BAB 2

LANDASAN TEORI

2.1 Sistem Informasi

Sistem Informasi adalah seperangkat komponen yang saling terhubung dalam mengumpulkan, memproses, menyimpan dan menyediakan keluaran informasi yang dibutuhkan untuk menyelesaikan tugas-tugas bisnis (Satzinger, Jackson, Burd, 2016).

Sistem informasi terdiri dari komponen, nilai dan karakteristik yang akan dijelaskan dibawah ini.

2.1.1 Komponen Sistem Informasi

Sistem informasi terdiri atas beberapa sistem informasi, diantaranya (Rainer Jr, Prince, Cegielski, 2016):

1. *Hardware*

   Hardware terdiri dari perangkat seperti *procesor, monitor, keyboard* dan *printer*. Hardware bertugas untuk menerima, memproses dan menampilkan data dan informasi.

2. *Software*

   Software adalah program atau kumpulan program yang memungkinkan *hardware* untuk mengolah data.

3. *Database*

   Database adalah kumpulan *file* atau tabel terkait untuk mengolah data sistem penghubung (*wireline* atau *wireless*) yang memungkinkan komputer yang berbeda untuk saling bertukar sumber daya.

4. *Procedure*

   Procedure adalah instruksi untuk menggabungkan komponen-komponen di atas untuk memproses informasi dan menghasilkan output yang diinginkan.

5. *People*

   People adalah individu yang menggunakan *hardware* dan *software*, berinteraksi dengannya atau memanfaatkan hasilnya.

2.1.2 Nilai Sistem Informasi

Nilai informasi secara langsung membantu pengambil keputusan mencapai tujuan organisasi mereka dengan membedakan data dari informasi dan

menggambarkan karakteristik yang digunakan untuk mengevaluasi kualitas data. Informasi yang berharga dapat membantu orang dalam organisasi mereka melakukan tugas dengan lebih efisien dan efektif (Stair, Reynold, 2014).

2.1.3 Karakteristik Sistem Informasi

Sistem Informasi terdiri atas beberapa karakteristik diantaranya (Romney, Steinbart, 2014):

1. Relevan mengurangi ketidakpastian, meningkatkan pengambilan keputusan, serta menegaskan atau memperbaiki ekspektasi sebelumnya.
2. Reliabel bebas dari kesalahan atau dapat menyajikan kejadian atau akivitas organisasi secara akurat.
3. Lengkap tidak menghilangkan aspek penting dari suatu kejadian atau aktivitas yang diukur.
4. Tepat waktu diberikan pada waktu yang tepat bagi pengambilan keputusan dalam mengambil keputusan.
5. Dapat dipahami disajikan dalam format yang dapat dimengerti dan jelas.
6. Dapat diverifikasi dua orang yang independen dan berpengetahuan di bidangnya, dan masing – masing menghasilkan informasi yang sama.
7. Dapat diakses tersedia untuk pengguna ketika membutuhkannya dan dalam format yang dapat digunakan.

2.2 Manajemen

Manajemen adalah proses perencanaan, pengorganisasian dan penggunaan sumber daya-sumber daya organisasi lainnya agar mencapai tujuan organisasi yang telah ditetapkan (Batlajery,
2016). Manajemen dibagi menjadi beberapa fungsi, yaitu merencanakan, mengkoordinasikan, mengawasi dan mengendalikan kegiatan dalam rangka usaha untuk mencapai tujuan yang diinginkan secara efisien dan efektif. Efektif berarti bahwa tujuan dapat dicapai sesuai dengan perencanaan, sementara efisien berarti bahwa tugas yang ada dilaksanakan secara benar, terorganisir, dan sesuai dengan jadwal.

2.3 Aset

Aset merupakan barang atau benda yang terdiri dari benda yang bersifat bergerak dan benda yang bersifat tidak bergerak yang tercakup dalam kekayaan suatu instansi (Sari & Devitra, 2017).

Aset adalah barang *(thing)* atau sesuatu barang *(anything)* yang mempunyai nilai ekonomi *(economic value)*, nilai komersial *(commercial value)* atau nilai tukar *(exchange value)* yang dimiliki oleh badan usaha, instansi atau individu (perorangan) (Aira, 2014).

2.4 Manajemen Aset

Manajemen Aset pada dasarnya adalah suatu tindakan pengelolaan aset, agar aset tersebut bisa memberikan manfaat yang sebesar-besarnya dengan biaya yang sekecil mungkin dan aset tersebut jangan sampai punah, kecuali memang sebaiknya harus dimusnahkan atau dihapuskan (Soemitro, Suprayitno, 2018).

Manajemen aset memiliki tujuan untuk meningkatkan proses pengambilan keputusan dan untuk mengalokasikan dana aset sebuah instansi sehingga pengembalian investasi yang terbaik diperoleh, manajemen aset mencakup semua proses, alat, dan data yang dibutuhkan untuk mengelola aset secara efektif untuk mencapai tujuan (Aira, 2014).

2.5 *System Development Life Cycle*

System Development Life Cycle (SDLC) adalah kerangka kerja yang mengidentifikasi semua kegiatan yang diperlukan untuk meneliti, membangun, menyebarkan, dan memelihara sistem informasi (Satzinger, Jackson, Burd, 2016). SDLC mencakup semua kegiatan yang diperlukan dalam pengerjaan sistem seperti tahap perencanaan, analisis sistem, desain sistem, pemrograman, pengujian, dan pelatihan, serta kegiatan manajemen proyek lainnya yang diperlukan untuk berhasil menggunakan sistem informasi baru. Berikut adalah enam proses inti SDLC:

1. *Identify the problem or need and obtain approval to proceed with the project*

Pada tahap ini akan ditentukan kebutuhan sistem dan persetujuan untuk melanjutkan pengerjaan proyek.

2. *Plan and monitor the project*

Pada tahap ini akan ditentukan apa yang harus dilakukan, bagaimana melakukannya, dan siapa saja yang akan melakukannya.

3. *Discover the system components that solve the problem or satisfy the need*

Pada tahap ini akan ditentukan apa yang diperlukan.

4. *Design the system components that solve the problem or satisfy the need*

Pada tahap ini akan ditentukan bagaimana cara kerja yang sebenarnya?

5. *Build, test and integrate system components*

Pada tahap ini akan dilakukan banyak pemrograman dan integrasi komponen.

6. *Complete system tests and then deploy the solution*

Pada tahap ini akan ditentukan apakah kebutuhan sudah terpenuhi.

2.6 *Object Oriented Analyst and Design* (OOAD)

2.6.1 *Object Oriented Approach*

Object-oriented approach merupakan metode pengembangan sistem berdasarkan sudut pandang bahwa suatu sistem adalah seperangkat objek yang saling berinteraksi dan bekerja sama dalam menyelesaikan tugas (Satzinger, Jackson, Burd, 2012). *Object-oriented approach* dijabarkan ke dalam beberapa diagram UML seperti *Use Case Diagram*, *use case scenario*, *class diagram*, *Activity Diagram* dan *sequence diagram*.

2.6.2 *Object Oriented Analysis*

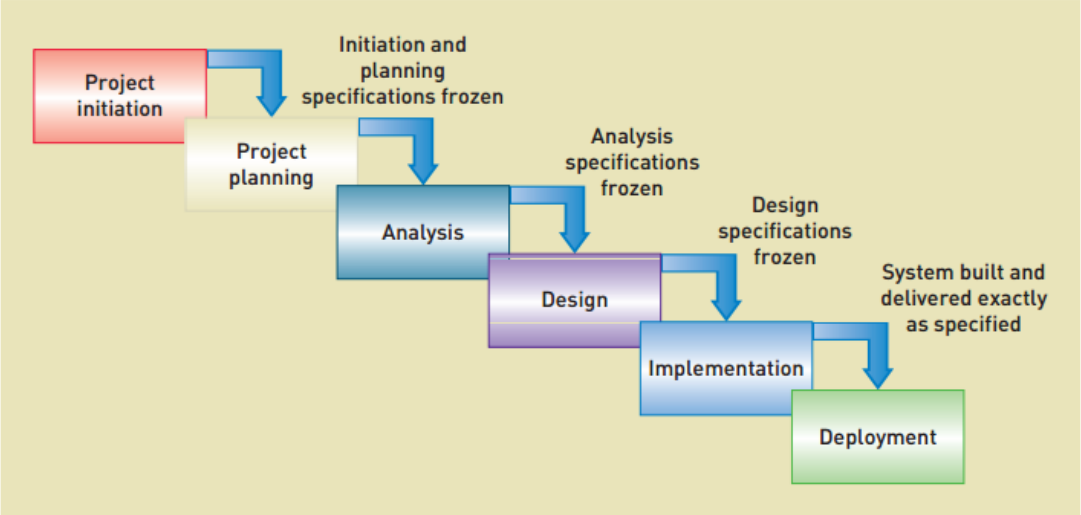
Object-oriented analysis merupakan proses mengidentifikasi dan mendefinisikan sekumpulan object pada sistem dan menentukan bagaimana interaksi user dalam menyelesaikan tugasnya (Satzinger, Jackson, Burd, 2012).

2.6.3 *Object Oriented Design*

Object-oriented design merupakan proses mendefinisikan semua jenis *object* yang akan saling berkomunikasi orang dan perangkat pada *system*. Hal tersebut dilakukan untuk menunjukkan bagaimana objek berinteraksi dalam menyelesaikan tugas dan menyempurnakan definisi setiap jenis objek sehingga dapat diimplementasikan bahasa atau *environment* tertentu (Satzinger, Jackson, Burd, 2012).

2.7 Model *Waterfall*

Model *waterfall* (air terjun) merupakan pendekatan model pengembangan sistem yang menyelesaikan tahap demi tahap secara berurutan (Satzinger, Jackson, Burd, 2012). Kesuksesan dan kegagalan dari satu tahap mempengaruhi kesuksesan dan kegagalan tahap berikutnya, suatu tahap tidak dapat kembali ke tahap sebelumnya, sehingga pada prakteknya model *waterfall* membutuhkan perencanaan yang baik pada setiap tahapannya. Berikut ini pada gambar diperlihatkan setiap tahapan dari model *waterfall*.



Gambar 2.1 Model *Waterfall* (Satzinger, Jackson & Burd, 2016)

Berdasarkan gambar diatas, beberapa tahapan model *waterfall* akan dijelaskan sebagai berikut:

1. *Project Initiation*

Kegiatan awal untuk mengidentifikasi masalah yang terjadi dan analisis kemungkinan penyelesaiannya. Selain itu, pada tahap ini juga mencakup persetujuan untuk mengembangkan sistem baru.

2. *Project Planning*

Kegiatan untuk mengembangkan rencana sumber daya, anggaran, jadwal proyek serta memetakan struktur keseluruhan proyek. Hal tersebut dilakukan untuk mengetahui apakah proyek tersebut layak untuk dilanjutkan pengembangannya.

3. *Analysis*

Kegiatan untuk menemukan dan memahami masalah dan kebutuhan secara detail, maksudnya adalah mencari tahu dengan tepat apa yang harus dilakukan sistem untuk mendukung proses bisnis.

4. *Design*

Kegiatan untuk menyusun dan mengkonfigurasi komponen-komponen dari sistem yang baru. Rancangan komponen tersebut dibuat berdasarkan kebutuhan-kebutuhan yang telah dikumpulkan sebelumnya untuk mengembangkan struktur dan algoritma program dari sistem baru.

5. *Implementation*

Tahapan ini mencakup pemrograman yang dikembangkan dari rancangan yang dilakukan sebelumnya. Setelah program selesai, maka program akan dibagi menjadi

beberapa unit dan dilakukan pengujian untuk mengetahui fungsi yang berkaitan dengan unit tersebut.

6. *Deployment*

Tahapan ini mencakup pemasangan sistem yang telah selesai ke dalam sistem operasi atau perangkat keras pengguna.

Penelitian ini hanya akan dilakukan sampai pada tahap *Design*. Tahap *Implementation* dan *Deployment* tidak akan dilakukan.

2.8 *Unified Modelling Language* (UML)

Unified Modeling Language (UML) adalah model *constructs* dan notasi yang didefinisikan oleh *Object Management Group* (OMG), organisasi standar untuk pengembangan sistem. Dengan menggunakan UML *analysts* dan *end user* dapat menggambarkan dan memahami berbagai jenis diagram yang digunakan dalam proyek pengembangan sistem (Satzinger, Jackson, & Burd, 2016). Diagram UML terdiri atas *Activity Diagram*, *Use Case Diagram*, *class digram*, *sequence diagram*, *state machine diagram*.

2.8.1 *Activity Diagram*

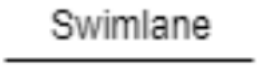
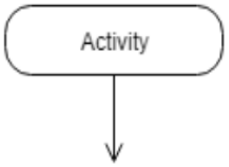
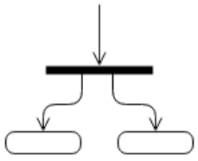
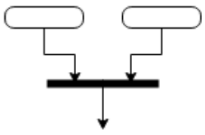
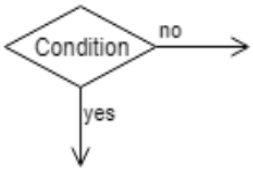
Activity Diagram merupakan salah satu diagram UML yang menggambarkan aktivitas pengguna (atau sistem), aktor atau komponen yang melakukan berbagai aktivitas, dan juga urutan aliran aktivitas yang berjalan (Satzinger, Jackson, & Burd, 2016).

Berikut aturan dalam membuat sebuah *Activity Diagram* (Satzinger, Jackson, & Burd, 2016):

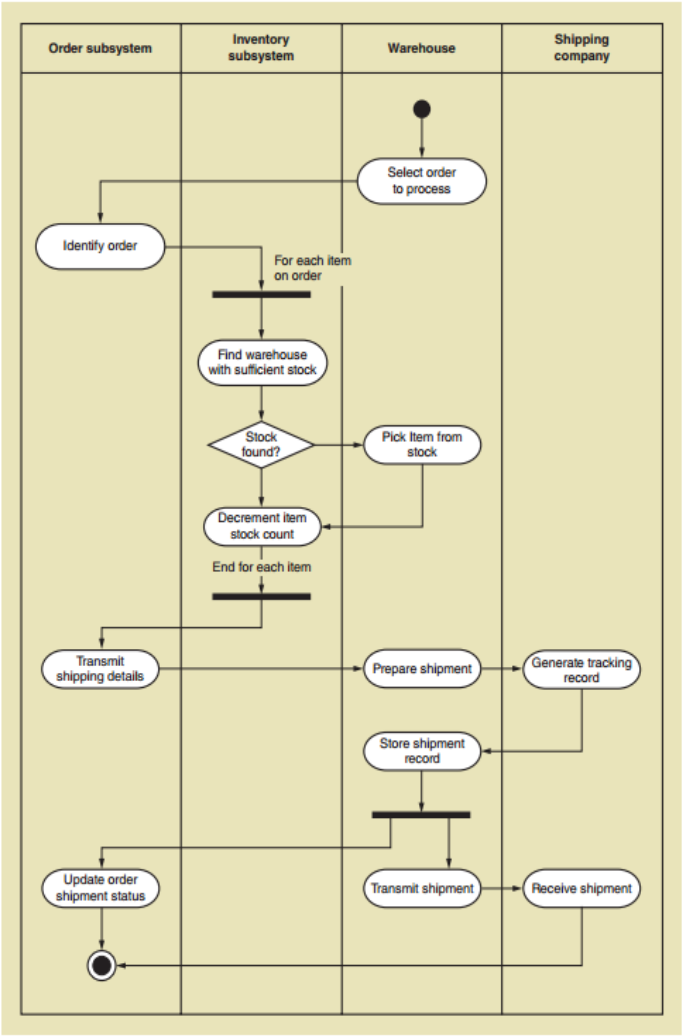
1. Mengidentifikasi agen untuk menciptakan *swimlane* yang sesuai
2. Mengikuti berbagai langkah alur kerja dan membuat *starting Activity* yang sesuai untuk kegiatan tersebut.
3. Menghubungkan *starting Activity* dengan panah yang menunjukkan alur kerja.
4. Menggunakan notasi *decision Activity* untuk mewakili suatu situasi.
5. Menggunakan *synchronize bar* untuk jalur paralel (situasi kedua jalur diambil). *Synchronize bar* juga dapat digunakan untuk merepresentasikan perulangan, seperti "*do while*".

Notasi dari *Activity Diagram* dapat dilihat pada tabel dibawah ini.

Tabel 2.1 Notasi *Activity Diagram* (Satzinger, Jackson & Burd, 2016)

Notasi	Deskripsi
	*Swimlane heading* Komponen yang membagi aktivitas ke dalam beberapa kelompok dimana dalam setiap aktivitas ditunjukkan agen yang menjalankan aktivitas tersebut.
	*Starting activity (Pseudo)* Tanda untuk memulai sebuah aktivitas
	*Transition Arrow* Menampilkan rangkaian eksekusi yang menghubungkan setiap proses
	*Ending Activity (Pseudo)* Tanda untuk mengakhiri sebuah aktivitas
	*Activity* Mewakili aktivitas yang dijalankan
	*Synchronization Bar (Split)* Jalur yang membagi *activity* ke dalam beberapa *activity*
	*Synchronization Bar (Split)* Menyatukan jalur beberapa *activity* ke dalam satu *activity* yang sama.
	*Decision Activity* Symbol yang menandakan harus memilih salah satu jalur untuk melanjutkan aktivitas dan memastikan berjalan pada satu jalur.

Contoh dari *Activity Diagram* dapat dilihat pada gambar di bawah ini yang menunjukkan *Activity Diagram* untuk transaksi *order fulfillment*.



Gambar 2.2 Contoh *Activity Diagram* untuk transaksi *order fulfillment* (Satzinger, Jackson, & Burd, 2016)

2.8.2 *Use Case Diagram*

Use Case Diagram merupakan model UML yang menggambarkan hubungan antara *use case* dan aktor. *Use Case* sendiri merupakan aktivitas yang dilakukan yang berhubungan dengan *user* dan sistem (Satzinger, Jackson, & Burd, 2016).

Berikut merupakan aturan dalam pembuatan *Use Case Diagram* (Satzinger,

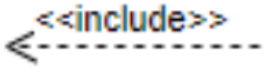
Jackson, & Burd, 2016).

1. Identifikasi semua *stakeholder* dan *User* yang akan mendapatkan manfaat dengan memiliki *Use Case Diagram*.

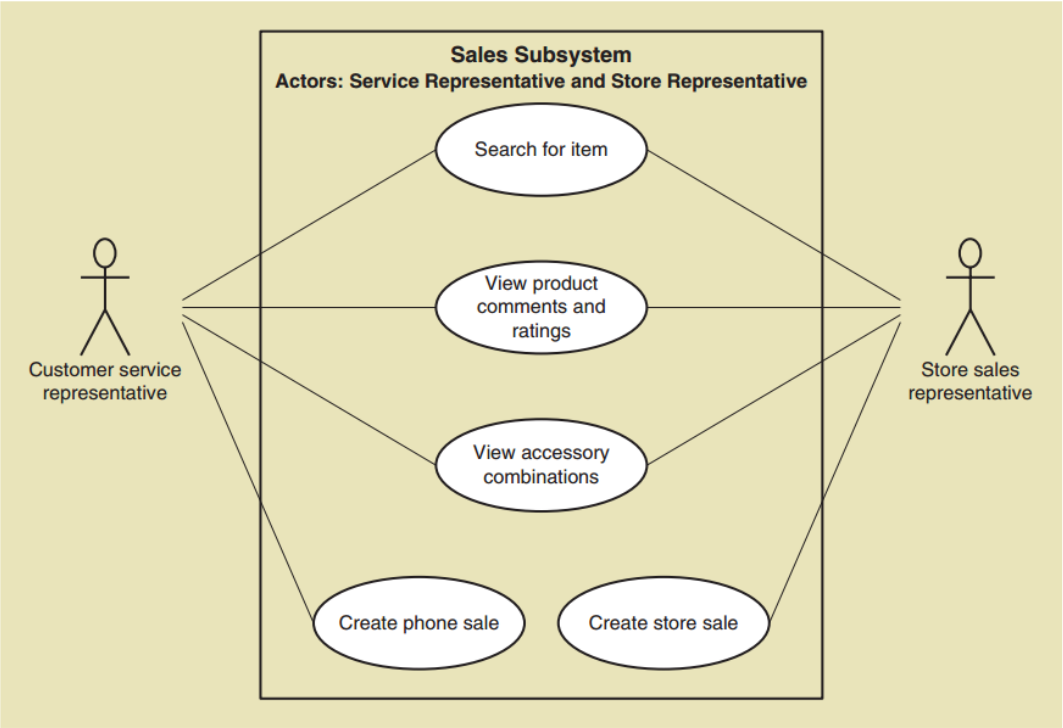
2. Menentukan kebutuhan setiap *stakeholder* dan *User* dalam penggunaan *Use Case Diagram*. Biasanya setiap *Use Case* dapat diproduksi untuk setiap sub sistem, setiap tipe *User*, dan *Use Case* dengan hubungan *include*.
3. Setiap kebutuhan komunikasi, pilih *Use Case* dan *actor* untuk menampilkan dan menggambarkan *Use Case Diagram*.
4. Memberi nama dengan teliti dari setiap *Use Case Diagram*.

Notasi dari *Use Case Diagram* dapat dilihat pada tabel dibawah ini.

Tabel 2.2 Notasi *Use Case Diagram* (Satzinger, Jackson, & Burd, 2016)

Notasi	**Deskripsi**
	*Actor* Komponen yang mewakilkan pengguna yang menjalankan/berinteraksi dengan sistem. *Actor* selalu berada di luar batas automation boundary. Dalam penggunaannya, *actor* tidak selalu berupa orang, akan tetpai bias menjadi sistem atau perangkat lain yang menerima layanan sistem.
	*Use Case* Komponen yang mewakili aktivitas yang berhubungan dengan *user* dan sistem
	*Automation Boundary* Menyatakan perbatasan antara bagian sistem yang terkomputerisasi dan orang yang mengoperasikannya tetapi keduanya masih menjadi bagian dari keseluruhan sistem. Simbol ini berisi kumpulan use case dan garis relasi yang menghubungkan antar use case.
	*Include Relationship* Hubungan antara Use Case dimana satu Use Case secara stereotip termasuk dalam penggunaan lainnya.

Contoh dari *Use Case Diagram* dapat dilihat pada gambar dibawah ini yang menunjukkan *Use Case Diagram* untuk *subSystem sales*, dimana terdapat dua aktor, yaitu *customer service representative* dan *store representative*.



Gambar 2.3 Contoh *Use Case Diagram* untuk *subSystem sales* (Satzinger, Jackson & Burd, 2016)

2.8.3 *Use Case Description*

Use Case Description adalah model tekstual yang menggambarkan dan menjelaskan semua proses detail untuk sebuah *use case*. Tujuan *Use Case Description* adalah untuk mendokumentasikan interaksi antara pengguna dan sistem dalam melakukan aktivitas tertentu (Satzinger, Jackson, & Burd, 2016). *Use Case Description* terbagi atas 2, yaitu *Brief Use Case Description* dan *Fully Developed Use Case Description*.

1. *Brief Use Case Description*

Brief Use Case Description merupakan deskripsi singkat yang terdiri dari 1 kalimat untuk menjelaskan mengenai *use case* (Satzinger, Jackson, & Burd, 2016). Berikut ini merupakan contoh *Brief Use Case Description* yang menjelaskan *Use Case Create customer account*, *look up customer*, dan *process account adjustment*.

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 2.4 Contoh *Brief Use Case Description* (Satzinger, Jackson, & Burd, 2016)

2. *Fully Developed Use Case Description*

Fully Devloped Use Case Description adalah metode paling formal untuk mendokumentasikan sebuah *use case*. Salah satu kesulitan bagi seorang *software developer* adalah menterjemahkan bisnis proses yang *user* inginkan, oleh Karena itu metode ini dapat membantu *software developer* dalam mengatasi hal tersebut (Satzinger, Jackson, & Burd, 2016). Berikut ini merupakan contoh *full developed Use Case Description* yang menjelaskan *useCase Create customer account*.

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 2.5 *Full Developed Use Case Description* (Satzinger, Jackson, & Burd, 2016)

2.8.4 *Domain Model Class Diagram*

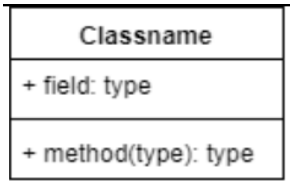
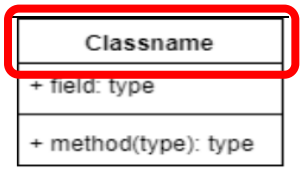
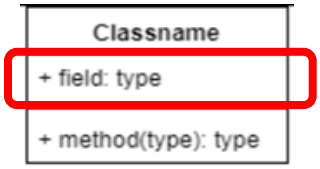
Domain class diagram merupakan sebuah *Class Diagram* yang mencakup kelas-kelas dari *problem domain*. *Class diagram* sendiri adalah diagram yang terdiri dari kelas-kelas (misalnya sekumpulan objek) dan hubungan antar kelas (Satzinger, Jackson, & Burd, 2016).

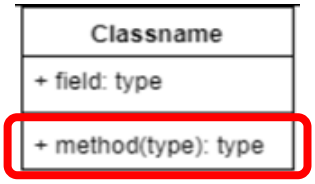
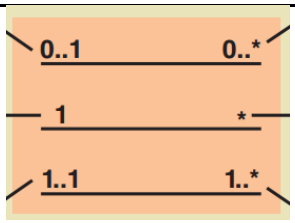
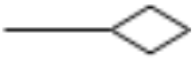
Berikut merupakan aturan dalam pembuatan *Domain Model Class Diagram* (Satzinger, Jackson, & Burd, 2016):

1. *Class diagram* terbagi atas 3 tingkatan, yang pertama adalah nama *class* dan terletak di bagian atas, yang kedua adalah nama atribut dan terletak di tengah, yang terakhir adalah daftar nama *method* dan terletak di bagian bawah.
2. Nama class, atribut dan method menggunakan camelCase dengan huruf awal menggunakan huruf capital untuk nama class dan huruf kecil untuk nama atribut dan method. Ketiga penamaan tersebut tidak diperbolehkan mengandung spasi, spasi diganti dengan tanda "_".
3. *Class diagram* digambar dengan menunjukkan *class* dan hubungan antar *class* (*Association class*).

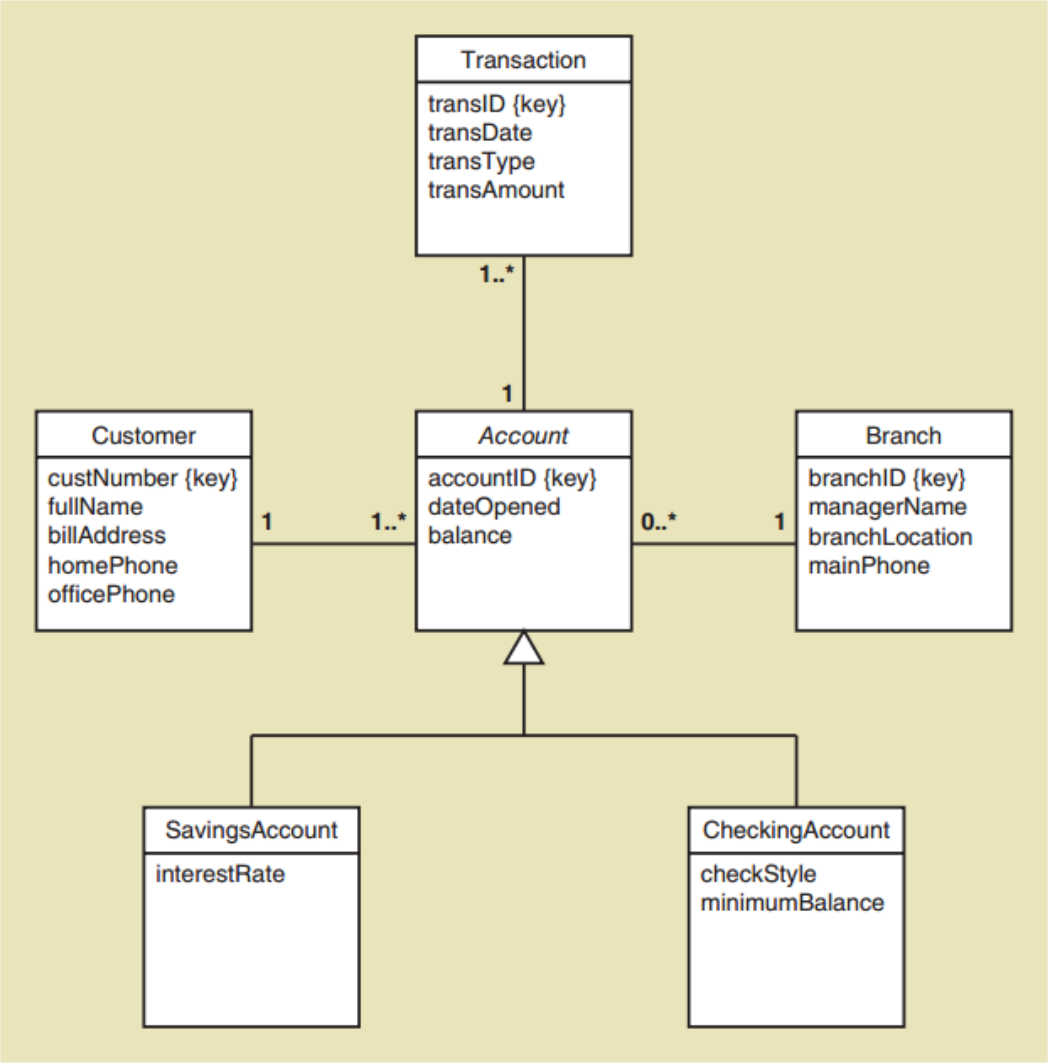
Notasi dari *Domain Class Diagram* dapat dilihat pada tabel dibawah ini.

Tabel 2.3 Notasi *Domain Class Diagram* (Satzinger, Jackson, & Burd, 2016)

Notasi	Deskripsi
	*Domain Class* Berbentuk persegi untuk kategori atau klasifikasi yang digunakan untuk menggambarkan kumpulan objek Setiap objek termasuk dalam class.
	*Class name* Nama dari *domain class* yang dituliskan menggunakan *camelCase* yang diawali dengan huruf kapital
	*Attribute* Semua objek dalam *class* memiliki atribut yang ditulis dengan *camelCase* dan diawali dengan huruf kecil

Notasi	Deskripsi
	*Method* Mewakili tindakan atau aksi yang dapat dilakukan oleh sebuah objek dalam *class*.
	*Association Class* Assosiasi adalah hubungan antara objek yang berarti bahwa sebuah object "menggunakan" object yang lain. Terbagi atas 6, yaitu: *Zero or one (optional), Zero or more (optional), One and only one (mandatory), ero or more alternate (optional), One and only one alternate (mandatory), One or more (mandatory)*
	*Generalization/specialization* Hubungan jenis hirarki dimana *class* bawahan merupakan himpunan dari *object class superior* dan merupakan hirarki *inheritance*.
	*Aggregation* Mengacu pada hubungan whole-part antara aggregate (whole) dan komponennya yang bagian-bagiannya dapat dipisah. Hubungan antar *class* yang menyatakan hubungan *"has-a."*
	*Composition* Mengacu pada hubungan *whole-part* yang lebih kuat, bagian bagiannya jika dikaitkan tidak dapat dipisahkan. Hubungan antar *class* yang menyatakan hubungan *"part-of."*

Berikut ini merupakan contoh dari *domain class diagram* yang menjelaskan hubungan *class transaction* dengan *class customer, sale, account, branch* dan *class* lainnya dengan berbagai macam *association* yang menghubungkan objek-objek tersebut.



Gambar 2.6 Contoh *Domain Class Diagram* (Satzinger, Jackson, & Burd, 2016)

Dalam *domain class diagram* ada yang disebut dengan *first class diagram dan updated class diagram* yang akan dijelaskan dibawah ini.

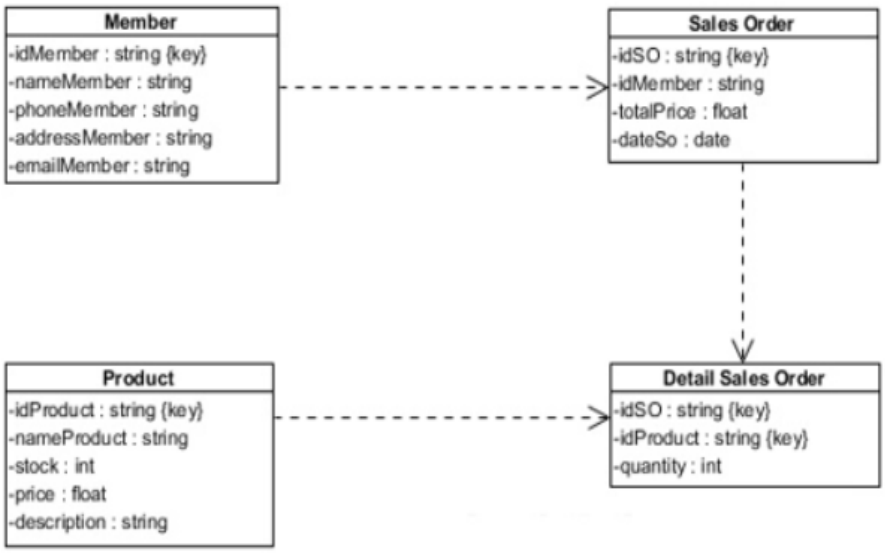
2.8.4.1 *First-Cut Design Class Diagram*

First-Cut Design Class Diagram adalah sebuah class diagram yang lebih menjelaskan mengenai alur data beserta tipe datanya. Dalam penggambaran proses desain, kita dapat memulai dengan merancang first-cut design class diagram berdasarkan domain class diagram. Pada perancangan ini, first-cut design class diagram tidak memerlukan method list (Satzinger, Jackson, & Burd, 2012).

Berikut merupakan aturan dalam pembuatan *First Class Diagram* (Satzinger, Jackson, & Burd, 2016).

1. Pertama, kita akan menguraikan attributes dengan tipe informasi
2. Kedua, kita akan mengidentifikasi *classes* mana yang akan terlibat dan *classes* mana yang membutuhkan *navigation visibility* untuk *classes*.
   a. Kita mengidentifikasi *classes* apa yang akan muncul pada saat dibutuhkan dalam menjalankan *use case*. Contoh: pada *use case Create Sales Order*, *class* yang dibutuhkan adalah *Sales Order class*.
   b. Kita menentukan *classes* lain apa yang diperlukan berdasarkan informasi apa yang dibutuhkan. Contoh: pada *use case Create Sales Order*, *classes* lain yang diperlukan adalah *Member class* dan *Product class*. Karna kita membutuhkan informasi member dan *product*.
3. Lalu kita akan membuat beberapa *logical decisions* mengenai *navigation visibility*. Dalam merancang *first-cut design class diagram* tidak lagi menggunakan *Association Relationship* yang terdapat pada domain class diagram, tetapi akan menggunakan *Navigation Visibility*.

Berikut ini merupakan contoh *First Class Diagram* yang menjelaskan *member, sales order, product, detail SO*.



Gambar 2.7 Contoh *First Cut Class Diagram* (Satzinger, Jackson, & Burd, 2016)

2.8.4.2 *Updated Design Class Diagram*

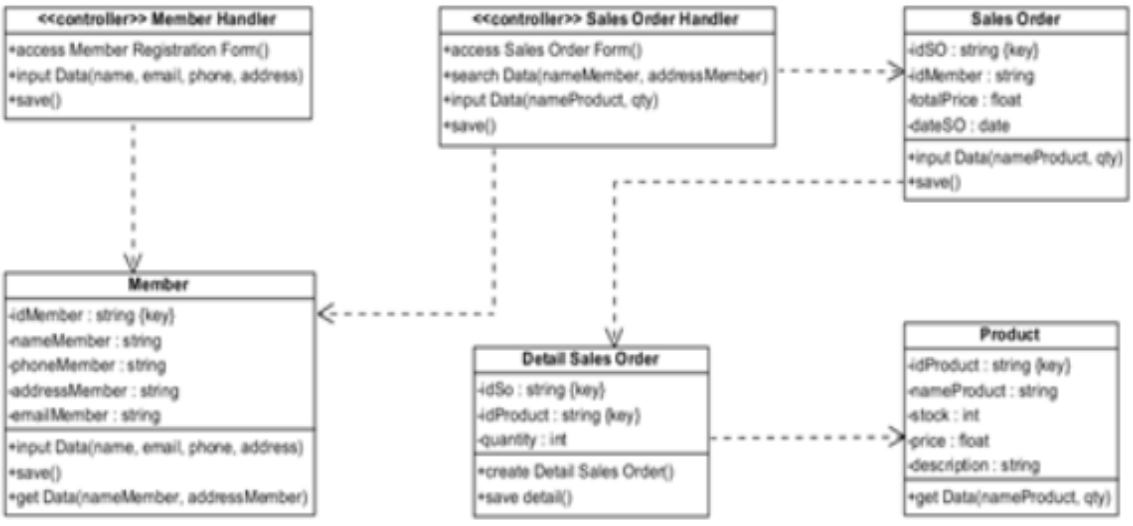
Updated Design Class Diagram adalah sebuah *class diagram* lanjutan dari *first-cut design class diagram* yang lebih *detail* dalam menjelaskan *input*

message yang terdapat pada *First Cut sequence diagram*, alur data beserta tipe datanya, dan *input message* yang akan dilaksanakan oleh *use case controller*. Dalam membuat *updated design class diagram*, kita dapat melihat *first-cut design class diagram* dan *First Cut sequence diagram* yang telah kita buat sebelumnya (Satzinger, Jackson, & Burd, 2012).

Berikut ini langkah-langkah penting yang harus dicermati, yaitu (Satzinger, Jackson, & Burd, 2012):

1. Semua input *message* pada *First Cut design* diagram dibutuhkan sebagai *method* pada *destination object*.
2. Lalu menjabarkan semua *input message* ke dalam semua *class* yang merupakan *destination object* dari *input message*. Semua *input message* akan menjadi *method information list* pada *updated design class diagram*.
3. Perhatikan *navigation arrow* yang terdapat pada *First Cut diagram*, sehingga kita perlu meng-*update navigation arrow* (arah panah) pada *first-cut design class diagram*. Hal tersebut sebagai *progress update* dari *desain class*.
4. Pada *updated design class diagram* terdapat tambahan *class* yang akan dijabarkan, yaitu *controller class* pada setiap *use case* yang terdapat pada *First Cut diagram*.
5. Pada *controller class*, kita hanya perlu menjabarkan *method information list* dan *navigation arrow* dari *controller class* ke *class* yang berhubungan.

Berikut ini merupakan contoh dari *Updated Design Class Diagram* dari *First Cut Design Class Diagram* sebelumnya:



Gambar 2.8 Contoh *Updated Class Diagram* (Satzinger, Jackson, & Burd, 2016)

2.8.5 *System Sequence Diagram*

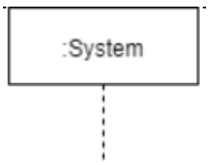
System Sequence Diagram (SSD) merupakan Diagram yang menampilkan urutan pesan antara aktor eksternal dan sistem dalam satu *Use Case* atau scenario *Use Case*. Diagram ini digunakan untuk mendeskripsikan arus dari informasi yang masuk dan keluar; mengidentifikasi interaksi antara actor dan sistem (Satzinger, Jackson, & Burd, 2016).

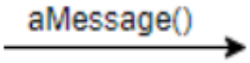
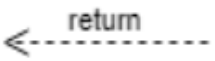
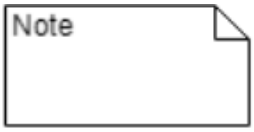
Berikut merupakan aturan dalam pembuatan SSD (Satzinger, Jackson, & Burd, 2016).

1. Mengidentifikasi pesan *input*. Setiap adanya interaksi pesan dari actor ke sistem diperlukan adanya data *input* yang di dalamnya terdapat *message*.
2. Menggambarkan *message* dari eksternal *actor* ke sistem dengan mengunakan notasi *message*. Nama *message* yang dimasukkan mendeskripsikan layanan yang diminta *actor* kepada sistem.
3. Mengidentifikasi dan menambahkan kondisi khusus pada *input message, true* atau *false*.
4. Mengidentifikasi dan menambahkan pesan *output* yang dihasilkan. Terdapat dua opsi untuk menampilkan informasi pengembalian, yaitu sebagai pesan dari pengembalian itu sendiri atau sebagai pesan pengembalian pesan secara terpisah yang ditandai dengan garis putus-putus.

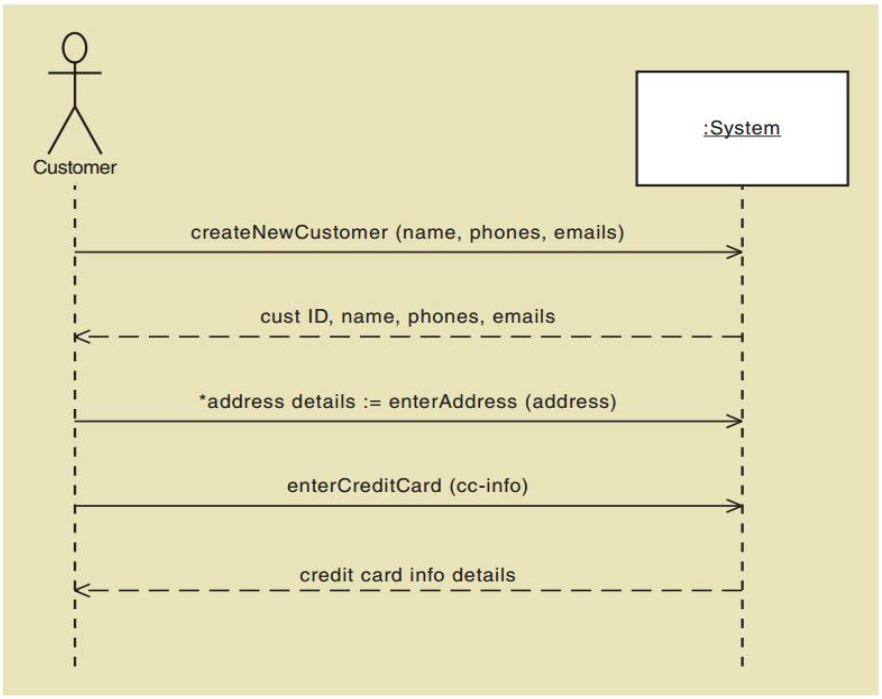
Notasi dari *System Sequence Diagram* dapat dilihat pada tabel dibawah ini:

Tabel 2.4 Notasi *System Sequence Diagram* (Satzinger, Jackson, & Burd, 2016)

Notasi	Deskripsi
	*Actor* Dalam SSD, aktor lebih ditekankan untuk berinteraksi dengan sistem dengan memasukkan data *input* dan menerima *output*.
	*Object* Simbol yang berlabelkan *:System* adalah semua yang mewakili sistem yang berjalan.

Notasi	Deskripsi
	Object Lifeline Garis vertikal dibawah object pada SSD yang menunjukkan siklus hidup/ jalannya sebuah objek.
	*Message* Panah dengan label yang diberikan untuk menjelaskan tujuan dari setiap Data input yang dikirim. Nama *message* harus menggunakan kata kerja.
	*Return* Tanda panah garis putus-putus yang menunjukkan respon atau jawaban dari *message* yang dikirim
	*Note* *Note* bersifat opsional untuk menambahkan sesuatu dalam suatu diagram.
	*Loop Frame* Label "loop" berisi teks deskripsi untuk mengontrol perilaku *message*. Semua yang masuk dalam kotak merupakan kondisi yang dilakukan secara berulang.

Berikut ini menunjukkan contoh dari *System Sequence Diagram* untuk proses *Create new customer*, dimana *customer* mengakses fungsi *Create* dan menerima *feedback* dari *input*-an data, menyimpan kembali data dan sistem menampilkan data yang telah disimpan.



Gambar 2.9 Contoh *System Sequence Diagram* (Satzinger, Jackson, & Burd, 2016)

Setelah membuat SSD, diperlukan desain yang lebih rinci untuk mengidentifikasi *class* yang diperlukan untuk sistem baru dan metode untuk masing-masing class tersebut dengan menghasilkan detail dari *Sequence Diagram* atau satu set dari *Class Responsibility Collaboration* (CRC) untuk setiap *Use Case* atau scenario *Use Case* (Satzinger, Jackson, & Burd, 2016).

2.8.5.1 *First Cut Sequence Diagram*

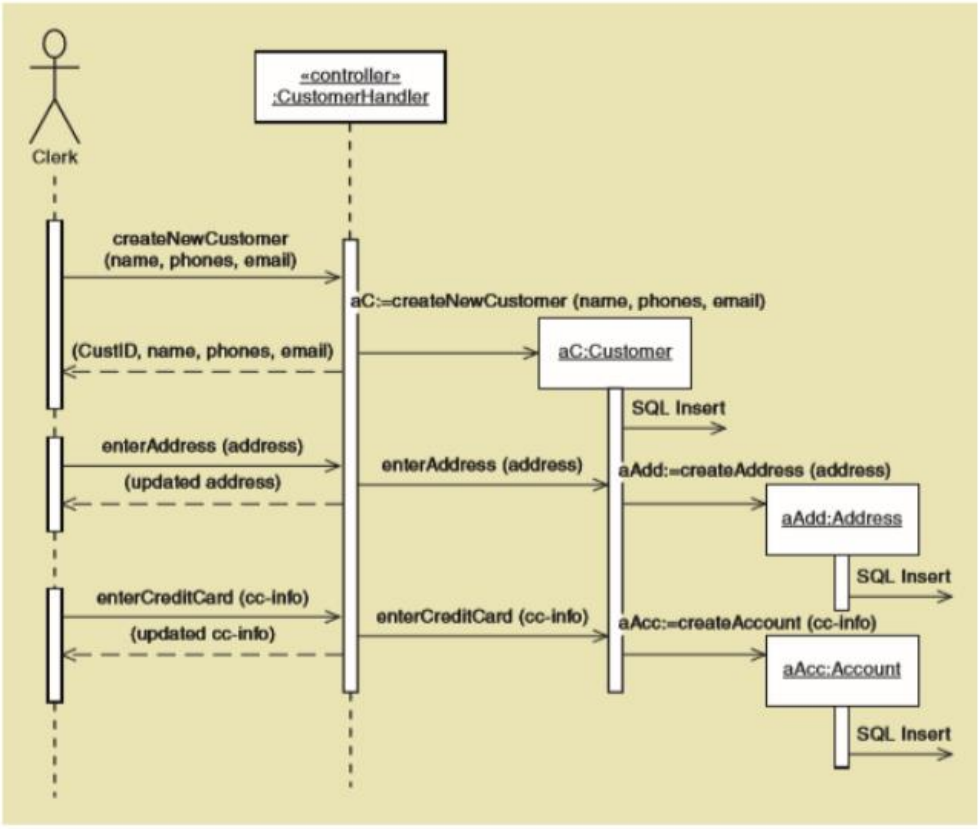
First Cut Class Diagram adalah detail dari *Sequence Diagram* menggunakan semua elemen yang digunakan SSD. Perbedaanya hanya pada objek: *System* yang diganti dengan semua objek internal yang berkolaborasi dengan pesan yang dikirim ke satu sama lain unuk melaksanakan *Use Case* atau *scenario Use Case*. *First-Cut* ini digunakan untuk mengembangkan desain untuk *Use Case* atau *scenario Use Case* menggunakan *Sequence Diagram* (Satzinger, Jackson, & Burd, 2016).

Berikut merupakan aturan dalam pembuatan *First Cut Sequence Diagram* (Satzinger, Jackson, & Burd, 2016).

1. Fokus dalam mengidentifikasi objek internal yang saling berhubungan dan pesan yang mereka kirimkan untuk menunjukkan *use case scenario*.
2. Lihat *problem domain* dan tentukan *class* mana yang memerlukan*use case* tersebut.

3. Tentukan pesan yang harus dikirimkan antara objek, termasuk objek mana yang harus menjadi sumber dan tujuan dari setiap pesan.
4. Keputusan tentang pesan apa yang harus dan objek mana yang terlibat adalah berdasarkan prinsip atau rancangan sebelumnya yang telah dideskripsikan.
5. Setiap pesan yang diinput dievaluasi kembali untuk menentukan pesan dan *problem domain class* apa yang sesuai dan harus dalam melengkapi proses input *Request*. Hal ini membantu dalam mengembangkan *First Cut sequence diagram*, yang mana hanya mencakup *problem domain class*.

Berikut merupakan contoh dari *First-Cut Sequence Diagram*, terlihat bahwa *object internal* yang berhubungan satu sama lain untuk melaksanakan *useCase Create new customer* dijabarkan lebih detail.



Gambar 2.10 Contoh *First-Cut Sequence Diagram* (Satzinger, Jackson, & Burd, 2016)

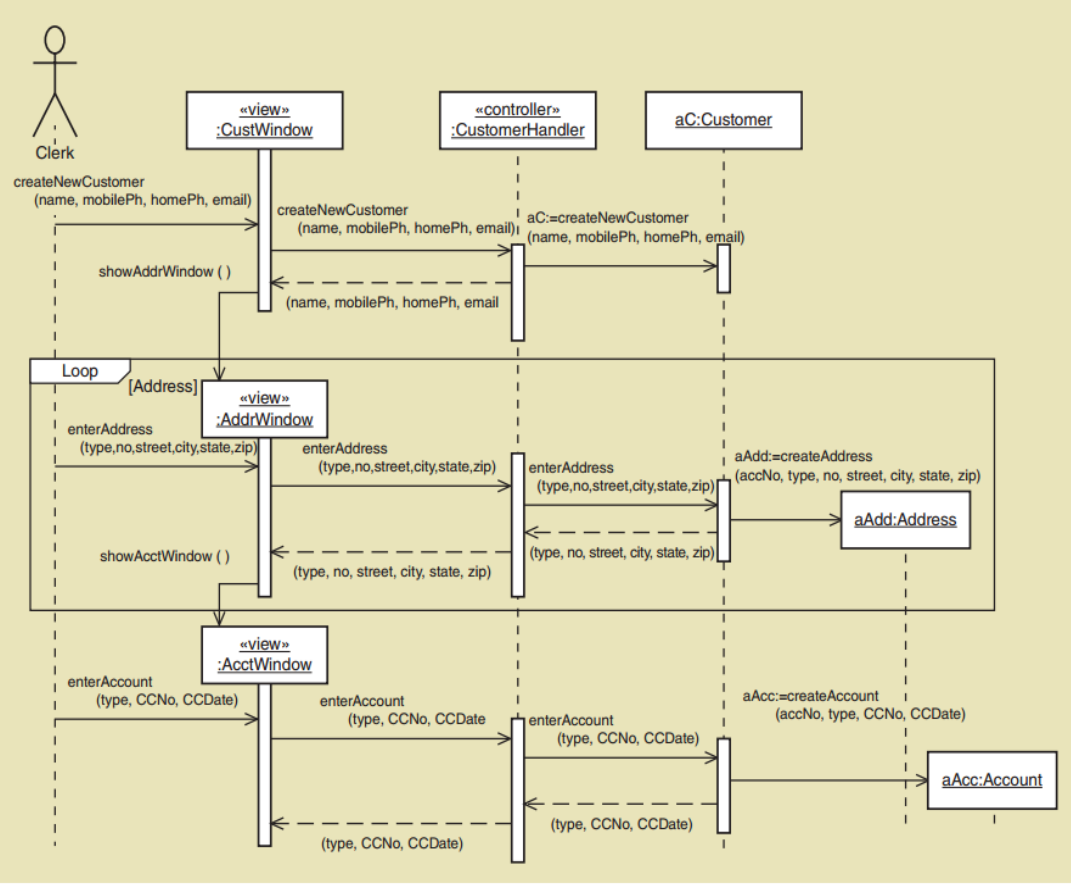
2.8.5.2 *Multilayer Sequence Diagram*

Multilayer Sequence Diagram adalah tahap berikutnya dari *First Cut Sequence Diagram*. Dalam *Multilayer Sequence Diagram* terdapat dua bagian yang dapat dikembangkan, yaitu *Data Access Layer* dan *View layer* untuk mengetahui

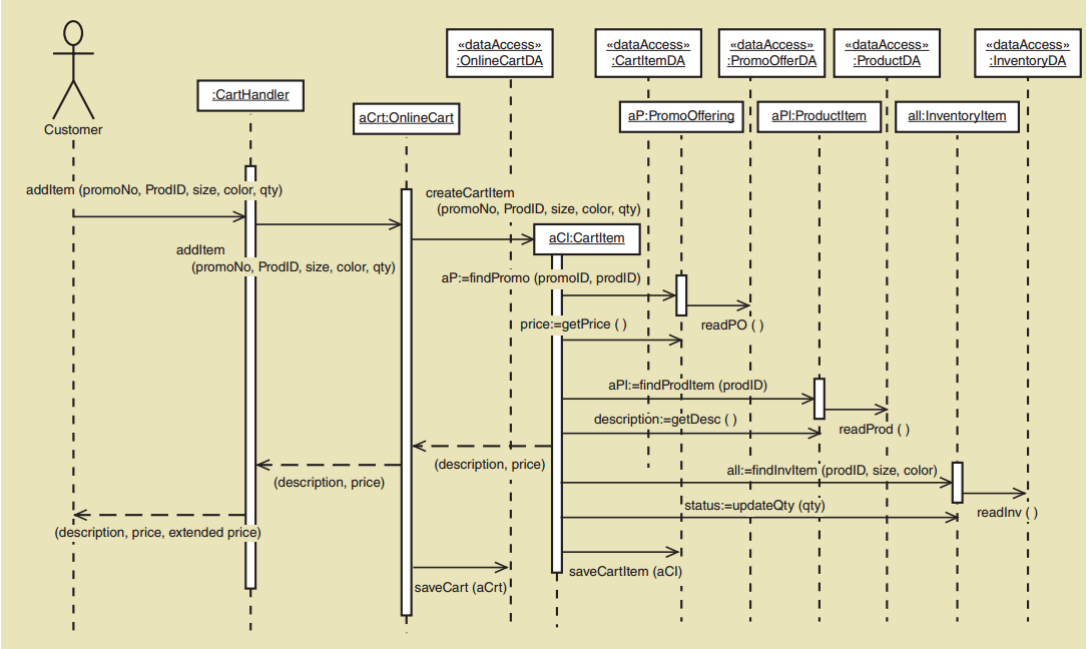
gambaran total dan identifikasi kebutuhan yang dibutuhkan oleh *developer* (Satzinger, Jackson, & Burd, 2016).

1. *View layer* digunakan untuk merepresentasikan layar *input* dan *output* pada aplikasi.
2. *Data Access Layer* diperlukan jika logika bisnis cukup kompleks dan harus diisolasi dari pernyataan SQL yang mengakses *database*.

Berikut ini merupakan contoh dari *Sequence Diagram View layer* dan *Data Access Layer*, menunjukkan proses yang lebih detail lagi, seperti ada representasi tampilan layar *input* dan *output*, akses ke *handler*, dititik mana terjadi perubahan, dan rangkaian proses lainnya.



Gambar 2.11 Contoh *View Layer* (Satzinger, Jackson, & Burd, 2016)



Gambar 2.12 Contoh *Data Access Layer* (Satzinger, Jackson, & Burd, 2016)

2.8.6 *Package Diagram*

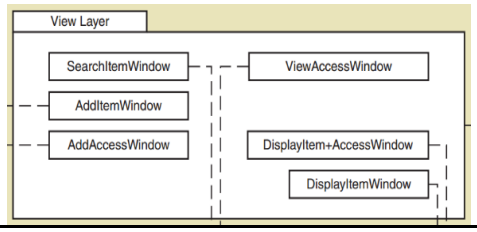
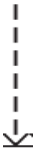
Package diagram adalah diagram UML yang setingkat lebih tinggi yang memungkinkan desainer untuk menghubungkan *class* dari grup yang saling berkaitan. Terkadang desainer perlu mencatat perbedaan dan persamaan relasi dalam *layer* yang berbeda mungkin memisahkan atau mengelompokkan objek berdasarkan lingkungan proses pendistribusiannya. Informasi seperti ini dapat dilihat dengan menampilkan setiap *layer* sebagai *package* yang terpisah (Satzinger, Jackson, & Burd, 2016).

Berikut merupakan aturan dalam pembuatan *Package Diagram* (Satzinger, Jackson, & Burd, 2016).

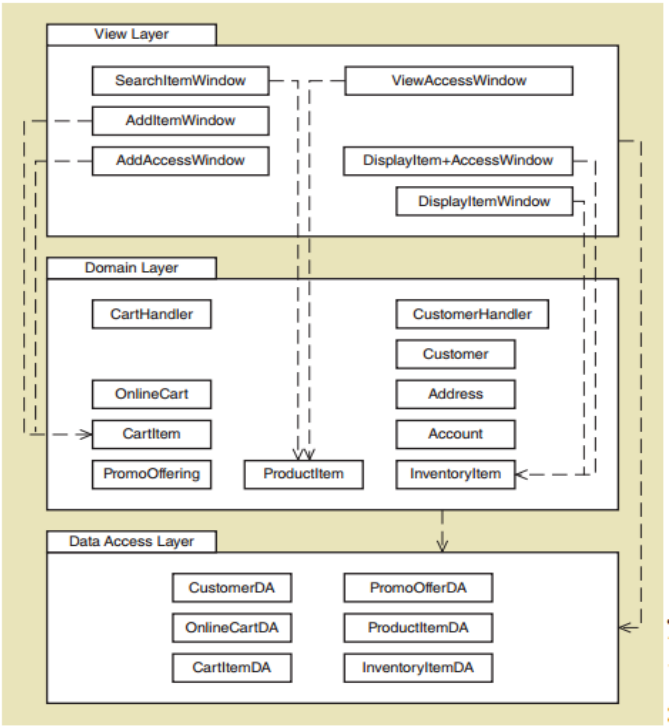
1. *Class* ditempatkan di dalam *package* yang sesuai berdasarkan pada *layer* tempat mereka berada.
2. Mengambil inti sari informasi dari desain *class diagram* dan interaksi digram untuk setiap *use case*
3. Gunakan *dependency relationship* untuk menghubungkan satu *layer* ke *layer* yang lain dalam diagram.

Notasi dari *Package Diagram* dapat dilihat pada tabel dibawah ini.

Tabel 2.5 Notasi *Package Diagram* (Satzinger, Jackson, & Burd, 2016)

Notasi	**Deskripsi**
	*Layer* Tempat bagi *class* yang sudah di-*package* dan ditempat sesuai dengan *layer*.
	*Dependency relationship* Penghubung antar *layer* dalam diagram

Berikut ini adalah contoh *Package Diagram*, dimana setiap *class* ditempatkan pada *layer* tempat mereka berada. Pada diagram terlihat bahwa *layer* dibagi menjadi beberapa *layer* yang berbeda, sesuai interaksi mereka dengan diagram.

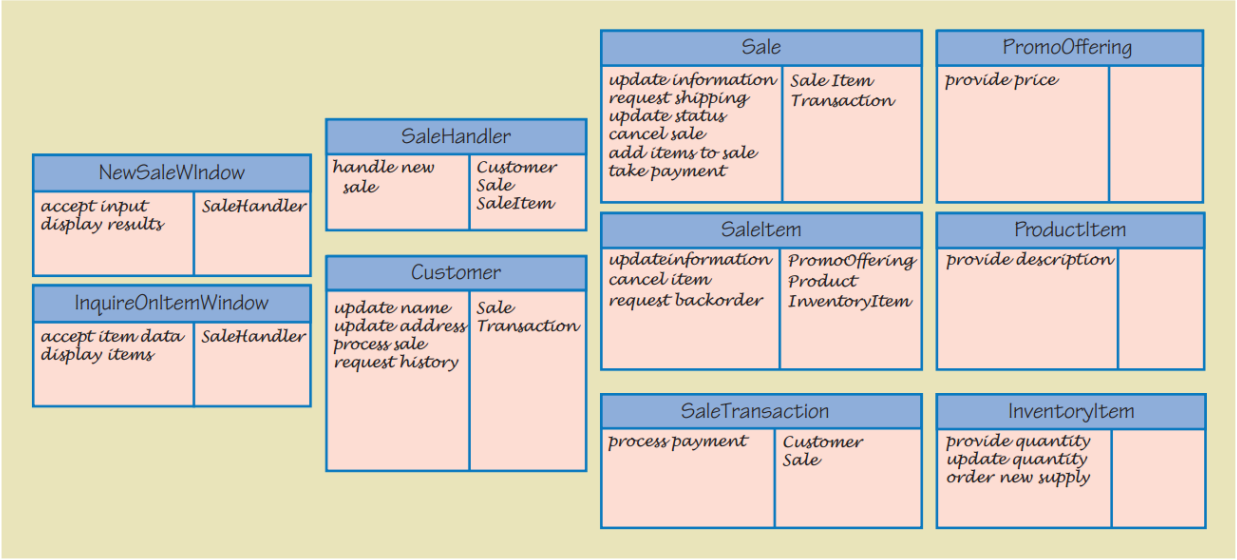


Gambar 2.13 Contoh *Package Diagram* (Satzinger, Jackson, & Burd, 2016)

2.8.7 *Persistent Object*

Persistent object atau dikenal juga dengan *persistent class* adalah sebuah *entity class* atau objek yang harus tetap ada sekalipun sistem dimatikan. Cara untuk membuat suatu data tetap ada adalah dengan menuliskannya ke *file* atau *database* kemudian disimpan dan dapat mengambilnya kembali ketika data tersebut dibutuhkan oleh perangkat lunak. Dalam kebanyakan kasus, *domain class* juga memiliki masalah karena *persistent class*, yang berarti bahwa nilai data mereka harus disimpan oleh sistem bahkan ketika aplikasi tidak beroperasi (Satzinger, Jackson, & Burd, 2016).

Berikut merupakan contoh dari *Persistent Object* dari *Use Case Create Sale Phone* yang menampilkan objek *Customer* membuat objek *Sale*, objek *Sale* membuat objek *SaleItem*, dan objek *SaleItem* mengakses objek *ProductItem* dan *InventoryItem* untuk mendapatkan informasi yang dibutuhkan.



Gambar 2.14 Contoh *Persistent Object* (Satzinger, Jackson, & Burd, 2016)

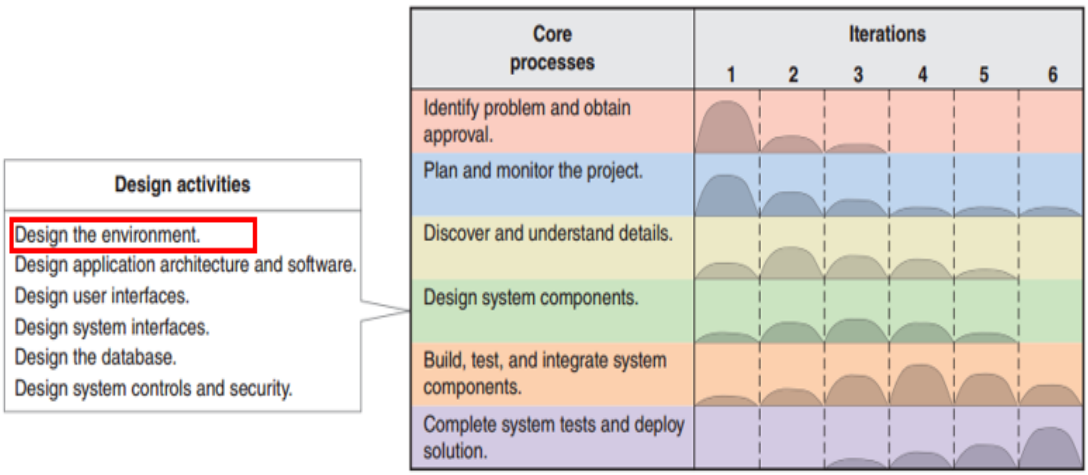
2.9 *Environment Design*

Environment Design merupakan salah satu kegiatan dalam tahap desain aplikasi untuk menentukan spesifikasi yang dibutuhkan oleh aplikasi saat dijalankan. *Environment* adalah semua teknologi yang dibutuhkan untuk mendukung sistem yang sedang dikembangkan. Sistem memerlukan seperangkat *device* yang Dalam pengembangan sistem ada 2 kebutuhan, yaitu kebutuhan fungsional dan kebutuhan non fungsional. Sistem yang sedang dikembangkan berhubungan dengan seperangkat *device* seperti desktop, *server*, *mobile* dan *device* tambahan yang masing-masing

memiliki sistem operasi, kemampuan komunikasi *input* dan *output* yang beragam. *Software* tambahan yang berperan sebagai jembatan yang menghubungkan integrasi antara *device* dan sistem juga mencakup *environment* yang diperlukan agar sistem yang dikembangkan dapat digunakan dengan benar. Oleh karena itu, *environment design* sangat diperlukan agar sistem dapat dijalankan (Satzinger, Jackson, & Burd, 2016).

Teknologi environment mencakup lebih dari sekedar perangkat keras, tetapi memiliki komponen penting lainnya yang terdiri dari sistem operasi, protokol dan sistem komunikasi serta perangkat lunak pendukung lainnya (mis: *middleware*). *Environment design* bahkan lebih rumit ketika sistem perangkat lunak tidak dapat berkomunikasi secara langsung antara satu dengan yang lain. Permasalahan inilah yang harus diatasi dengan melakukan *environment design* yang baik (Satzinger, Jackson, & Burd, 2016).

Berikut merupakan tahapan dalam kegiatan desain aplikasi, dimana di dalamnya terdapat tahap *environment design*.



Gambar 2.15 Tahapan *Design Activity*

2.10 *Website*

Website atau sering disebut *web* dapat diartikan sebagai kumpulan halaman yang menampilkan informasi data teks, data gambar, dataan imasi, suara, video dan gabungan dari semuanya, baik yang bersifat statis maupun dinamis yang membentuk satu rangkaian bangunan yang saling terkait, dimana masing-masing dihubungkan dengan jaringan-jaringan halaman (hyperlink) (Hariyanto, 2015).

2.10.1 *Hypertext Transfer Protocol* (HTTP)

HyperText Transfer Protocol (HTTP) merupakan sebuah *protocol* yang digunakan untuk mengirimkan halaman web melalui internet. HTTP mendefenisikan bagaimana klien dan *server* berkomunikasi (Connolly dan Begg, 2015).

HTTP didasarkan pada paradigma *Request-response*. Sebuah transaksi HTTP terdiri dari beberapa bagian (Connolly dan Begg, 2015):

1. *Connection*, klien membuat koneksi dengan *web server*
2. *Request*, klien mengirimkan pesan *Request* ke *web server*
3. *Response*, *web server* mengirimkan respon misalnya dokumen HTML ke klien
4. *Close*, koneksi ditutup oleh *web server*

HTTP menjadi salah satu dasar dalam pengembangan *World Wide Web*. HTTP merupakan *protocol* yang *stateless*, artinya server tidak menyimpan informasi diantara *Request* yang sedang berlangsung. Untuk sebagian aplikasi *stateless* merupakan keuntungan dimana tidak membutuhkan memori lebih untuk menyimpan data-data selama transaksi, akan tetapi *stateless* membuat kesulitan untuk mendukung konsep *session* yang merupakan esensi dari transaksi DBMS (Connolly dan Begg, 2015).

2.10.2 *Hypertext Markup Language* (HTML)

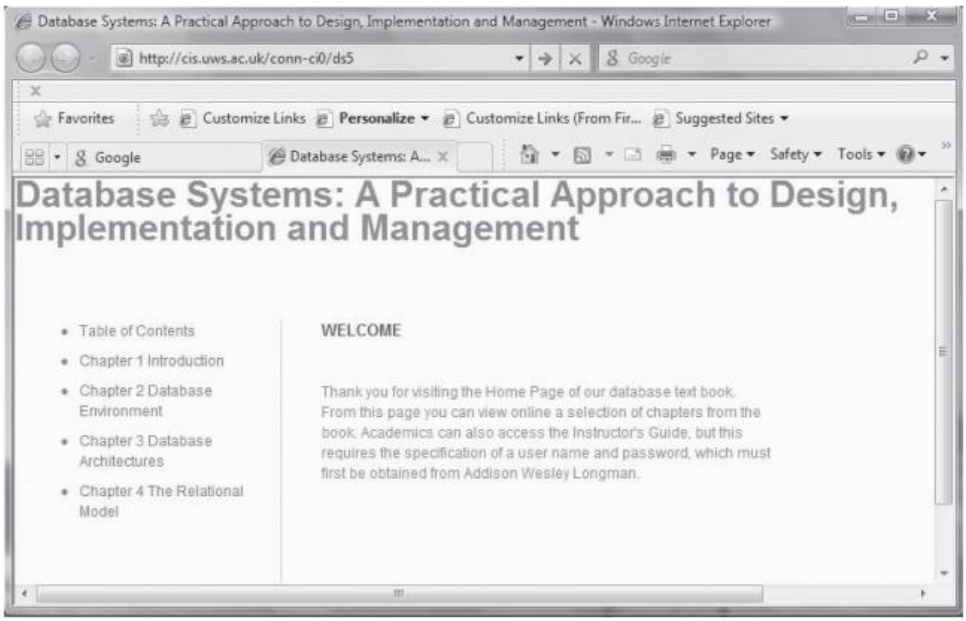
Hyper Text Markup Language adalah dokumen yang mengatur bahasa-bahasa yang digunakan untuk mendesain kebanyakan halaman web. HTML dirancang dengan menggunakan sistem markah/tanda (*tagging*) sehingga markah tersebut mampu ditampilkan menjadi halaman web yang utuh (Connolly dan Begg, 2015).

HTML (*Hyper Text Markup Language*) diuraikan sebagai bahasa standar yang digunakan oleh *browser* internet untuk membuat halaman dan dokumen pada sebuah web yang kemudian dapat diakses dan dibaca layaknya sebuah artikel. HTML juga dapat digunakan sebagai penghubung antara *file-file* dalam situs atau dalam komputer dengan menggunakan *localhost*, atau jaringan yang menghubungkan antar situs dalam dunia *internet*.

Berikut merupakan contoh dari HTML dan hasil konversinya menjadi bentuk web.

```
<HTML>
<HEAD>
<TITLE>Database Systems: A Practical Approach to Design, Implementation and Management </TITLE>
</HEAD>
<BODY bgcolor=#FFFFCC
<H2>Database Systems: A Practical Approach to Design, Implementation and Management</H2>
<P>Thank you for visiting the Home Page of our database text book. From this page you can view online a selection of chapters from the book. Academics can also access the Instructor's Guide, but this requires the specification of a user name and password, which must first be obtained from Addison Wesley Longman. <BR>
<BR>
<A HREF="http://cis.paisley.ac.uk/conn-ci0/book/toc.html">Table of Contents <BR>
</A><A HREF="http://cis.paisley.ac.uk/conn-ci0/book/chapter1.html">Chapter 1 Introduction <BR>
</A><A HREF="http://cis.paisley.ac.uk/conn-ci0/book/chapter2.html">Chapter 2 Database Environment <BR>
</A><A HREF="http://cis.paisley.ac.uk/conn-ci0/book/chapter3.html">Chapter 3 The Relational Data Model
</A></P>
<P><A HREF="http://cis.paisley.ac.uk/conn-ci0/book/ig.html">Instructor's Guide</A></P>
<P>If you have any comments, we would be more than happy to hear from you.</P>
<P><IMG SRC="net.gif" HEIGHT=34 WIDTH=52 ALIGN=CENTER>
<A HREF="mailto:conn-ci0@paisley.ac.uk">EMail</A>
<IMG SRC="fax.gif" HEIGHT=34 WIDTH=43 ALIGN=CENTER>
<A>   Fax: 0141-848-3542</P>
</BODY>
</HTML>
```

Gambar 2.16 Contoh Sintaks HTML (Connolly dan Begg, 2015)



Gambar 2.17 Tampilan HTML pada Browser (Connolly dan Begg, 2015)

2.10.3 *Uniform Resource Locator* (URL)

Uniform Resource Locator (URL) merupakan rangkaian karakter alfanumerik unik yang digunakan untuk menunjukkan lokasi atau alamat sebuah sumber (*resource*) dan bagaimana sumber tersebut harus diakses di internet. URL paling sering digunakan sebagai skema identifikasi dan dasar bagi HTTP dan Web (Connolly dan Begg, 2015).

Sintaks dari URL terbagi atas 3 bagian, yaitu *protocol* yang digunakan untuk koneksi, *host name*, dan *path name* dimana sumber bias ditemukan. Sebagai

tambahan, URL secara opsional dapat menspesifikasikan port yang akan digunakan sesuai dengan koneksi yang akan dibuat (*default* HTTP adalah 80). Bentuk dari URL adalah sebagai berikut:

<protocol>:// <host> [:<port>] / absolute_path [? arguments]

<protocol> menjelaskan metode yang digunakan untuk berkomunikasi dengan sumber oleh *browser*. Metode akses yang biasa digunakan adalah HTTP, S-HTTP (*secure* HTTP), FTP, *mailto* (mengirim *Email* untuk menspesifikasikan alamat *Email*), Gopher, NTTP dan Telnet. Contohnya:

http://www.w3.org/MarkUp/MarkUp.html

URL diatas digunakan untuk mengakses *homepage* mengenai informasi HTML di W3C. Protokolnya adalah HTTP, *host name* adalah www.w3.org, dan *path virtual* HTML adalah /MarkUp/MarkUp.html.

2.10.4 *Personel Home Page* (PHP)

Personal Home Page (PHP) adalah bahasa pemrograman *open-source* yang menempel pada HTML yang didukung oleh banyak *websever* termasuk apache HTTP *server* dan Microsoft's IIS. PHP juga disukai dalam bahasa pemrograman pengembang web Linux. Pengembangan PHP telah dipengaruhi oleh sebagian besar bahasa pemrograman seperti C, Java, Perl. Tujuan bahasa ini adalah untuk memungkinkan pengembang Web menulis halaman yang dihasilkan secara dinamis dengan cepat. Salah satu keuntungan dari PHP adalah ekstensibilitas, dan sejumlah modul ekstensi telah disediakan untuk mendukung koneksi basis data, surat-menyurat (*mail*), dan XML (Connolly dan Begg, 2015).

PHP merupakan bahasa standar yang digunakan dalam dunia website. PHP adalah bahasa pemrograman yang berbentuk script yang diletakkan didalam web server. PHP dapat diartikan sebagai Hypertext Preeprocessor. Ini merupakan bahasa yang hanya dapat berjalan pada server yang hasilnya dapat ditampilkan pada klien. Interpreter PHP dalam mengeksekusi kode PHP pada sisi server disebut server side (Trimarsiah dan Arafat, 2017).

2.10.5 *Cascading Style Sheet* (CSS)

Cascading style sheet (CSS) adalah bahasa yang mendeskripsikan desain dari sebuah dokumen HTML dan bagaimana elemen-elemen HTML tersebut seharusnya ditampilkan. CSS sendiri merupakan sebuah teknologi internet yang

direkomendasikan oleh W3C (World Wide Web Consortium) dan diperkenalkan pada tahun 1996.

Dengan menggunakan CSS, pengembang dapat menerapkan desain ke halaman web agar tampilan web sesuai dengan apa yang diinginkan. Hal ini dapat terjadi karena CSS terintegrasi ke DOM (*Document Object Model*) sehingga pengembang dapat merubah desain setiap elemen dengan cepat dan mudah. Cara untuk merubah desain halaman web dengan CSS adalah dengan menambahkan *required tag* di bagian atas halaman web, yaitu diantara <head> dan </head> (Nixon, 2015).

Berikut ini menunjukkan contoh sintaks CSS untuk merubah *font* h1 dimulai dari ukuran, warna dan jenis tulisan:

```
<style>
    h1 {color:red; font-size:3em; font-family:Arial;}
</style>
```

2.10.6 *Javascript*

JavaScript adalah bahasa *scripting* berbasis objek yang dikembangkan pada program pengembangan bersama antara Netscape dan Sun, dan telah menjadi bahasa scripting Web Netscape. *JavaScript* merupakan bahasa pemrograman yang sangat sederhana yang memungkinkan HTML memakai fungsi dan *script* yang dapat merespon kepada *event* pengguna seperti *mouse click*, imput pengguna dan navigasi halaman. Dengan kata lain *JavaScript* mampu menerapkan *event* pada halaman web dengan baris program yang relative sedikit (Connolly dan Beg, 2015).

Penggunaan sintaks *JavaScript* hampir sama dengan PHP, contohnya dalam pendefenisian fungsi. Membuat sebuah *file JavaScript* diawali dengan tanda <script> dan </script>, setelah itu untuk menulis sebuah fungsi *JavaScript* diawali dengan kata *function* lalu nama fungsinya dengan menuliskannya langsung atau diawali dengan *underscore*. Penamaan dalam *JavaScript* termasuk *case sensitive*, setiap huruf pertama dari kata ditulis dengan huruf capital kecuali kata yang pertama sekali, yang berarti *lowercase* (Nixon, 2015).

Berikut contoh sintaks fungsi *JavaScript* yang digunakan untuk menampilkan data dalam *array*:

```
<script>
```

```
displayItem("Dog","Cat", "Pony", "Hamster","Tortoise")
function displayItems(v1,v2,v3,v4,v5)
{
  document.write(v1 + "<br>")
  document.write(v2 + "<br>")
  document.write(v3 + "<br>")
  document.write(v4 + "<br>")
  document.write(v5 + "<br>")
}
</script>
```

Sintaks diatas jika dijalankan maka akan menghasilkan teks di *browser* seperti berikut:

Dog

Cat

Pony

Hamster

Tortoise

2.11 Basis Data (*Database*)

Basis Data adalah kumpulan dari data yang tersimpan secara terintegrasi, di-*manage* dan dikontrol secara terpusat. *Database* terdiri dari dua bagian yang berhubungan yaitu *physical data store* dan *schema*. *Physical data store* menampung data bit dan *byte* mentah yang dibuat dan digunakan oleh sistem informasi (misalnya nama, harga, saldo). Sedangkan *schema* menampung informasi deskriptif tentang data yang ditampung dalam *physical data store* (Satzinger, Jackson, Burd, 2016).

2.11.1 Pengertian *Database Management System*

Database Management System (DBMS) merupakan perangkat lunak yang memungkinkan pengguna untuk mendefenisikan, membuat, mengelola dan mengatur akses dengan *database*. DBMS merupakan perantara bagi pengguna dengan basis data. Salah satu tujuan dari DBMS adalah memberikan tampilan kepada pengguna dalam penyampaian data dengan abstraksi data. Abstraksi data terbagi menjadi tiga tingkatan, diantaranya (Conolly and Beg, 2015):

1. Level Fisik

Level fisik merupakan level yang paling bawah, dimana pada level ini memperlihatkan bagaimana sesungguhnya data disimpan.

2. Level Konseptual

Level ini menggambarkan bagaimana basis data disimpan dan berhubungan dengan data lainnya.

3. Level *View*

Level ini menunjukan sebagian dari basis data. Pada umumnya pengguna tidak terlibat secara langsung sehingga pengguna melihat data sesuai dengan yang dibutuhkan saja.

Selain itu, DBMS memiliki 5 komponen utama pada DBMS *environment*, diantaranya:

1. *Hardware*

DBMS dan aplikasi membutuhkan *hardware* untuk dapat dijalankan dalam menyimpan dan mengolah *database*.

2. *Software*

Perangkat-perangkat lunak yang dibutuhkan dalam menjalankan DBMS. Perangkat lunak ditulis dengan bahasa pemrograman *third-generation*, seperti C, C++, C#, Java, Visual Basic atau Pascal.

3. *Data*

Data adalah bagian penting dalam DBMS karena data merupakan penghubung antara komponen mesin dan manusia.

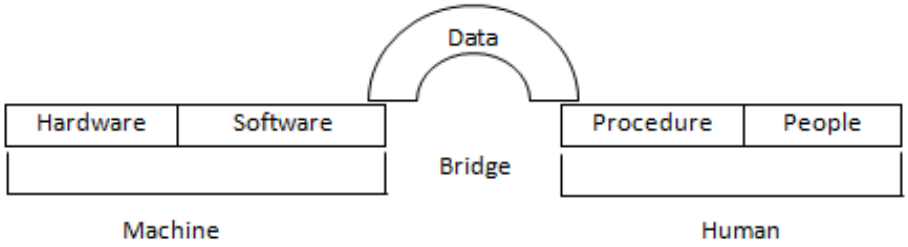
4. *Procedures*

Procedures adalah bagaimana aturan-aturan dalam menggunakan basis data. *User* membutuhkan *procedure* yang terdokumentasi tentang cara menjalankan sebuah sistem.

5. *People*

Merupakan komponen terakhir yang terlibat dengan sistem, misalnya data *administrators, database administrators, database designers, application developers*, dan *end users*.

Berikut gambar dari komponen-komponen DBMS dengan urutannya:



Gambar 2.18 Komponen DBMS (Conolly and Beg, 2015)

2.11.2 *Microsoft SQL Server*

Microsoft SQL Server merupakan salah satu *Relational Database Management System* (RDBMS). *SQL Server* dikembangkan oleh Microsoft yang berfungsi untuk menampung dan menggunakan data yang terintegrasi dengan aplikasi. *Microsoft SQL Server* mampu mengelola *database* dari pembuatan hingga konsep CRUD (*Create*, *Read*, *Update*, dan *Delete*) (Enterprise, 2018).

Salah satu kelebihan dari SQL Server adalah bahasa SQL yang didukungnya yaitu Transact SQL (TSQL). T-SQL merupakan bahasa pengelolaan basis data yang digunakan pada *Database Management System* (DBMS) Microsoft SQL Server. Kelebihan dari T-SQL adalah, perintahperintah T-SQL dapat disisipkan dalam bahasa pemograman. T-SQL memiliki tiga kategori yaitu:

1. *Data Manipulation Languange* (DML) yaitu perintah-perintah untuk melakukan *query* data dan manipulasi data. Perintah yang termasuk dalam kategori ini adalah SELECT, INSERT, UPDATE, DELETE.
2. *Data Definition Language* (DDL) yaitu perintah-perintah untuk mendefinisikan obyek-obyek basis data (*table*, *view*, *stored procedure*). Perintah yang termasuk dalam kategori ini adalah CREATE, ALTER, DROP.
3. *Data Control Language* (DCL) yaitu perintahperintah untuk pengaturan keamanan. Perintah yang termasuk dalam kategori ini adalah GRANT, REVOKE, DENY.

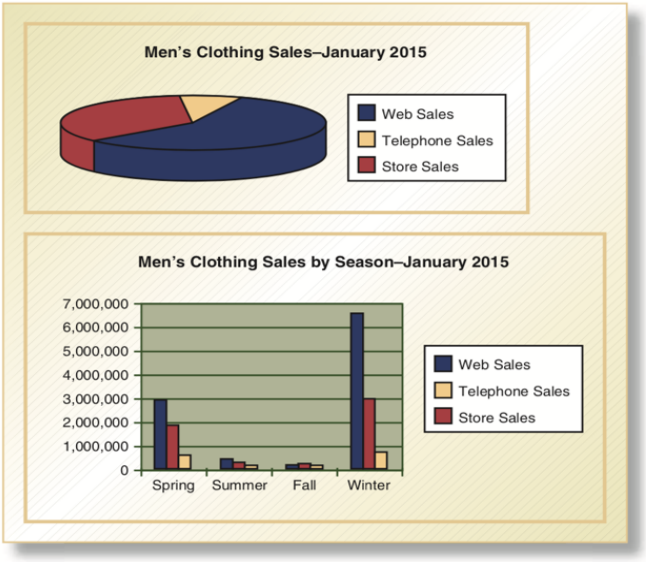
2.11.3 *SQL Server Management Studio*

SQL Server Management Studio (SSMS) adalah *tools* terintegrasi *Relational Database Management System* yang dikembangkan oleh Microsoft. SSMS dapat menjalankan laporan analisis kinerja transaksi langsung pada basis data produksi, melakukan evaluasi beban kerja, dan membuat daftar langkah-langkah untuk melakukan migrasi (Varga, Cherry, & D'Antoni, 2016).

2.12 *System Interface*

System interface merupakan input dan output yang membutuhkan sedikit campur tangan manusia. *System interface* memungkinkan sebuah inputan ditangkap secara otomatis oleh perangkat input khusus seperti sistem *scanner*, sistem pesan elektronik ke sistem lain dan sebaliknya, atau transaksi juga bisa ditangkap oleh sistem lain (Satzinger, Jackson, Burd, 2012).

Berikut merupakan contoh *System Interface* dari salah satu aplikasi, yaitu RMO Customer Support System yang menampilkan detail produk.

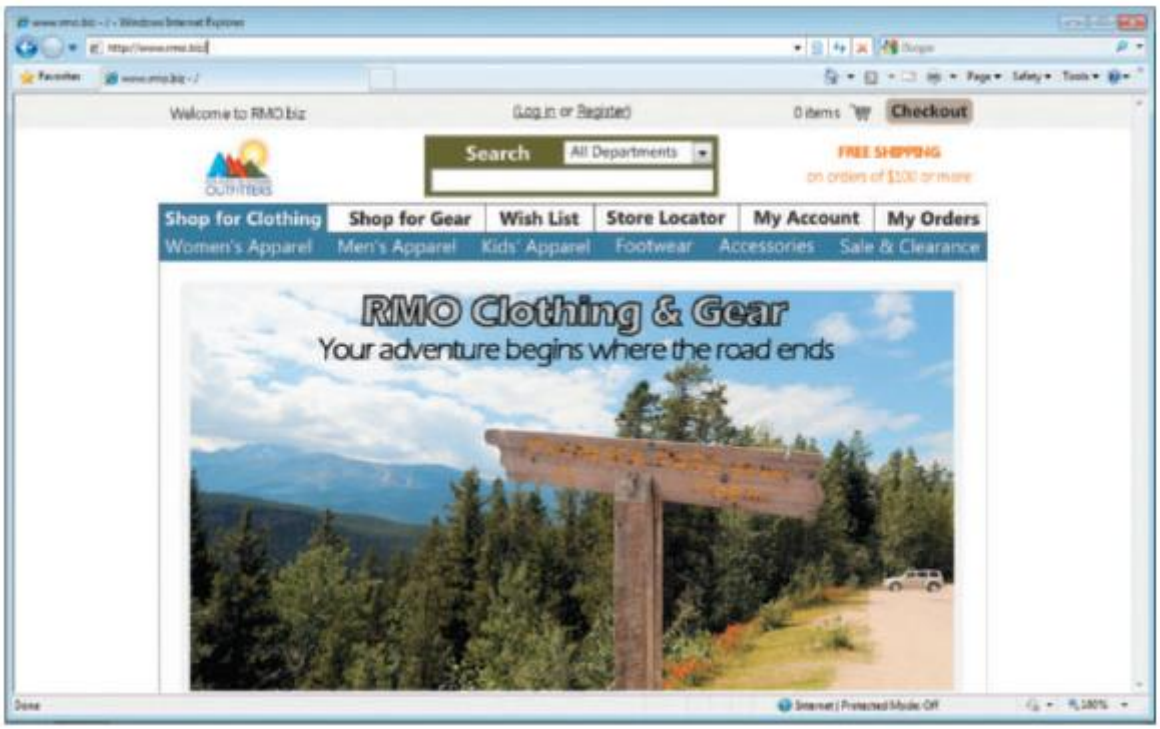


Gambar 2.19 *System Interface* (Satzinger, Jackson, & Burd, 2016)

2.13 *User Interface*

User Interface merupakan *input* dan output yang melibatkan pengguna secara langsung. *User Interface* atau antarmuka pengguna dapat digunakan oleh pengguna internal maupun eksternal. Bagi pengguna, antarmuka adalah sebuah sistem. Karena satu-satunya yang dilihat oleh pengguna adalah antarmuka. Sehingga antarmuka menjadi suatu elemen utama dalam pengalaman pengguna terhadap suatu sistem. (Satzinger, Jackson, Burd, 2016).

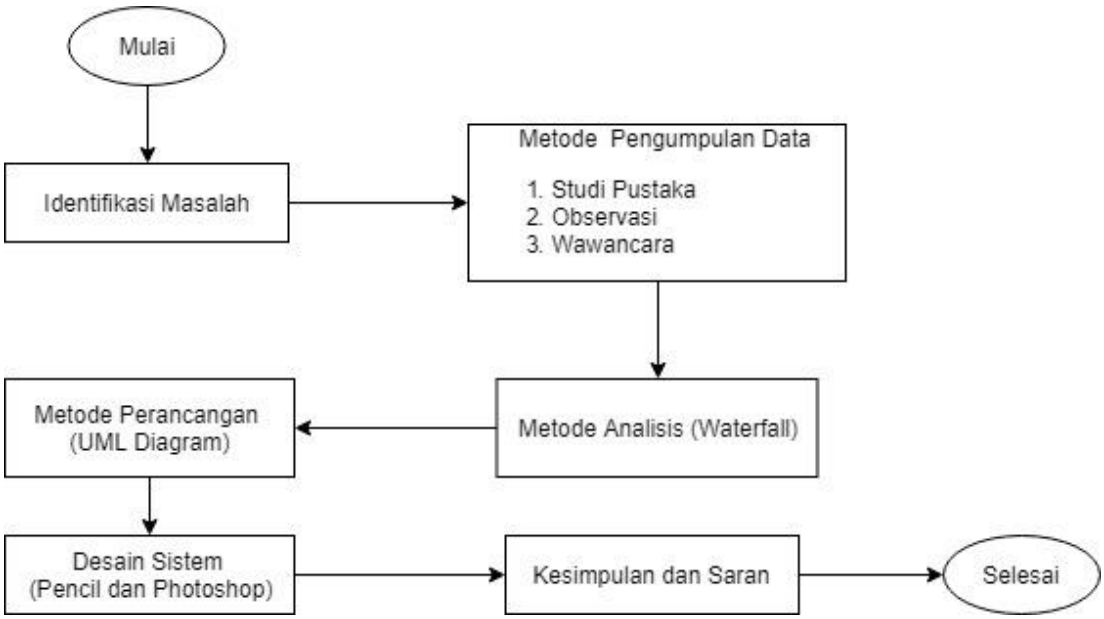
Berikut merupakan contoh *User Interface* dari salah satu aplikasi, yaitu RMO Customer Support System yang menampilkan *homepage* aplikasi.



Gambar 2.20 *User Interface* (Satzinger, Jackson, & Burd, 2016)

2.14 Kerangka Pemikiran

Berikut ini merupakan tahapan penelitian yang dilakukan:



Gambar 2.21 Kerangka Pemikiran

Berikut merupakan rincian dari kerangka pemikiran penelitian ini:

1. Identifikasi masalah

Melakukan identifikasi beberapa masalah yang terjadi dalam melakukan proses permintaan, peminjaman, pengembalian dan pemeliharaan aset pada perusahaan PT. Ras Digital Media, yaitu belum menerapkan IT secara maksimal dalam system manajemen asset khususnya pada divisi IT.

2. Metode Pengumpulan Data

Pengumpulan data dilakukan dengan tiga cara yaitu observasi, studi pustaka dan wawancara. Observasi dilakukan dengan cara melakukan pengamatan secara langsung proses permintaan, peminjaman, pengembalian dan pemeliharaan aset pada perusahaan PT. Ras Digital Media. Studi Pustaka dilakukan dengan cara melakukan studi literatur dan pustaka dari hasil-hasil penelitian terdahulu, yang relevan dengan penelitian yang akan dilakukan. Pada tahap ini dilakukan kajian mengenai konsep dan teori yang berasal dari penelitian sebelumnya, jurnal ilmiah, dan buku yang menjadi dasar studi. Selanjutnya melakukan pengumpulan data dan informasi melalui wawancara secara langsung kepada pihak yang terlibat dalam menangani proses permintaan, peminjaman, pengembalian, dan pemeliharaan aset pada

perusahaan PT. Ras Digital Media. Tahap wawancara ini juga bertujuan untuk mendapatkan informasi seperti struktur organisasi, visi dan misi, profil, tugas dan wewenang serta dokumen lainnya yang relevan pada penelitian ini.

3. Metode Analisis

Melakukan model analisis sistem yang menyelesaikan tahap demi tahap secara berurutan. Model *waterfall* ini terdiri dari tahap project initiation, project planning, analysis, design, implementation dan deployment. Tetapi penelitian ini hanya akan dilakukan sampai pada tahap design saja. Tahap implementation dan deployment tidak akan dilakukan.

4. Metode Perancangan

Melakukan perancangan sistem dengan menggambarkan rancangan sistem menggunakan *Unified Modelling Language* (UML) *Diagram*. UML *diagram* yang dibuat terdiri dari *Activity Diagram*, *Use Case Diagram*, *Domain Model Class Diagram*, *First Cut Class Diagram*, *Update Design Class Diagram*, *System Sequence Diagram*, dan *Entity Relationship Diagram*

5. Desain Sistem

Pada tahap ini akan dilakukan desain prototype sistem sesuai dengan hasil perancangan yang dilakukan sebelumnya. Desain sistem akan menggunakan *pencil* dan *photoshop* sebagai tools.

6. Kesimpulan dan Saran

Melakukan kesimpulan dari hasil perancangan sistem yang telah dilakukan, dan memberikan saran untuk dapat melakukan pengembangan terhadap perancangan sistem yang dibuat untuk kedepannya.