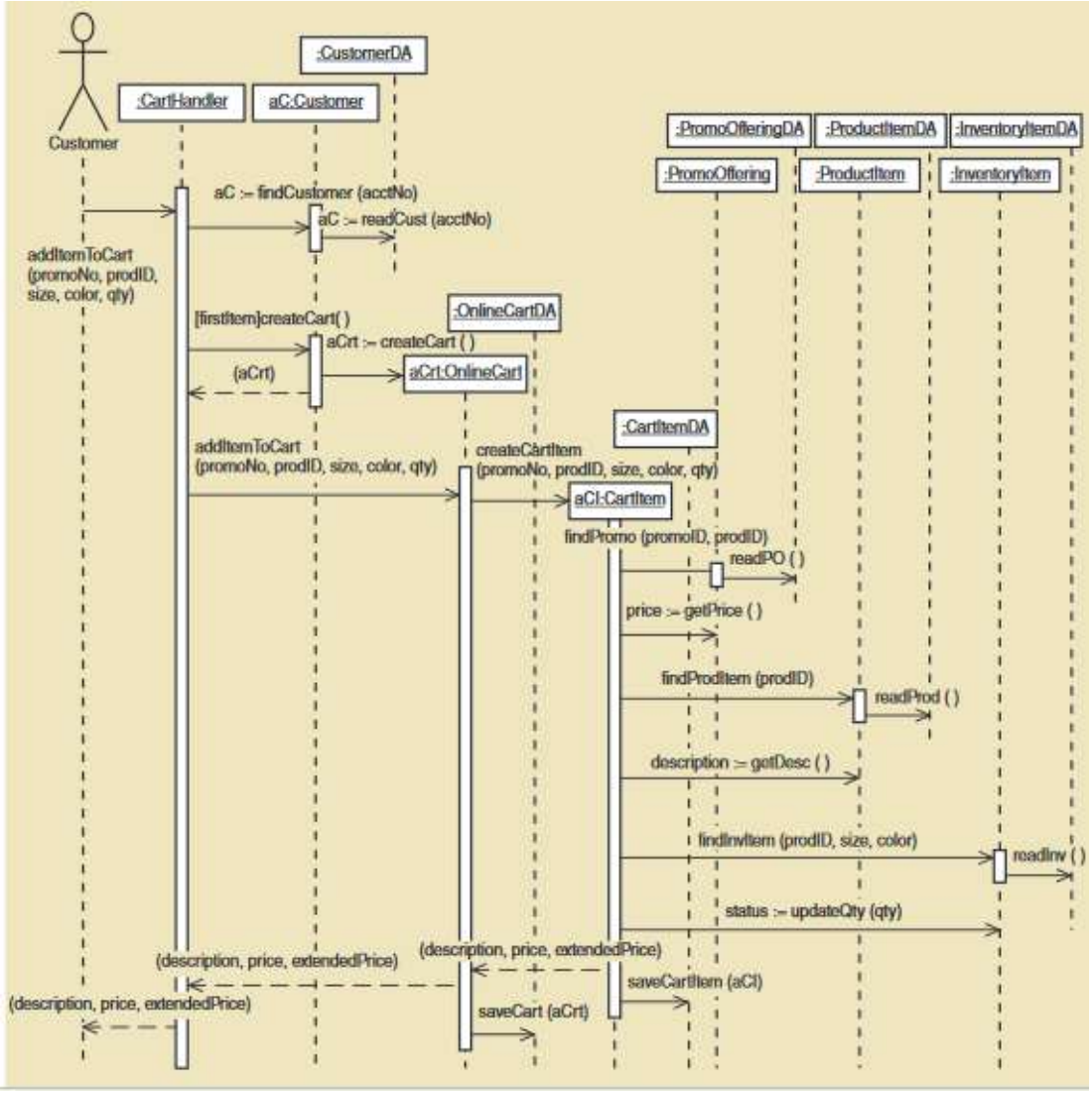
2.8.4.2. *Data Access Layer*

Data access layer diperlukan pada *business logic* yang kompleks dan harus diisolasi dari SQL *statement* yang mengakses ke *database* (Satzinger et al., 2009).

Berikut langkah-langkah tambahan dalam membuat *data access layer sequence diagram* yaitu:

1. Menambahkan *constructor method* setiap *object problem domain*.
2. Menambahkan pengiriman *message* ke objek *data access layer* dan membaca ke *database* intansiasi *problem domain object*.

Berikut contoh dari *data access layer sequence diagram* pada proses pembelian oleh pelanggan yaitu:



Gambar 2.17 Contoh data access layer sequence diagram
(Sumber: Satzinger et al., 2009)

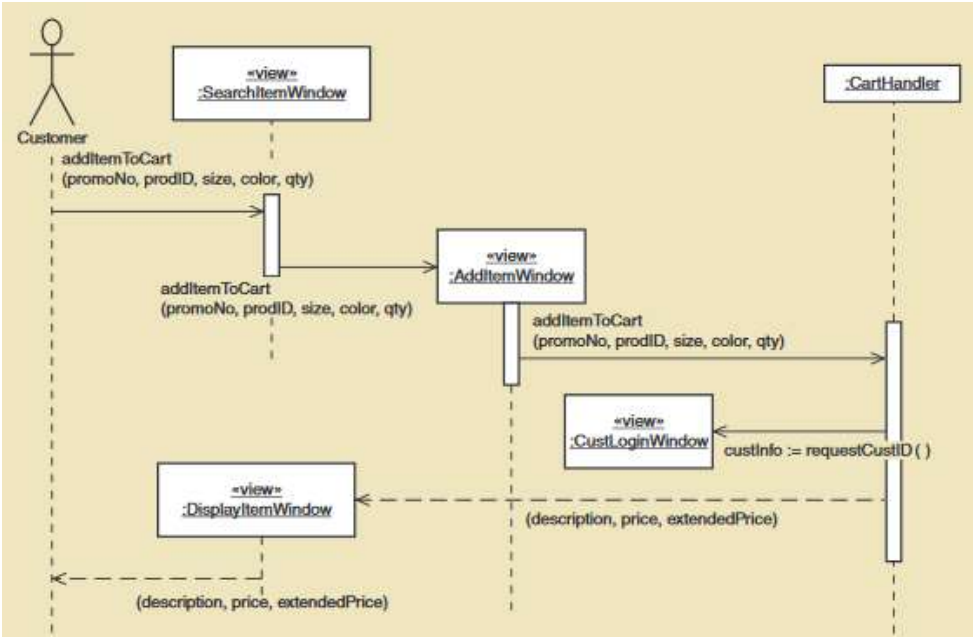
2.8.4.3. *View Layer*

View layer melibatkan interaksi antara pengguna dan komputer dengan *user interface* pada masing-masing *use case* nya.

Berikut langkah-langkah tambahan dalam membuat *view layer sequence diagram* yaitu dengan menambahan komponen user interface. Ada dua sumber sumber input untuk desain view layer yaitu (Satzinger et al., 2009):

1. Desain *user interface.*
2. *First cut sequence diagram* atau *sequence diagram* dengan *data access* yang sudah diidentifikasi.

Berikut contoh dari *view layer sequence diagram* pada proses pembelian oleh pelanggan yaitu:



Gambar 2.18 Contoh view layer sequence diagram
(Sumber: Satzinger et al., 2009)

2.8.5. *Persistent Object*

Persistent object adalah objek yang diingat oleh sistem dan tersedia untuk digunakan dari waktu ke waktu walaupun program sudah berhenti (Satzinger et al., 2009).

Berikut contoh dari *persistent object* katalok produk yaitu:

CatalogProduct : Table

CatalogID	ProductID	Price	SpecialPrice
23	1244	$15.00	$12.00
23	1245	$15.00	$12.00
23	1246	$15.00	$13.00
23	1247	$15.00	$13.00
23	1248	$14.00	$11.20
23	1249	$14.00	$11.20
23	1252	$21.00	$16.80
23	1253	$21.00	$16.40
23	1254	$24.00	$19.20
23	1257	$19.00	$15.20

Record: 11 of 11

Gambar 2.19 Contoh persistent object
(Sumber: Satzinger et al., 2009)

2.8.6. *Package Diagram*

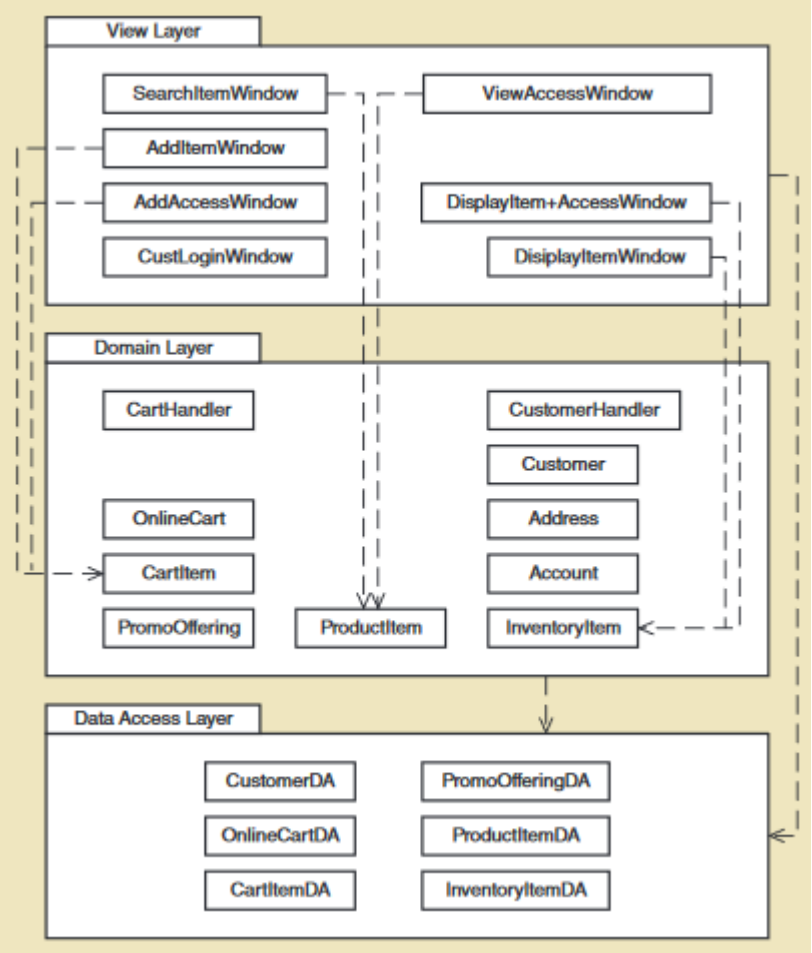
Package diagram menunjukan komponen relasi dan ketergantungan, biasanya digunakan untuk merelasikan *class* atau komponen sistem lain seperti *network nodes* (Satzinger et al., 2009). Berikut sismbol-simbol yang digunakan pada *package diagram*:

Tabel 2. 5 Simbol-simbol *package diagram*
(Sumber: Satzinger et al., 2009)

Notasi	Deskripsi
 *Package*	Menunjukan nama package dalam sistem.
 *Dependency*	Menunjukan relasi antar *package* atau *class* dalam sistem.

Cara membuat *package diagram* yaitu dengan mengekstrak informasi dari desain *class diagram* dan menentukan interaksi diagram untuk setiap *use case*.

Berikut contoh dari *package diagram* pada proses pembelian oleh pelanggan yaitu:



Gambar 2.20 Contoh package diagram
(Sumber: Satzinger et al., 2009)

2.9 *Web* Aplikasi

Web adalah sebuah kumpulan halaman yang diawali dengan halaman muka yang berisikan informasi, iklan serta program aplikasi (Asropudin, 2013).

Aplikasi adalah *software* yang dibuat oleh suatu perusahaan komputer untuk mengerjakan tugas-tugas tertentu, misalnya Ms.World, Ms.Excel (Asropudin, 2013).

Web Aplikasi adalah aplikasi yang dijalankan melalui *browser*. Tidak seperti aplikasi *desktop* yang tradisional, yang mana dijalankan oleh sistem operasi, *web* aplikasi harus diakses melalui *web browser* (Abdul Kadir, 2009).

2.9.1. Hypertext Prepocessor

Hypertext Preprocessor (PHP) adalah bahasa *server-side–scripting* yang menyatu dengan HTML Pemrograman PHP sangat cocok dikembangkan dalam lingkungan web, karena PHP bisa dilekatkan pada *script* HTML (*HyperText Markup Language*) atau sebaliknya. Secara khusus PHP dirancang untuk membentuk aplikasi *web* dinamis. Maksudnya, PHP mampu menghasilkan website yang secara terus-menerus hasilnya bisa berubah-ubah sesuai dengan pola yang diberikan. Hal tersebut tergantung pada permintaan *client browser*-nya (bisa menggunakan *browser* Opera, *Internet Explorer*, Mozzila, dan lainnya). Pada umumnya, pembuatan *web* dinamis berhubungan erat dengan *database* sebagai sumber data yang akan ditampilkan (Arief, 2011).

2.9.2. CodeIgniter

CodeIgniter adalah sebuah *framework* Hypertext Prepocessor (PHP) yang dapat membantu mempercepat *developer* dalam pengembangan aplikasi *web* berbasis PHP dibanding jika menulis semua kode program dari awal. CodeIgniter merupakan PHP *framework* yang menerapkan sistem berbasis MVC (*Model-View-Controller*) yang secara sederhana dapat diartikan bahwa CodeIgniter memisahkan komponen-komponen didalam pengkodean aplikasi berbasis *web*, sehingga diharapkan nantinya lebih mudah untuk dikelola (Hakim, 2010).

2.10 *Mobile Application*

Mobile application merupakan jenis perangkat lunak aplikasi yang dirancang untuk dijalankan di perangkat seluler, seperti komputer *smartphone* atau tablet. Aplikasi seluler sering berfungsi untuk menyediakan layanan serupa bagi pengguna yang diakses di komputer. Aplikasi biasanya kecil, unit perangkat lunak individu dengan fungsi terbatas (Janssen, 2018).

M*obile application* juga memiliki banyak kelebihan seperti kemudahan akses secara *online* maupun *offline*, personalisasi konten, pemanfaatan fitur pada *device*, *push notification*, *update* secara instan, dan lebih interaktif.

Adapun pengembangan *mobile application* mempunyai banyak *platform*, namun saat ini yang paling poluler adalah yang berbasis *platform* Android dan iOS.

2.10.1 Android

Android adalah sistem operasi seluler yang dikembangkan oleh Google. Ini digunakan oleh beberapa *smartphone* dan tablet. Sistem operasi Android didasarkan pada kernel Linux. Tidak seperti Apple iOS, Android adalah *open source*, yang berarti pengembang dapat memodifikasi dan menyesuaikan *Operating Sstem* (OS) untuk setiap ponsel. Oleh karena itu, ponsel berbasis Android yang berbeda sering memiliki antarmuka pengguna *Graphical User Interface* (GUI) yang berbeda meskipun mereka menggunakan OS yang sama.

Pengembang dapat membuat program untuk Android menggunakan perangkat pengembang perangkat lunak Android gratis. Program Android ditulis di Java dan dijalankan melalui *Java Virtual Machine* (JVM) yang dioptimalkan untuk perangkat seluler. Pengguna dapat mengunduh dan memasang aplikasi Android dari Google Play dan lokasi lainnya.

Java adalah nama sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada komputer yang berdiri sendiri (*standalone*) ataupun pada lingkungan jaringan (Shalahuddin, 2010).

2.10.2 iOS

iOS adalah sistem operasi yang dikembangkan oleh perusahaan Apple untuk ponsel iPhone. Awalnya bernama OS iPhone, tetapi diubah namanya menjadi iOS pada bulan Juni, 2009. Kemudian berkembang dan dapat digunakan ke dalam perangkat Apple yang lainnya seperti iPod Touch, Apple TV dan iOS saat ini berjalan pada iPhone, iPod touch, dan iPad. Seperti sistem operasi desktop modern, iOS menggunakan *Graphical User Interface* (GUI).

Adapapun bahasa pemrograman yang digunakan untuk mengembangkan aplikasi pada *platform* iOS yaitu Swift. Swift adalah bahasa pemrograman yang kuat dan intuitif untuk macOS, iOS, watchOS dan tvOS. Menulis kode Swift bersifat interaktif dan menyenangkan, sintaksnya ringkas namun ekspresif, dan Swift menyertakan fitur-fitur modern yang disukai pengembang (Aldhepia, 2015).

2.11 Basis data (*Database*)

Basis data terdiri dari 2 kata, yaitu basis dan data. Basis kurang lebih dapat diartikan sebagai markas atau gudang, tempat bersarang/berkumpul. Sedangkan data adalah representasi fakta dunia nyata yang mewakili suatu objek seperti manusia, barang, konsep, keadaan dan sebagainya yang direkam dalam bentuk angka, huruf, simbol, teks, gambar atau kombinasinya (Fathansyah, 2010).

Basis data sendiri dapat didefinisikan dalam sejumlah sudut pandang, seperti:

1. Himpunan kelompok data (arsip) yang saling berhubungan yang diorganisasi sedemikian rupa agar kelak dapat dimanfaatkan kembali dengan cepat dan mudah.
2. Kumpulan data yang saling berhubungan yang disimpan secara bersama sedemikian rupa dan tanpa pengulangan (redundansasi) yang tidak perlu, untuk memenuhi berbagai kebutuhan.
3. Kumpulan file/tabel/arsip yang saling berhubungan yang disimpan dalam media penyimpanan elektronis.

Basis data sesungguhnya memiliki prinsip kerja dan tujuan yang sama. Prinsip utamanya adalah pengaturan data atau arsip. Tujuan utamanya adalah kemudahan atau kecepatan dalam pengambilan kembali data atau arsip. Perbedaannya hanya terletak pada media penyimpanan yang digunakan.

Pemanfaatan basis data dilakukan untuk memenuhi sejumlah tujuan seperti berikut ini:

1. Kecepatan dan Kemudahan (*Speed*)

   Memungkinkan untuk dapat menyimpan data atau melakukan perubahan/manipulasi terhadap data atau menampilkan kembali data tersebut dengan lebih cepat dan mudah, daripada jika menyimpan data secara manual.

2. Efisiensi Ruang Penyimpanan (*Space*)

   Efisiensi/optimalisasi penggunaan ruang penyimpanan dapat dilakukan, karena dapat melakukan penekanan jumlah redundansi data, baik dengan menerapkan

sejumlah pengkodean atau dengan membuat relasi-relasi antar kelompok data yang saling berhubungan.

3. Keakuratan (*Accuracy*)

   Pengkodean atau pembentukan relasi antar data bersama dengan penerapan aturan/batasan tipe data, domain data, keunikan data, dan sebagainya, yang secara ketat dapat diterapkan dalam sebuah basis data, sangat berguna untuk menekan ketidakakuratan pemasukan/penyimpanan data.

4. Ketersediaan (*Availability*)

   Pertumbuhan data sejalan dengan waktu akan semakin membutuhkan ruang penyimpanan yang besar. Data yang sudah jarang atau bahkan tidak pernah lagi digunakan, dapat diatur untuk dilepaskan dari sistem basis data yang sedang aktif baik dengan cara penghapusan atau dengan memindahkannya ke media penyimpanan *offline*.

5. Kelengkapan (*Completeness*)

   Untuk mengakomodasi kebutuhan kelengkapan data yang semakin berkembang, maka tidak hanya dapat menambahkan *record* data, tetapi juga dapat melakukan perubahan struktur dalam basis data, baik dalam bentuk penambahan objek baru (tabel) atau dengan penambahan *field* pada suatu tabel.

6. Keamanan (*Security*)

   Dapat menentukan pengguna yang boleh menggunakan basis data beserta objek-objek di dalamnya dan menentukan jenis-jenis operasi apa saja yang boleh dilakukan.

7. Kebersamaan Pemakaian (*Sharability*)

   Pemakai basis data seringkali tidak terbatas pada satu pengguna saja atau di satu lokasi saja atau oleh satu sistem/aplikasi saja. Basis data yang dikelola oleh sistem (aplikasi) yang mendukung lingkungan *multiuser* akan dapat memenuhi kebutuhan ini tetapi tetap dengan menjaga/menghindari terhadap munculnya persoalan baru seperti inkonsistensi data atau kondisi *deadlock* (Fathansyah, 2010).

2.11.1. SQL Server Management

SQL Server Management adalah *Relational Database Management System* (RDBMS) yang dirancang untuk aplikasi dengan arsitektur *client*/*server*. Istilah *client*, *server*, dan *client*/*server* dapat digunakan untuk merujuk kepada konsep yang sangat umum atau hal yang spesifik dari perangkat keras atau perangkat lunak. Pada *level* yang sangat umum, sebuah *client* adalah setiap komponen dari sebuah sistem yang meminta layanan atau sumber daya (*resource*) dari komponen sistem lainnya. Sedangkan sebuah *server* adalah setiap komponen sistem yang menyediakan layanan atau sumber daya ke komponen sistem lainnya (Mujiono, 2015).

Kelebihan Microsoft SQL Server:

1. Dengan kemampuannya untuk mengolah data yang besar maka DBMS ini sangat cocok untuk perusahaan mikro, menengah hingga perusahaan besar sekalipun.
2. DBMS jenis memiliki kelebihan dalam mengatur *userdata* serta masing-masing *user* dapat diatur hak aksesnya terhadap pengaksesan *database* oleh *database administrator* (DBA).
3. Mempunyai tingkat keamanan data yang sangat baik.
4. Dapat melakukan *back up*, *recovery* dan rollback data dengan mudah.
5. Mempunyai kelebihan untuk membuat *database mirroring* dan *clustering.*

Kekurangan Microsoft SQL Server:

1. DBMS jenis ini hanya dapat berjalan pada sistem operasi / *platform* windows saja.
2. *Software* ini mempunyai lisensi dari microsoft sehingga pemakaiannya membutuhkan biaya yang cukup mahal.

2.12 Jaringan

Jaringan komputer adalah hubungan dari sejumlah perangkat yang dapat saling berkomunikasi satu sama lain.

2.12.1. Internet

Internet (*Interconneted Network*) adalah jaringan komputer yang menghubungkan antar jaringan secara global, *internet* dapat juga dapat disebut jaringan alam suatu jaringan yang luas (Sibero, 2011).

2.12.2. *Local Area Network*

Local Area Network (LAN) merupakan jaringan komputer terkecil untuk pemakaian pribadi. *Local Area Network* (LAN) memiliki skala jangkauan mencakup 1 KM hingga 10 KM, dalam bentuk koneksi *wired* (kabel), *wireless* (nirkabel), maupun kombinasi keduanya (Putu Agus, 2014).

2.13 Cron Job

Cron adalah penjadwal pekerjaan berbasis waktu dalam sistem operasi mirip Unix (Linux, FreeBSD, Mac OS dan lainnya). Pekerjaan atau tugas ini disebut sebagai Cron Jobs. Daemon adalah program yang berjalan di latar belakang setiap saat, biasanya dimulai oleh sistem. Daemon cron ini bertanggung jawab untuk meluncurkan Cron Jobs ini sesuai jadwal. Jadwal berada dalam file konfigurasi yang berisi semua tugas dan pengatur waktu Cron Jobs terdaftar bernama Crontab.

Cron Jobs digunakan untuk menjadwalkan tugas untuk dijalankan di *server*. Cron Jobs paling sering digunakan untuk mengotomatisasi pemeliharaan atau administrasi sistem, menghapus *cache*, pengiriman *email* laporan berkala serta sikronisasi *database*. Namun, Cron Jobs juga relevan dengan pengembangan aplikasi *web*. Ada banyak situasi ketika aplikasi *web* mungkin memerlukan tugas-tugas tertentu untuk berjalan secara berkala (Guzel, 2016).

2.14 *System Interface*

Sebuah langkah kunci dari desain sistem adalah untuk mengklasifikasi *input* dan *output* setiap *event* sebagai *system interface* atau *user interface*. *System interface* merupakan *input* dan *output* yang diperlukan minimal campur tangan manusia. *System interface* mungkin dapat berupa input yang didapat secara otomatis menggunakan alat khusus seperti *scanner*, pesan elektromagnetik dari atu ke sistem lain, atau transaksi yang terkam oleh sistem lain. Banyak *output* dapat dianggap sebagai *system interface* jika mereka mengutamakan pengiriman pesan atau informasi ke sistem lain, atau menghasilkan laporan, *statement*, atau dokumen untuk

agen eksternal atau aktor tanpa banyak campur tangan manusia seperti pengiriman *statement* kartu kredit tiap akhir bulan (Satzinger,2009).

2.15 *User Interface*

User interface merupakan kumpulan input dan output yang secara langsung melibatkan sistem dangan pengguna. *User interface* dapat digunakan untuk pengguna internal dan eksternal. Desainnya variatif bergantung pada tujuan, karakteristik pengguna, atau karakteristik perangkatnya (Satzinger et al., 2009).

Berikut beberapa konsep yang perlu diperhatikan dalam melakukan desain user interface yaitu

1. *Affordance and visibility*, mudah dilihat dan dimengerti pengguan apa kegunaannya.
2. *Consistency*, fungsi dan desain yang kosisten agar tidak membingungkan.
3. *Shortcut*, *user interface* yang biasa digunakan biasanya dapat menjadi objek yang mengganggu sehingga *user interface* tersebut dapat disembunyikan, *shortcut* biasanya akan digunakan oleh pengguna yang berpengalaman.
4. *Feedback*, respon untuk aksi pengguna seperti *hover* dan bunyi.
5. *Dialogs*, membatu pengguna untuk *flow* tertentu.
6. *Error handling*, pesan penanggulangan jika terjadi *error* sehingga pengguna dapat cepat mengambil tidakan koreksi.
7. *Easy Reversal*, aksi pengguna dapat di batalkan dengan mudah.

Berikut contoh dari *user interface* yaitu:

Gambar 2.21 Contoh user interface
(Sumber: Satzinger et al., 2009)