BAB 2

LANDASAN TEORI

2.1 Teori Umum

2.1.1 Membangun Perangkat Lunak

Menurut pendapat Roger S. Pressman (2010, p1) “Perangkat Lunak Komputer adalah produk yang dibangun oleh seorang tenaga profesional dan bisa dikembangkan dalam jangka waktu yang panjang. Meliputi program yang akan dieksekusi oleh komputer dengan ukuran dan arsitektur tertentu. Konten yang berada di dalamnya merupakan implementasi dari eksekusi program komputer, bisa berupa informasi deskriptif maupun informasi virtual”.

Rekayasa perangkat lunak mencakup proses, kumpulan metode, dan berbagai alat yang mendukung tenaga profesional untuk membangun perangkat lunak komputer yang berkualitas tinggi.

Sekelompok praktisi perangkat lunak membangun serta mengembangkan perangkat lunaknya secara bersama-sama. Pada era *modern* ini, hampir semua orang yang berada di dunia industri memanfaatkan perangkat lunak demi kemajuan industrinya.

Perangkat lunak menjelma menjadi sesuatu yang penting bagi kehidupan manusia, karena keberadaannya mempengaruhi setiap aspek kehidupan. Meliputi perdagangan, budaya, serta kegiatan sehari-hari.

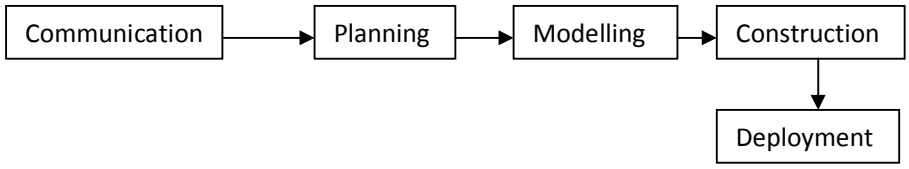
Membangun sebuah perangkat lunak sama dengan membangun produk-produk lainnya. Penerapan proses yang terencana, beradaptasi dengan perkembangan perangkat lunak dengan maksimal agar menghasilkan produk berkualitas tinggi dan dapat memenuhi kebutuhan orang banyak.

Ada beberapa cara dalam membangun atau membuat sebuah perangkat lunak antara lain adalah *Building and Fix Model*, *Waterfall Model, Prototyping, Incremental Model,* dan *Spiral Model*. Namun karena dalam pembangunan perangkat lunak ini yang digunakan adalah *Waterfall model* sehingga akan dijelaskan tentang *Waterfall model* secara menyeluruh.

2.1.2 *Waterfall Model*

Menurut Roger S. Pressman (2010, p39), "*Waterfall Model* dapat juga disebut sebagai *Classic Life Cycle*. Menunjukkan sebuah pendekatan sistematis untuk pengembangan perangkat lunak. Diawali dengan *communication, planning, modelling, construction,* dan *deployment"*.

Seperti gambar dibawah ini:



Gambar 2.1 Tahapan dalam *Waterfall Models*

(Sumber*: Software Engineering, A Practitioner Approach* 7th – Pressman p39)

1. ***Communication*** : Sebelum melakukan pekerjaan yang bersifat teknis, penting bagi *developer* untuk berkomunikasi dan berkolaborasi secara berkala dengan pelanggan atau perusahaan. Hal ini dilakukan agar

developer mengerti tujuan utama perusahaan dalam pembuatan perangkat lunak dan dalam rangka mengumpulkan segala persyaratan untuk membuat fitur dalam perangkat lunak itu sendiri.

2. ***Planning*** : Segala macam perjalanan sulit membutuhkan map untuk mempermudahnya. Seperti itulah proyek pembuatan perangkat lunak. Proyek perangkat lunak diibaratkan sebagai sebuah perjalanan. Sedangkan mapnya adalah perencanaan proyek perangkat lunak. Pada tahap ini *developer* mulai memikirkan tugas-tugas teknis apa yang akan dilakukan, risiko yang mungkin terjadi, sumber daya apa saja yang diperlukan, produk apa saja yang akan di produksi serta membuat modul kerja.

3. ***Modelling*** : Seorang arsitek, penata taman, pembangun jembatan, insinyur penerbangan, sampai tukang kayu, selalu bergelut dalam pembuatan desain model. Semuanya diawali dengan pembuatan sketsa. Jika semua bagian-bagiannya cocok dengan karakteristik lainnya, sketsa mulai dibuat lebih detail agar kita dapat lebih memahami masalah yang ada. Seperti itulah Rekayasa Perangkat Lunak. Kita harus memulainya dengan membuat suatu model agar dapat memahami kebutuhan perangkat lunak tersebut. Kemudian desain yang dibuat harus sesuai agar mencapai kebutuhan awal yang diminta.

4. ***Construction*** : Pada tahap ini *developer* mulai membuat koding (pembuatan kode) baik manual atau otomatis. Jika sudah selesai, maka

pengujian harus langsung dilakukan untuk meminimalisir kesalahan-kesalahan dalam koding.

5. ***Deployment*** : Perangkat Lunak sudah dapat dikirimkan kepada pelanggan dan pelanggan akan memberikan umpan balik sekiranya ada yang perlu di evaluasi pada perangkat lunak tersebut.

2.1.3 Interaksi Manusia dan Komputer

2.1.3.1 Pengertian Delapan Aturan Emas

Menurut Shneiderman dan Plaisant (2010, p88-89), terdapat delapan aturan emas dalam merancang *interface*, yaitu:

1. Berusaha Untuk Konsisten

Urutan tindakan yang konsisten harus diminta dalam situasi yang sama, terminologi identik harus digunakan dalam *prompt*, menu dan layar *help* dan warna yang konsisten, tata letak, kapitalisasi, *font*, dan sebagainya harus digunakan secara keseluruhan. Pengecualian, seperti konfirmasi yang diperlukan dari perintah menghapus atau tidak mengulang *password*, harus dipahami dan terbatas jumlahnya.

2. Dapat Digunakan Secara Universal

Mengenali kebutuhan *user* yang beragam dan desain yang elastis, memfasilitasi transformasi dari konten. Membedakan *user* pemula dan ahli, rentang usia, kecacatan, dan keragaman teknologi yang menambah jenis-jenis kebutuhan yang menuntun suatu desain. Penambahan fitur untuk pemula, seperti penjelasan, dan fitur untuk

ahli, seperti *shortcut* dan lompat halaman lebih cepat, dapat memperkaya desain antarmuka dan meningkatkan kualitas sistem.

3. **Menawarkan Umpan Balik yang Informatif**

Untuk setiap tindakan *user*, harus ada sistem umpan balik. Untuk tindakan yang sering dan aneh, bisa dengan respon yang sederhana, sedangkan untuk tindakan yang jarang dan penting, respon harus lebih bersubstansi. Presentasi visual dari kepentingan obyek dalam menyediakan lingkungan yang nyaman dalam menampilkan perubahan yang eksplisit.

4. **Merancang Dialog yang Menghasilkan Penutupan (Keadaan Akhir)**

Urutan tindakan harus diatur ke dalam kelompok dengan sebuah awalan, pertengahan dan akhir. Umpan balik yang informatif pada penyelesaian sekumpulan tindakan memberikan kepuasan pada penyelesaian, rasa lega, sinyal untuk memutuskan rencana darurat dan sinyal untuk menyiapkan sekumpulan tindakan berikutnya.

5. **Mencegah Kesalahan**

Sebanyak mungkin, merancang sistem sehingga *user* tidak membuat kesalahan yang serius. Jika *user* membuat kesalahan, antarmuka harus mendeteksi kesalahan dan menawarkan instruksi sederhana, konstruktif dan spesifik untuk perbaikan. Tindakan yang salah tidak boleh meninggalkan perubahan, atau antarmuka akan memberikan konstruksi untuk mengembalikan keadaan.

6. Memungkinkan untuk Mengembalikan Keadaan dengan Mudah

Sebanyak mungkin, tindakan harus dapat dibatalkan. Fitur ini untuk mengurangi kecemasan, karena *user* tahu bahwa kesalahan dapat dibatalkan, sehingga mendorong *user* untuk bereksplorasi ke pilihan yang tidak biasa.

7. Mendukung Pusat Kendali Internal

User yang berpengalaman memiliki keinginan kuat bahwa *user* dihadapkan pada suatu antarmuka dan bahwa antarmuka akan merespon tindakan *user*. *User* tidak menginginkan kejutan ataupun perubahan yang sama, dan *user* juga tidak menginginkan tindakan pengulangan dalam memasukan data secara terus-menerus, kesulitan dalam memperoleh 11 informasi, dan ketidakmampuan untuk menghasilkan sesuatu yang diinginkan.

8. Mengurangi Beban Ingatan Jangka Pendek

Keterbatasan dalam memproses informasi pada manusia dalam memori jangka pendek mengharapkan agar tampilan tetap sederhana, menampilkan penggabungan beberapa halaman, frekuensi gerakan tampilan dikurangi, dan waktu pelatihan yang cukup dibagikan untuk pengkodean, mnemonic dan tindakan berurutan. Jika sesuai, akses *online* ke bentuk sintak *command*, singkatan, kode, dan informasi lainnya harus disediakan.

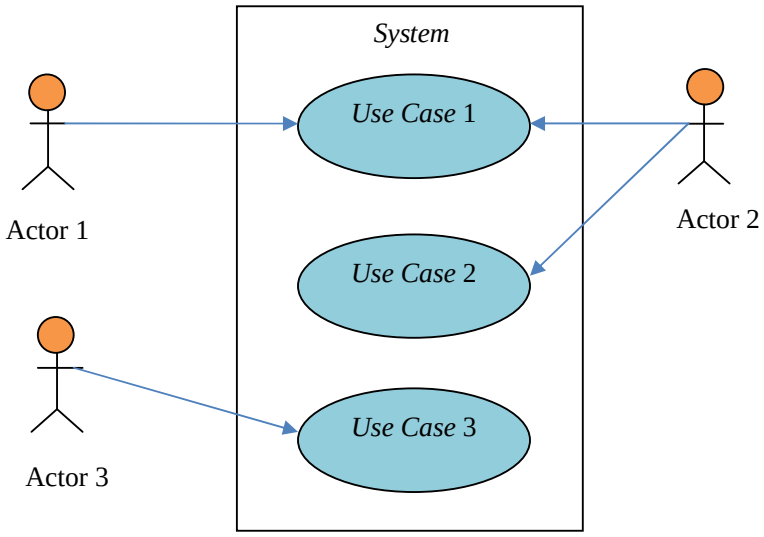
2.1.4 Unified Modeling Language (UML)

UML adalah suatu kumpulan permodelan konvensional yang digunakan untuk menspesifikasi atau mendeskripsikan suatu sistem perangkat lunak dalam bentuk obyek. (Whitten & Bentley, 2007, p371).

Berikut akan dijelaskan 4 macam *diagram* yang digunakan dalam pembangunan aplikasi berorientasi obyek, yaitu *use case diagram, sequence diagram, activity diagram, dan class diagram.*

2.1.4.1 *Use Case Diagram*

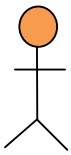
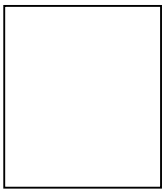
Menurut Whitten dan Bentley (2007, p246), *Use Case Diagram* menggambarkan interaksi antara sistem dan sistem eksternal, serta *user*. Dengan kata lain, *Use Case Diagram* menggambarkan siapa yang akan menggunakan sistem dan dengan jalan apa yang diinginkan *user* untuk berinteraksi dengan sistem. Selain itu, *Use Case* digunakan untuk secara tekstual menggambarkan urutan langkah setiap interaksi tersebut. Berikut adalah contoh *Use Case Diagram* sederhana :

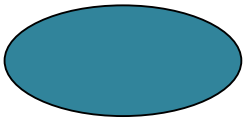


Gambar 2.2 Contoh *Use Case Diagram* Sederhana

(*System Analysis and Design Methods* 7th - Whitten & Bentley)

Tabel 2.1 Keterangan *Use Case Diagram*

Actor		Orang, organisasi atau sistem eksternal yang berperan dalam satu atau lebih interaksi dengan *Use case*.
System Boundary Boxes		Menunjukkan ruang lingkup dari sistem anda. Apapun di dalam kotak merupakan fungsi yang ada di dalam lingkup, apapun yang ada diluar kotak tidak termasuk didalam fungsi.
Association		Asosiasi ada setiap kali seorang aktor yang terlibat dengan interaksi di jelaskan oleh *use case*. Di modelkan sebagai garis yang menghubungkan kasus penggunaan dan aktor satu sama lain. Dengan mata

		panah *optional* pada salah satu ujung baris, panah sering digunakan untuk menunjukkan arah awal hubungan atau untuk menunjukkan aktor utama.
Use Cases		Menjelaskan suatu urutan tindakan yang menghasilkan sebuah nilai yang terukur untuk aktor.

Tabel 2.2 Keterangan *Use Case Description*

Author : (1)
Date : (2)
Version: (3)

1. *Author* : Individu yang berkontribusi besar atas pembuatan *use case*. Serta menjadi sumber informasi terkait *use case* yang telah dibuat.
2. *Date* : Tanggal terakhir kali *use case* tersebut di modifikasi.
3. Versi dari *use case* itu sendiri.

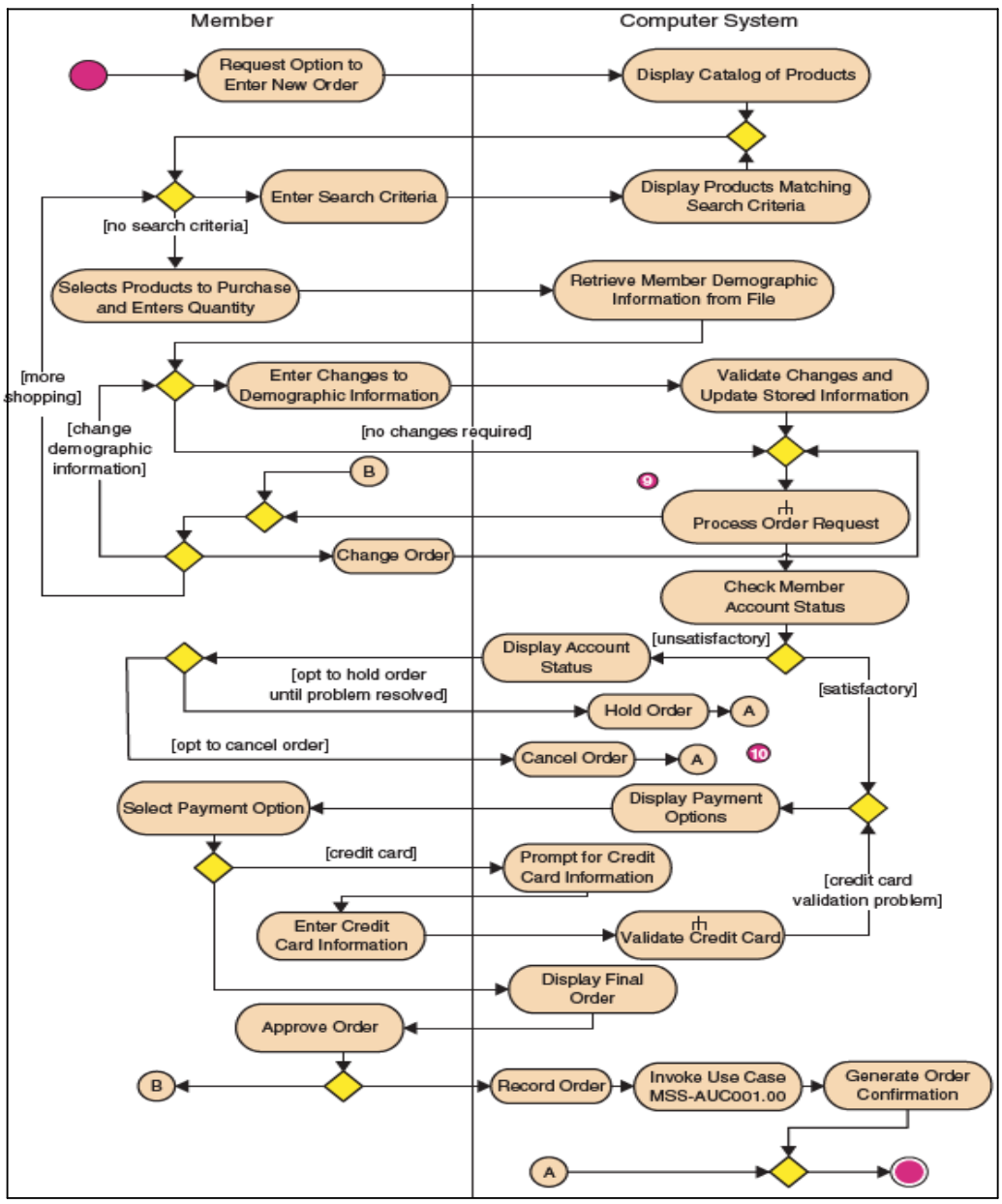
Use Case Name	Nama *use case* harus sesuai dengan tujuan *use case* tersebut dibuat.
Use Case ID	Sebuah tanda pengenal unik *use case*.
Priority	Berguna untuk mengetahui seberapa penting *use case* tersebut, *Low, Medium,*

	atau *High*.
Source	Mendefinisikan identitas yang memicu terciptanya *use case*. Didalamnya terdapat kebutuhan-kebutuhan, dokumen tertentu, dan *stakeholder*
Pre Condition	Kondisi sebelum pemesanan harus diberitahukan.
Other Participation Actor	Aktor lain yang berpartisipasi dalam pembuatan *use case*. Mereka biasanya membantu aktor utama memulai, memfasilitasi aktor utama, menjadi *server* aktor dan aktor sekunder.
Other Interested Stakeholder	Individu yang memiliki kepentingan dalam pengembangan dan pengoperasian sistem perangkat lunak. Sedangkan *Interested Stakeholder* adalah individu yang memiliki kepentingan terkait dengan *use case* yang telah dibuat.
Description	Ringkasan yang berisi beberapa kalimat yang menjelaskan tujuan dari kasus *use case* dan aktivitas didalamnya.
Post Condition	Pesanan telah dicatat dan jika produk yang telah dipesan tersedia maka kondisi

	sudah benar. Jika belum tersedia, maka dibuat perintah khusus.

2.1.4.2 *Activity Diagram*

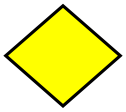
Activity Diagram adalah *diagram* yang bisa digunakan untuk menggambarkan aliran proses bisnis, langkah – langkah dari *use case*, atau logika suatu metode dari suatu obyek secara grafis (Whitten & Bentley, 2007, p390). Secara grafis *activity diagram* sama dengan *flowcharts* yang menggambarkan aliran sekuensial aktifitas bisnis atau *use case*. Tetapi perbedaannya adalah *activity diagram* memiliki mekanisme untuk menggambarkan aktivitas yang terjadi secara paralel. Karena itulah *diagram* ini sangat berguna dalam memodelkan suatu aksi yang akan dilakukan ketika suatu operasi dijalankan bersama dengan hasil dari aksi tersebut.



Gambar 2.3 ***Activity Diagram with Partitioning of the Place New Member Order Use Case***

(System Analysis and Design Methods 7th- Whitten & Bentley)

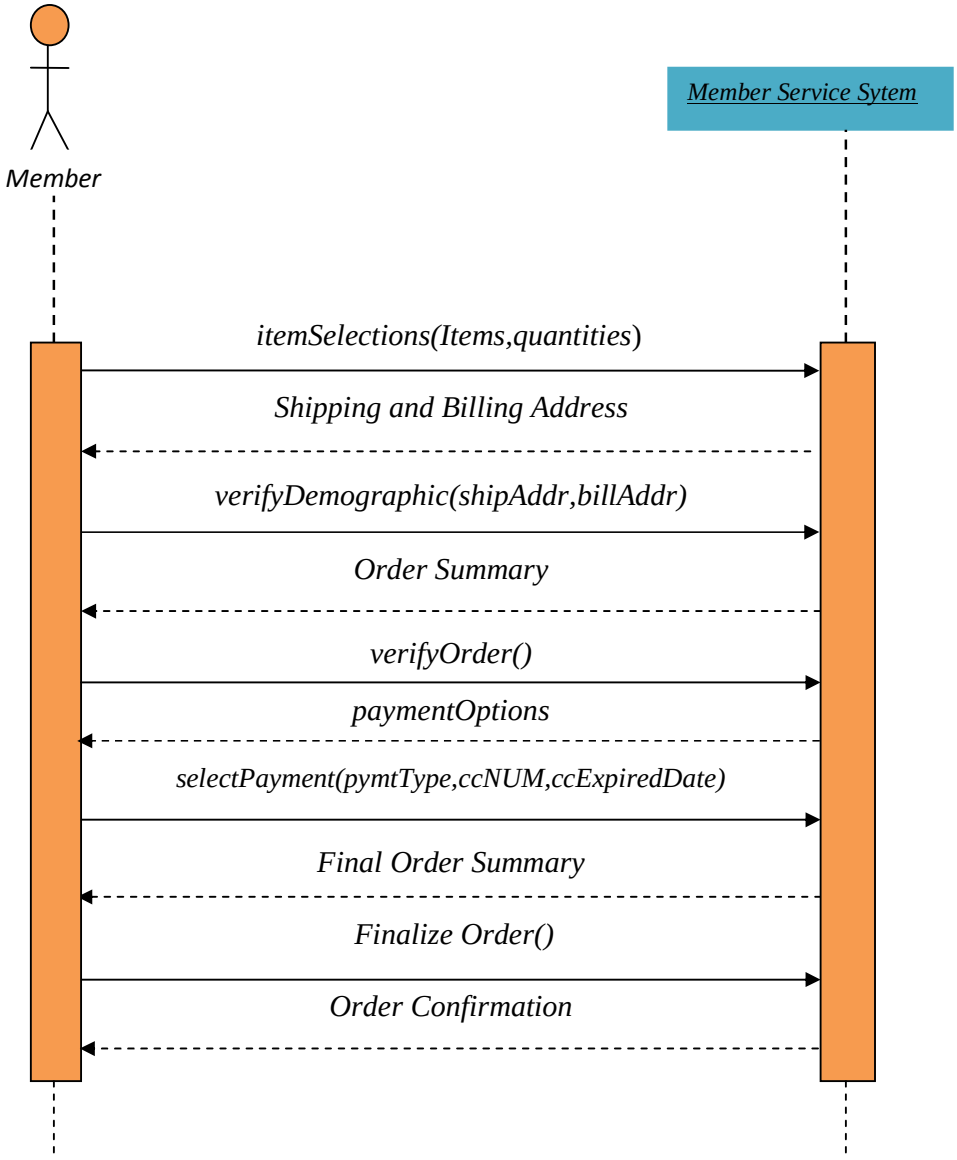
Tabel 2.3 Keterangan *Activity Diagram*

Initial Node		Bulatan merah muda di dalam lingkaran menggambarkan awal dari proses.
Actions		Bulatan persegi panjang menggambarkan langkah individu.
Flow		Panah di dalam *Diagram* menunjukan alur proses.
Decision		Bentuk berlian dengan 1 *flow* yang masuk dan 2 atau lebih *flow* yang keluar. *Flow* yang keluar ditandai untuk mengindikasikan kondisinya.
Activity final		bulatan merah muda dengan garis hitam berbentuk lingkaran menandakan akhir dari proses.

2.1.4.3 *Sequence Diagram*

Sequence Diagram menggambarkan bagaimana obyek berinteraksi satu sama lain melalui pesan di dalam pelaksanaan suatu *use case* atau operasi (Whitten & Bentley 2007, p394). Dunia berbasis obyek berjalan dengan saling mengirim pesan diantara obyek. Sistem dari *sequence*

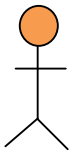
diagram membantu untuk memulai mengindetifikasi *high-level messages* yang masuk dan keluar sistem.



Gambar 2.4 ***System Sequence Diagram For Place New Order Use Case***

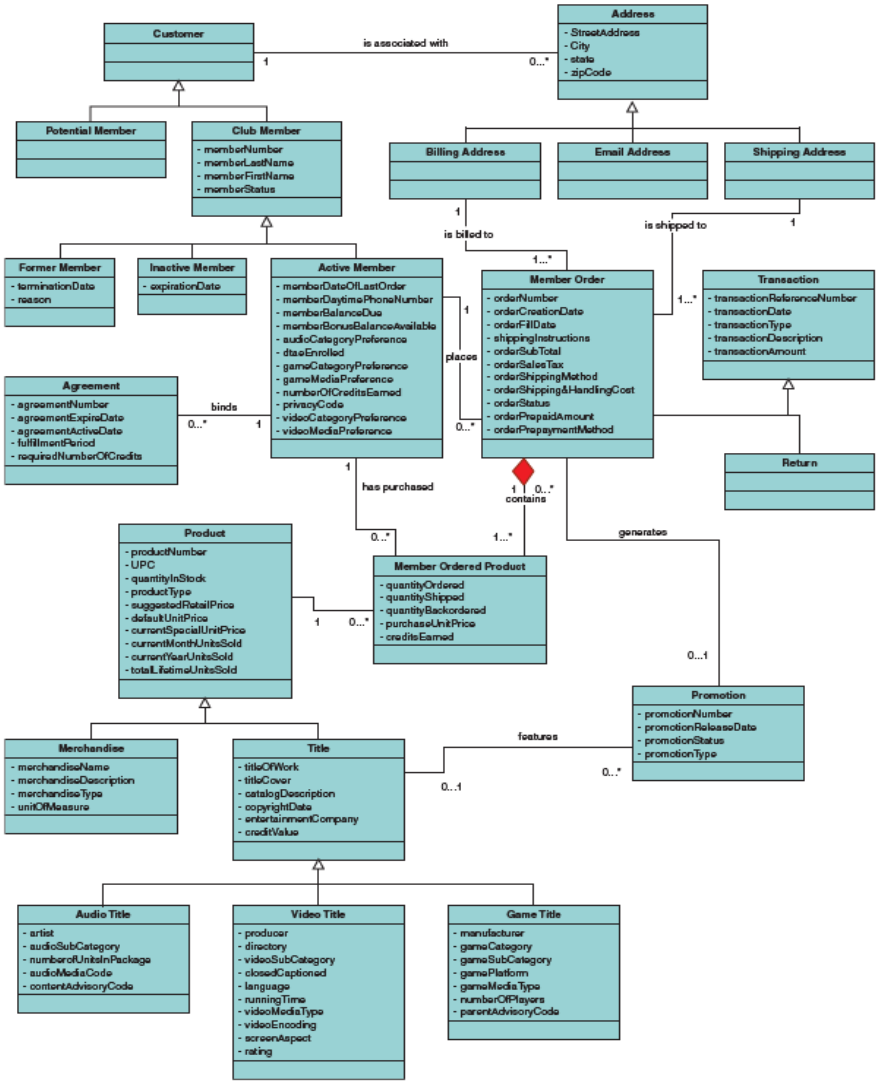
(System Analysis and Design Methods 7th- Whitten & Bentley)

Tabel 2.4 Keterangan *Sequence Diagram*

Actor		Yang memulai yang diambil dari *use case* digambarkan dengan *use case* aktor simbol.
System		kotak mengindikasikan sistem sebagai "*black box*". Merupakan standar notasi pada *sequence Diagram* yang mengindikasikan *instance* yang berjalan dari sistem.
Lifelines		garis *vertical* putus-putus ke arah bawah dari *actor* dan simbol, mengindikasikan hidup di *sequence*.
Activation bars		persegi panjang yang berada pada *lifelines* mengindikasikan periode waktu peserta aktif dalam interaksi.
Input messages		panah horizontal dari aktor ke sistem mengindikasikan pesan masuk.
Output messages		panah horizontal dari sistem ke aktor digambarkan dengan garis putus-putus.

2.1.4.4 *Class Diagram*

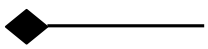
Menurut Whitten dan Bentley (2007, p382), *Class Diagram* menggambarkan struktur dari sistem. Serta menampilkan *Class Object* yang berada di dalam sistem serta hubungan antara obyek tersebut dan obyek lainnya.

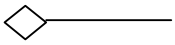
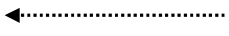


Gambar 2.5 ***Member Services System Class Diagram***

(System Analysis and Design Methods 7th- Whitten & Bentley)

Tabel 2.5 Keterangan *Class Diagram*

Class	**Nama *Class*** – *atribut* – *atribut* – *atribut* – *method* – *method*	*Class* adalah blok-blok pembangunan pada pemrograman berorientasi obyek. Sebuah *class* digambarkan sebagai sebuah kotak yang terbagi atas tiga bagian. Bagian atas mendefinisikan bagian nama *class*, bagian tengah menyatakan atribut *class*, sedangkan bagian bawah adalah *method-method* dari *class*
Association	1..n *owned by* 1	Sebuah asosiasi merupakan *relationship* paling umum antara dua *class* dan dilambangkan oleh sebuah garis yang menghubungkan dua *class* tersebut. Garis ini bisa melambangkan tipe-tipe *relationship* dan juga dapat menampilkan hukum-hukum *multiplicity* pada sebuah *relationship*. (contoh: *One-to-one, one-to-many, many-to-many*)
Composition		Jika sebuah *class* tidak bisa berdiri sendiri dan harus merupakan bagian dari *class* lain, maka *class* tersebut memiliki relasi *Composition* terhadap *class* tempat dia bergantung tersebut. Sebuah *relationship composition* digambarkan sebagai garis

		dengan ujung berbentuk jajaran genjang berisi/solid.
Aggregation		***Aggregation*** mengindikasikan keseluruhan bagian *relationship* dan biasanya disebut dengan *relation.*
Dependency		Terkadang sebuah *class* menggunakan *class* yang lain. Hal ini disebut *dependency*. Umumnya penggunaan *dependency* dugunakan untuk menunjukkan operasi pada suatu *class* yang menggunakan *class* lain. Sebuah *dependency* dilambangkan sebagai sebuah panah bertitik-titik.

2.1.5 *Object Oriented Programming* (OOP)

Object oriented merupakan suatu cara untuk melakukan pemodelan sistem dengan berorientasikan pada obyek-obyek yang terlibat pada sistem tersebut. (Britton, 2001, p4).

Fitur yang paling penting dalam *object oriented* adalah bahwa seluruh proses pengembangan didasarkan pada sebuah konsep tunggal utama. obyek-obyek dalam dunia nyata menjadi komponen-komponen dalam analisis dan perancangan model dan akhirnya menjadi bagian dari kode akhir.

Keuntungan dari *object oriented* adalah :

1. Merupakan konsep umum yang dapat digunakan untuk memodelkan hampir semua fenomena dan dapat dinyatakan dalam bahasa umum. Misalnya :
    a. Kata benda menjadi *object class.*
    b. Kata kerja menjadi *behavior.*
    c. Kata sifat menjadi *attributes.*
2. Memberikan informasi yang jelas tentang konteks dari sistem.
3. Mengurangi biaya *maintenance*
4. Memudahkan dalam mencari hal yang diubah.
5. Membuat perubahan menjadi bersifat lokal, tidak berpengaruh terhadap modul lainnya.

Alasan-alasan mengapa pendekatan *object oriented* diperlukan, adalah para praktisi perangkat lunak mencari sebuah pendekatan baru pada pengembangan sistem yang akan menghasilkan perangkat lunak yang lebih:

1. *Maintainable* (Dapat Dipelihara)

Selama masa hidupnya, sebuah aplikasi akan diminta untuk berubah untuk memenuhi persyaratan-persyaratan baru.

2. *Testable* (Dapat Diuji)

Unit-unit peranti lunak yang dibangun menggunakan pendekatan *object oriented* menghasilkan hasil yang mencukupi dan independen dengan *interface* yang didefinisikan dengan jelas. Ini berarti setiap unit dapat diuji

dengan hati-hati dalam suatu isolasi sebelum disatukan kembali ke dalam sistem. Integrasi dapat dilaksanakan secara berangsur sehingga kesalahan-kesalahan yang ditemukan dapat dilacak ke modul yang salah.

3. ***Reusable*** **(Dapat Digunakan Kembali)**

Karena banyak masalah yang berisi fitur-fitur yang sama, cara yang pasti untuk menghasilkan peranti lunak yang efesien adalah dengan menggunakan kembali solusi-solusi dari *software* yang telah ada, daripada menulis ulang kode tersebut setiap kali.

4. ***Able to Cope with Large and Complex System*** **(Mampu Mencakup Sistem yang Besar dan Kompleks)**

Sistem peranti lunak sekarang ini lebih besar dan lebih komplek daripada yang sebelumnya. Sekarang kebanyakan aplikasi *software* membutuhkan sebuah *GUI* untuk dapat diterima.

Tabel 2.6 Terminologi Dalam *Object Oriented*

Istilah	Definisi
Object	Unit peranti lunak menyatukan data dan metode-metode secara bersama-sama untuk manipulasi data.
Class	*Template* untuk menciptakan *object.*

Attribute	*Item* data yang didefinisikan sebagai bagian dari sebuah *class* atau *object.*
Operation	Prosedur atau fungsi yang didefinisikan sebagai bagian dari sebuah *class* atau *object*, menunjuk pada *interface* umum prosedur.
Method	Prosedur atau fungsi yang didefinisikan sebagai bagian dari sebuah *class* atau *object*, menunjuk pada implementasi prosedur.
Message	Permintaan yang dikirim ke sebuah *object* untuk menjalankan salah satu dari metodenya.
Encapsulation	Penggabungan data dan operasi ke sebuah *object*
Data Hiding	Membuat rincian detail dari sebuah *object* yang tidak dapat diakses oleh *object* lain.
Inheritance	Mekanisme untuk mendefinisikan sebuah *class*

	baru dari *class* yang sudah ada.
Polymorphism	Kemampuan yang dipakai untuk menyembunyikan implementasi – impelmentasi yang berbeda di balik sebuah *interface* umum.

2.1.6 *Internet*

Internet mempuyai pengaruh besar terhadap perkembangan informasi, ilmu pengetahuan dan pandangan dunia. Informasi dalam Internet umumnya disebarkan melalui suatu halaman website yang dibuat dengan format bahasa pemrograman HTML (*Hypertext Markup Languange*). Untuk dapat menampilkan halaman website diperlukan suatu perangkat lunak aplikasi yang disebut dengan *browser*. Mozilla Firefox, Opera, Google Chrome, Safari dan Internet Explorer merupakan contoh dari browser. Halaman utama suatu website disebut dengan *homepage*. Dari halaman utama kita dapat membuka berbagai macam informasi melalui tombol yang disebut dengan *link*. Link dapat menghubungkan kita dengan halaman atau website lainnya, sehingga informasi yang dapat kita peroleh menjadi kaya.

Layanan berupa situs yang digunakan dalam memudahkan pencarian informasi disebut dengan *Web Search Engine*. Contoh dari *web search engine* adalah Google, Yahoo, dan Bing. Dengan *web search engine* kita cukup menuliskan kata kunci dari informasi yang akan kita cari, dan dalam hitungan detik informasi tersebut dapat ditemukan. Misalnya dalam mencari informasi

tentang artis favorit, kita tinggal mengetik nama artis tersebut sebagai kata kunci di *web search engine*.

Menurut Atzeni et al. (2003, p490), internet dapat didefinisikan sebagai sebuah federasi dari jaringan yang berkomunikasi dengan cara yang sama seperti protokol, seperti TCP/IP (*Transmission Control Protocol/Internet Protocol*). Biasanya, sebuah *node* dari internet adalah bagian dari jaringan lokal, yang menghubungkan *workstation* atau personal komputer yang terletak dalam wilayah kecil.

Pengguna (orang atau program) dapat merujuk pada setiap *node* dengan alamat yang dikenal. Dari sudut pandang logis, setiap node internet dapat terhubung langsung dengan *node* lain. Karakteristik fundamental dari semua aplikasi yang beroperasi di internet adalah dengan menerapkan paradigma *client-server*. *Client* mengelola interaksi dengan pengguna dan *server* melaksanakan operasi yang diminta, memberikan respon yang sesuai pada *client*. *Server* menawarkan serangkaian fungsi standar (khususnya, jasa 9 *web*), berdasarkan *standard protocol* (seperti HTTP), dengan menggunakan layanan yang diberikan oleh TCP/IP.

Setiap *node* (komputer) di internet memiliki alamat IP, yang diidentifikasi dengan cara yang unik. Alamat IP terdiri dari empat angka, misalnya 131.175.21.1. Biasanya, mesin di internet juga memiliki nama simbolis, merupakan tanda pengenal yang dipisahkan oleh titik, yang dapat digunakan sebagai pengganti dari alamat IP.

2.1.6.1 *World Wide Web*

Menurt Darma, Jarot S., Shenia A (2009), *World Wibe Web* (WWW) merupakan suatu ruang informasi yang dipakai oleh pengenal global yang disebut pengidentifikasi sumber seragam untuk mengenal pasti sumber daya berguna. WWW terkadang sering dianggap sama dengan internet secara keseluruhan, walaupun sebenarnya hanyalah bagian daripada internet.

Menurut Sunarto, S.Kom (2006), WWW mempunyai kegunaan untuk menyediakan data dan informasi untuk dapat digunakan bersama, WWW adalah bagian yang menarik dari internet. Melalui web, para pengguna dapat mengakases informasi tidak hanya berupa teks melainkan dapat berupa gambar, suara, video dan animasi.

Menurut Atzeni et al. (2003, p491) , *hypertext* adalah dokumen dengan struktur yang tidak berurutan, terdiri dari berbagai bagian, yang saling berhubungan dengan menggunakan *link*. Komponen dapat langsung diakses dengan mengikuti *link*, tanpa memperhatikan kekakuan dari struktur fisik sekuensial.

Menurut Sunarto, S.Kom (2006), suatu dokumen tidak hanya dapat berbentuk tekstual tetapi juga jenis lain, seperti gambar, *video*, atau *audio* (*hypertext* multimedia, yang disingkat menjadi *hypermedia*). Beberapa dokumen yang membentuk suatu *hypermedia* dapat didistribusikan pada lebih banyak *node*. Dapat dikatakan, bahwa *world wide web* adalah *distributed* multimedia *hypertext* dengan komponen independen, karena *world wide web* menghubungkan dokumen dari berbagai jenis dan di-*maintain* oleh *subject* yang berbeda melalui internet.

Web tidak hanya mengijinkan akses ke dokumen statis tetapi juga menawarkan kesempatan untuk menjalankan program yang secara dinamis menghasilkan halaman baru. Dengan demikian, isi halaman yang baru dapat didasarkan pada isi yang di-*extract* dari *database.*

Komponen teknologi utama dari *web* adalah bahasa HTML (*HyperText Markup Language*), konsep URL (*Uniform Resource Locator*) untuk sumber daya referensi, dan HTTP (*HyperText Transfer Protocol*) *communication protocol.*

2.1.6.2 HTML (*Hypertext Markup Language*)

Menurut Atzeni *et al*, (2003, p492). Dokumen yang membentuk struktur *hypertext* dari suatu *web* ditulis dalam bentuk HTML. HTML memungkinkan untuk memformat dokumen dan menyediakan deskripsi dari *link hypertext.*

Menurut Wendy willard (2006), HTML adalah sebuah bahasa markup yang digunakan untuk membuat sebuah halaman web, menampilkan berbagai informasi di dalam sebuah penjelajahan web Internet dan formating *hypertext* sederhana yang di tulis kedalam berkas format ASCII agar dapat menghasilkan tampilan wujud yang integrasi..

Dokumen HTML dapat ditampilkan melalui *web browser*, yang dapat mengakses dokumen lokal atau dokumen yang terletak di *remote sites*. HTML menjelaskan karakteristik logis dari dokumen dan bukan hal fisik. Karakteristik ini sangat penting, karena dokumen yang sama dapat dilihat oleh banyak *client, dengan device* dan *browser* yang berbeda.

Sebuah fitur mendasar dari *hypertext* adalah menciptakan *link*, menghubungkan dokumen yang berbeda atau bagian-bagian berbeda dari dokumen yang sama. *Anchor* adalah *link* yang ditentukan untuk menghubungkan URL dengan sebagian dari teks (atau dengan gambar). URL menetapkan *resource* (misalnya, dokumen atau program) yang ditentukan oleh *anchor*. Bentuk yang paling sederhana dari URL berisi spesifikasi dari sebuah *file* lokal. Bentuk *anchor* yang berupa teks biasanya ditampilkan dengan bergaris bawah. Ketika pengguna mengklik *anchor*, *browser* memberikan *request* pada *service* sesuai dengan URL yang ditentukan.

HTML juga memungkinkan untuk membuat halaman dimana pengguna dapat memasukkan nilai-nilai parameter yang akan dikirim ke *server*. Halaman ini menggunakan struktur *form*, yang menyediakan *field*, tempat di mana pengguna dapat memasukkan data. *Action* berhubungan dengan *form*, dengan menentukan URL dari program yang harus dijalankan ketika *form* itu telah selesai diisi (dan "sudah dikumpulkan" ke *web server*). Data yang sudah dimasukkan oleh pengguna ditransmisikan ke program sebagai parameter atau *input*.

2.1.6.3 HTTP

Menurut Atzeni *et al*. (2003, p494), *World Wide Web* dioperasikan oleh HTTP *server* (disebut juga *web server*), sistem program yang menawarkan jasa ke *browser* melalui HTTP, dengan menggunakan TCP (kemudian IP) pada tingkat yang lebih rendah. HTTP terdiri dari empat

fase: atzeni, paolo et,al.database system : Concepts, languages, and architectures. Mc-grawhill. Singapore)

1. *Opening the connection*: *browser* (yang berperan sebagai *client*) menghubungi HTTP *server*, di alamat tersebut dan dengan protokol yang tercantum dalam URL untuk memverifikasikan akurasi dan ketersediaan. Secara khusus, sebuah koneksi TCP di-*request*.
2. *Establishment of the connection*: *server* menerima koneksi dan mengirimkan konfirmasi.
3. *Request*: *client* mengirimkan pesan ke *server*, dengan meminta suatu layanan, rincian sumber daya yang dipanggil dan *parameter* yang berhubungan dengan pemanggilan itu.
4. *Reply*: *server* memberitahu apakah permintaan dapat dipenuhi dan, jika demikian, pada saat yang sama, *server* mengakhiri sambungan, tanpa mempertahankan informasi yang digunakan untuk koneksi berikutnya.

HTTP *protocol* tidak memiliki memori. Jadi, ketika *client* meminta beberapa HTTP *request* ke *server* yang sama, *server* itu sendiri tidak mampu mempertahankan informasi tentang operasi dan hasil dari operasi yang sudah dilakukan oleh *client* yang sama. Pilihan ini dimotivasi oleh kesederhanaan *protocol*, yang mengelola setiap permintaan secara terpisah tanpa menyimpan catatan dari operasi tersebut.

2.2 Teori Khusus

2.2.1 Pornografi

Berdasarkan definisi pornografi dari Lin (2003, p2), halaman *web* yang mengandung citra pornografi didefinisikan sebagai halaman *web* yang di dalamnya terdapat citra yang mengandung obyek manusia telanjang, manusia menunjukkan organ seksual, atau manusia sedang melakukan hubungan seksual. Lebih jauh lagi, Lin menyebutkan bahwa situs *web* pornografi umumnya mengandung kata-kata yang mengindikasikan pornografi. (Lin et. al, 2003, p2).

Istilah lain dari pornografi bila dilacak pengertiannya secara etimologis berasal dari bahasa Yunani kuno "*porne*" yang berarti wanita jalang, dan "graphos" yang artinya gambar atau lukisan. Dalam Kamus Besar Bahasa Indonesia pornografi diartikan sebagai:

1. Penggambaran tingkah laku secara erotis dengan lukisan atau untuk membangkitkan nafsu birahi, mempunyai kecenderungan merendahkan kaum wanita.
2. Bahan yang dirancang dengan sengaja dan semata-mata untuk membangkitkan nafsu *seks.* Supartiningsih (2010, p5), berpendapat bahwa pornografi secara material menyatukan seks yang berhubungan dengan kelamin sehingga dapat menurunkan martabat atau harga diri.

Beberapa istilah yang seringkali dikaitkan dengan pornografi di antaranya adalah *pornokitsch* yang bermakna selera rendah; *obscenity* yang bermakna kecabulan, keji dan kotor, tidak senonoh, melanggar kesusilaan dan kesopanan. Bila hal-hal yang terkandung maknanya dalam pornografi ini diwujudkan melalui tindakan maka itulah yang disebut dengan pornoaksi

(Widarti, 2003, p8). FX. Rudi Gunawan (2001, p18) mengidentikkan pornoaksi dengan *sexual behaviour* atau perilaku seksual yang mencakup cara berpakaian seronok, gerak-gerik dan ekspresi wajah yang menggoda, suara yang mendesah dan majalah porno yang menampilkan gambar *nude*.

2.2.2 *Web Browser*

Menurut organisasi W3C (2004), *web browser* adalah sebuah *software* aplikasi yang berguna untuk memberikan dan mengambil sumber informasi dari *World Wide Web* (WWW). Prosesnya berawal dari pengidentifikasian sebuah sumber informasi yang dilakukan oleh *Uniform Resource Identifier* (URI), kemudian sumber informasi tersebut menjelma menjadi halaman *web* yang bisa berupa gambar, *video*, artikel, dan sebagainya. Beberapa *web browser* yang sudah memiliki nama di dunia internet adalah *Google Chrome, Internet Explorer, Safari, dan Mozilla Firefox.*

Ditambahkan dari *How Browsers Work* (2009), Fungsi utama *browser* adalah menyajikan informasi *web* tertentu, dengan cara meminta pada *server* lalu jendela *browser* akan menampilkan informasi yang dicari. Format yang paling umum terdapat pada *Web Browser* adalah HTML. Namun, terkadang ada juga format PDF pada data-data tertentu.

Di antara banyak *browser*, sebagian besar diantaranya memiliki layar antarmuka yang kurang lebih sama. Beberapa elemen layar antarmuka yang umum bagi pengguna adalah:

a. *Address Bar* untuk memasukkan URL

b. Tombol *Back* dan *Forward*

c. Pilihan untuk *Bookmark*

d. Tombol *Refresh* dan *Stop*, untuk memperbarui atau menghentikan panggilan layar *web.*

e. Tombol *Home* untuk kembali ke layar awal

Meski begitu, bukan berarti elemen-elemen di atas harus selalu diikuti dalam pembuatan layar antarmuka sebuah *browser.* Elemen-elemen di atas dianggap sebagai spesifikasi formal yang pada awalnya saling ditiru oleh para pembuat *browser* itu sendiri.

Struktur *Browser* :

1. *User Interface* : Termasuk *address bar*, tombol *back* dan *forward*, menu *bookmark* dan lainnya.
2. *Browser Engine* : Antarmuka untuk melakukan *query* dan manipulasi mesin *rendering.*
3. *Rendering Engine* : Bertanggung jawab untuk menampilkan isi yang diminta. Misalnya jika konten yang diminta adalah HTML, ia wajib untuk melakukan *parsing* antara HTML dan CSS, lalu menampilkan konten di layar.
4. *Networking* : Digunakan untuk pemanggilan jaringan seperti pemanggilan HTTP. Merupakan *Platform interface* yang berdiri sendiri dan memiliki implementasi tersendiri pada tiap *platform*-nya.
5. *UI Backend* : Berfungsi untuk menggambar sebuah *widget* seperti *Combo Boxes* dan *Windows*.

6. *Javascript Interpreter* : Digunakan untuk mengeksekusi kode *Javascript*

7. *Data Storage* : Tempat penyimpanan data. *Browser* harus menyimpan segala macam data pada *hard disk*nya sendiri. Contohnya adalah *cookie* dan *history*.

W3C (2009) mengatakan *web browser* pertama kali muncul pada tahun 1990. *Browser* ini dibuat oleh seorang pakar TI bernama Tim Berners-Lee. Lee memberikannya nama *World Wide Web* sebelum akhirnya diganti dengan nama Nexus. Namun *Web Browser* mulai dikenal masyarakat luas pada tahun 1993. *Browser* ini dibuat oleh Marc Andreesen. Marc membuat *browser* bernama Mosaic yang menjadi *Browser* pertama yang paling dikenal di dunia kala itu. Marc membuat sistem yang sangat mudah digunakan pada Mosaic, sehingga setiap orang tak perlu waktu lama untuk mempelajarinya. Tak lama kemudian nama Mosaic ia ganti menjadi *Netscape*, sesuai dengan nama perusahaannya, *Netscape Communications Corporations*. Setelah itu, barulah muncul nama-nama seperti *Microsoft Internet Explorer, Mozilla Firefox, hingga yang teranyar Google Chrome.*

2.2.3 Microsoft Visual Basic .NET

Menurut Bain (2002), *Microsoft Visual Basic* (sering disingkat sebagai VB saja) merupakan sebuah bahasa pemrograman yang bersifat *event driven* dan menawarkan *Integrated Development Environment* (IDE) *visual* untuk membuat program aplikasi berbasis sistem operasi *Microsoft Windows* dengan menggunakan model pemrograman *Common Object Model* (COM). *Visual*

Basic merupakan turunan bahasa *BASIC* dan menawarkan pengembangan aplikasi komputer berbasis grafik dengan cepat, akses ke basis data menggunakan *Data Access Objects (DAO), Remote Data Objects (RDO), atau ActiveX Data Object (ADO),* serta menawarkan pembuatan kontrol *Active*X dan obyek *Active*X.

Visual Basic merupakan turunan bahasa *BASIC* dan menawarkan pengembangan aplikasi komputer berbasis grafik dengan cepat, akses ke basis data menggunakan *Data Access Objects (DAO), Remote Data Objects (RDO),* atau *Active*X Data *Object (ADO),* serta menawarkan pembuatan kontrol *Active*X dan obyek *Active*X. Beberapa bahasa skrip seperti *Visual Basic for Applications (VBA)* dan *Visual Basic Scripting Edition (VBScript),* mirip seperti halnya *Visual Basic*, tetapi cara kerjanya yang berbeda. Para programmer dapat membangun aplikasi dengan menggunakan komponen-komponen yang disediakan oleh *Microsoft Visual Basic.* Program-program yang ditulis dengan *Visual Basic* juga dapat menggunakan *Windows* API, tapi membutuhkan deklarasi fungsi eksternal tambahan.

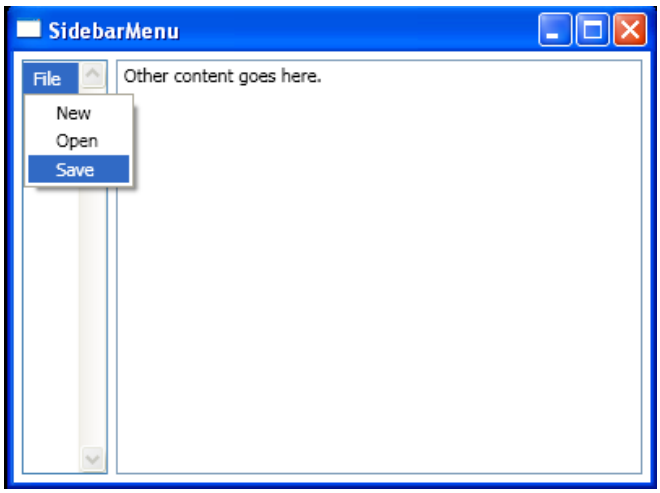
Contoh code dalam VB 2009 :

```
<Window x:Class="MenusAndToolbars.SidebarMenu"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presen
tation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="SidebarMenu" Height="300" Width="400">
    <DockPanel LastChildFill="True" Margin="5">
      <Border BorderBrush="SteelBlue" BorderThickness="1">
      <ScrollViewer DockPanel.Dock="Left">
        <Menu>
```

```
            <Menu.ItemsPanel>
              <ItemsPanelTemplate>
                <StackPanel Background="White"></StackPanel>
              </ItemsPanelTemplate>
            </Menu.ItemsPanel>

            <MenuItem Header="File">
              <MenuItem Header="New"></MenuItem>
              <MenuItem Header="Open"></MenuItem>
              <MenuItem Header="Save"></MenuItem>
            </MenuItem>
            <MenuItem Header="Help"></MenuItem>
          </Menu>
        </ScrollViewer>
        </Border>

        <TextBox Margin="5,0,0,0" TextWrapping="Wrap">Other cont
ent goes here.</TextBox>
    </DockPanel>
</Window>
```



Gambar 2.6 Hasil Dari Code VB.Net

2.2.4 *Blocking*

Dalam Kamus Besar Bahasa Indonesia, blokir memiliki arti membekukan atau menghentikan. Pemblokiran dapat terjadi pada berbagai macam hal karena bersifat universal. Menurut William Stallings (2004, p123), pemblokiran terjadi ketika suatu proses tidak menyelesaikan jalurnya dengan benar atau tertahan karena terjadi kesalahan dalam melakukan proses. Ketika suatu proses telah diblokir, ia harus menunggu hingga akses kembali didapatkan dan operasi I/O berjalan dengan semestinya.

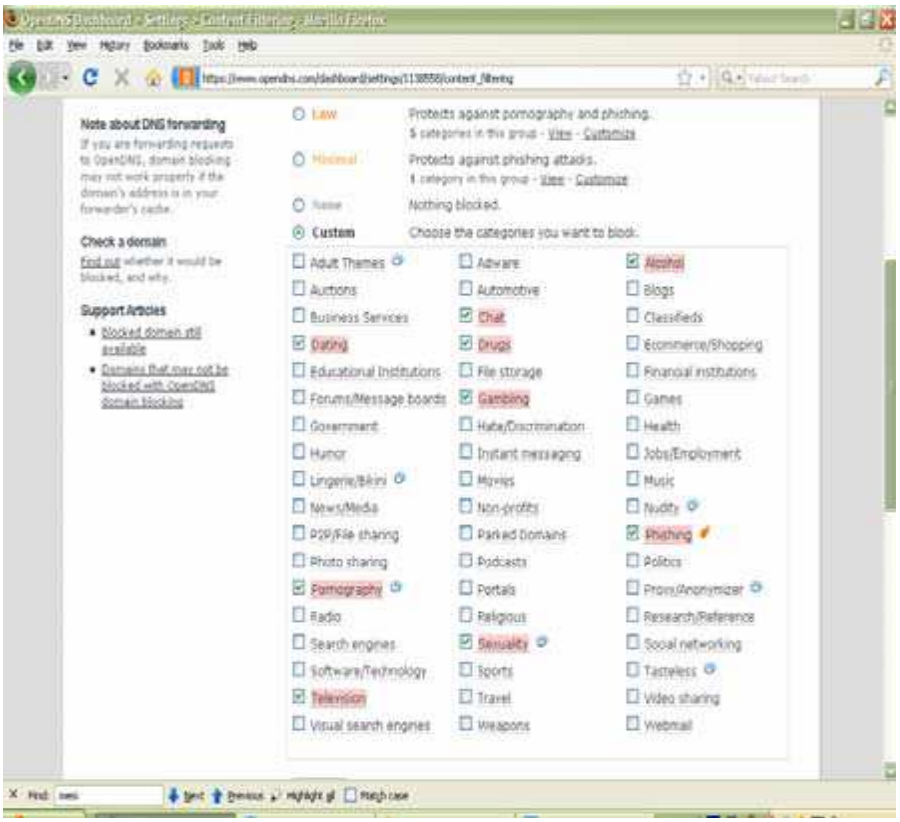
Diambil dari Ensiklopedia PCMagz, Secara teknis, ada beberapa cara untuk memblokir suatu *website*. Ada cara pemblokiran konten dengan menghambat alur lalu lintas koneksi. Mungkin saja akses ke *web* tertentu diizinkan. Namun, *user* tidak bisa melakukan transfer file dalam halaman *web* tersebut. Hal tersebut juga bisa dikatakan sebagai pemblokiran. Suatu konten juga dapat diblokir oleh suatu situs. Biasanya situs memiliki daftar katalog URL yang sekiranya tidak boleh disinggahi.

Macam-macam metode pemblokiran menurut Muhammad Sholeh (2010) adalah:

1. ***OpenDNS***

   Proses pemblokiran *openDNS* hanya bisa digunakan pada pengguna tunggal dan berlaku hanya untuk satu komputer saja. Proses pemblokiran dapat dilakukan pada *proxy*. Pemblokiran dengan *proxy* berlaku untuk semua pengguna yang menggunakan *proxy* yang telah di*setting*. Pemblokiran dengan *proxy* sangat penting terutama untuk komputer yang ada di kantor, sekolah ataupun warung internet.

Layanan blokir ini bersifat gratis dan *online*. Pemakai dapat melakukan pendaftaran untuk mengaktifkan layanan ini. Dalam proses memblokiran ini, banyak jenis pilihan yang dapat ditentukan untuk mendefinisikan situs yang dikategorikan dalam larangan pengaksesan. Hasil konfigurasi tersebut, akan dilakukan proses pengecekan dari setiap ada pengguna yang melakukan pengaksesan suatu *web*. Jika *web* yang diakses termasuk kategori yang diblok, maka *website* tersebut tidak akan diteruskan.

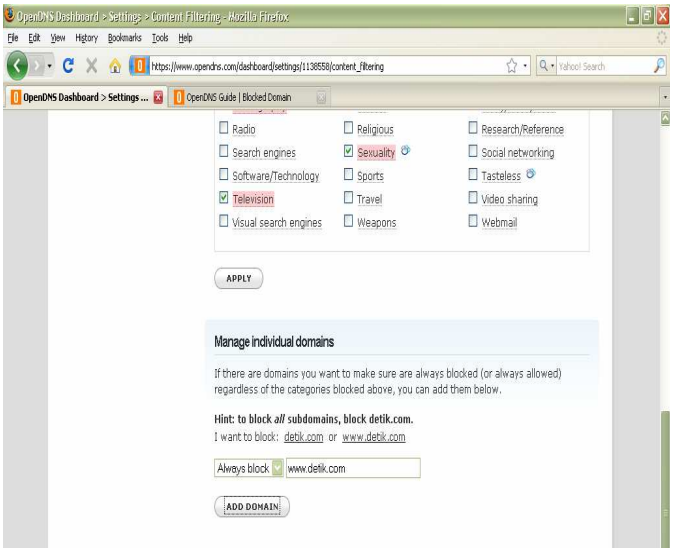


Gambar 2.7 Pemblokiran dengan *OpenDNS*

2. Blokir Nama Domain Tertentu.

Seiring dengan makin banyaknya pengguna Internet, *openDNS* kadang tidak bisa memfilter nama domain yang sebenarnya masuk

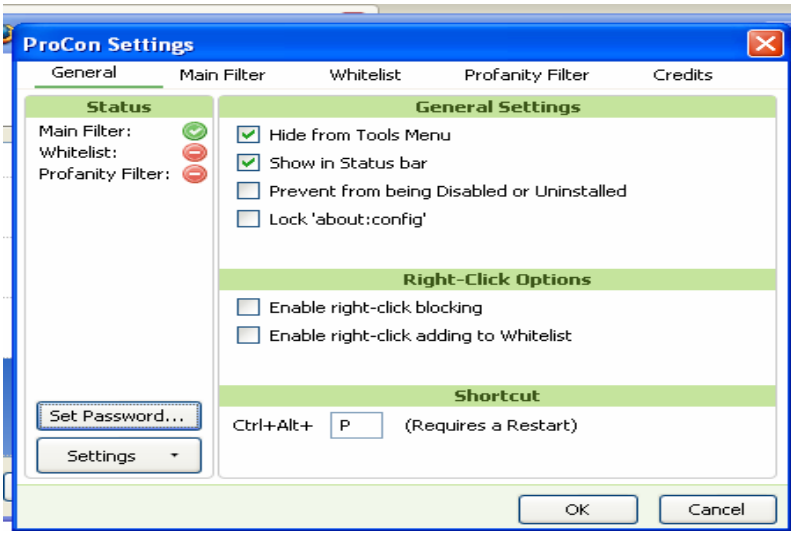
kategori dilarang. Hal ini disebabkan nama domain tersebut baru dan belum masuk di basis data *openDNS*. Antisipasi untuk nama domain yang belum masuk ini, dapat dilakukan dengan membuat basisdata sendiri yang berisi domain-domain yang dikategorikan dilarang. Didalam *openDNS* sendiri sebenarnya disediakan fasilitas untuk menambahkan nama domain yang dikategorikan dilarang. Disamping menggunakan *openDNS* memblokiran nama domain dapat juga dilakukan dengan memanfaatkan fasilitas *file host* yang ada di dalam *Window*s XP. Proses pemblokiran dengan *file Host* ini dapat dilakukan karena setiap pengaksesan internet, *Browser* akan mengirimkan *request* ke sebuah server DNS dan *server* tersebut akan mencari alamat IP yang tepat dan kemudian mengirim alamat IP tersebut ke *Browser*. Pada saat mengakses dengan mengetikkan alamat *website, Window*s XP akan mencari data pada *DNS cache* untuk melihat apakah terdapat informasi DNS. Jika terdapat informasi tersebut maka *Window*s XP tidak akan mengirimkan *request* data ke *DNS* tujuan untuk mendapatkan alamat IP-nya. Alamat *website* dapat diakses lewat *DNS cache* ini. Data *cache* tersebut dibuat berdasarkan informasi yang terdapat dalam *file host.*



Gambar 2.8 Pemblokiran dengan *OpenDNS* berdasarkan Nama Domain Tertentu

3. Aplikasi *ProCon Latte*

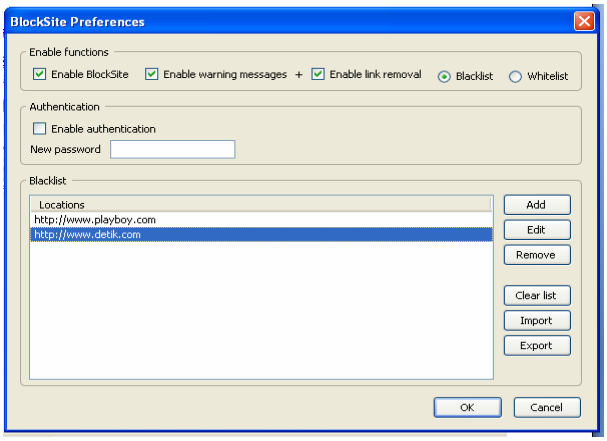
ProCon Latte, adalah sebuah *add-on content filter Browser Mozilla firefox* yang dapat menyaring segala jenis situs-situs berbahaya (pornografi, judi, *hacking, cracking*, dan sebagainya), juga dapat memblokir semua lalu lintas data lainya. *ProCon* juga memiliki proteksi *password* dan tersedia *White List* pada aplikasi ini yang dapat diakses pada https://addons.mozilla.org/id/firefox/addon/1803. *Support Firefox*: 2.0 – 3.5. Agar *software procon latte* dapat digunakan di dalam *Browser*, khususnya *Browser mozilla*, pengguna harus melakukan proses penginstalan aplikasi tersebut dan melakukan proses konfigurasi.



Gambar 2.9 Pemblokiran menggunakan Procon Latte

4. *Blocksite*

Blocksite berbeda dengan *procon*, aplikasi *blocksite*, digunakan untuk memblok situs-situs yang kita inginkan. *Add-on* ini tidak difungsikan sebagai *parental control* utama. Langkah awal dalam pemakaian aplikasi *blocksite* adalah melakukan proses instalasi dan proses pengaturan konfigurasi, terutama halaman-halaman situs yang dilarang.



Gambar 2.10 Pemblokiran Menggunakan *Blocksite*

2.2.5 *Regular Expression*

Tahun 1956, Stehpen Cole Klenee seorang ahli matematika membuat sebuah model *pattern string* menggunakan notasi matematika, yang disebut *regular sets.* Kemudian Ken Thompson mengimplementasikan notasi tersebut ke dalam *text* editor QED buatannya untuk pencarian string dengan pola tertentu. Dia juga menambahkan fitur tersebut ke dalam, sebuah *text* editor dalam sistem operasi *UNIX* yang dikenal dengan ED. Untuk pencarian *string* dalam ED digunakan *pattern* : /g /re /p (/g : *globally*, /re : *regular expression*, /p : *print*) yang maksudnya adalah pencarian global baris-baris dalam sebuah *file* yang memiliki pola tertentu, dan ditampilkan atau di cetak. Sejak saat itu *regular expression* mulai banyak digunakan.

Regular expression atau yang sering disebut sebagai *regex* adalah sebuah formula untuk pencarian pola suatu kalimat/*string* (Agus Muliantara, 2009). Sering kali orang beranggapan bahwa *regex* susah dan membingungkan. Namun sebenarnya *regex* sangatlah membantu dalam menemukan pola-pola kalimat. Sehingga percobaan terhadap semua kemungkinan pola kalimat tidak perlu dilakukan.

Regular Expression umumnya digunakan oleh banyak pengolah kata/*text* editor dan peralatan lainnya untuk mencari dan memanipulasi kalimat dengan berdasarkan kepada suatu pola tertentu. Banyak bahasa pemrograman yang mendukung *regular expression* seperti misalnya *PHP, perl, VB* dan *Tcl.*

Sebuah alasan yang sangat bagus untuk menggunakan *regex* adalah karena *regex* sangatlah *powerfull.* Pada *level* rendah, *regex* dapat mencari

sebuah penggalan kata. Pada *level* tinggi *regex* mampu melakukan kontrol terhadap data. Baik mencari, menghapus dan merubah.

Konsep pola regex ada tiga bagian yaitu *literal character*, *metacharachter* dan *escape sequence*.

1. *Literatur character* adalah sebuah karakter (atau angka) yang digunakan dalam pencarian. Misalnya, string “ab” dapat ditemukan pada kata/string “**ab**u”, “r**ab**u”, dan “seb**ab**”.

2. *Metacharacter* adalah sebuah karakter spesial yang mempunyai fungsi yang unik. Misalnya adalah karakter “^”.

3. *Escape Sequence* pengertiannta adalah karakter “\” yang memiliki fungsi untuk mengubah fungsi dari metacharacter menjadi fungsi *literal character*. Biasanya karaktern ini ditempatkan di depan *metacharacter*tersebut.

Mari kita pikirkan bagaimana cara untuk mencari sebuah *file* di *hard disk.* Seringkali digunakan karakter “?” dan “*”. Penggunaan karakter “?” mengandung arti bahwa sedang dicari sebuah sebuah *file* yang mengandung sebuah karakter tertentu dan karakter “*” mengandung arti sedang dicari nol atau lebih karakter. Sebagai contoh : pencarian dengan menggunakan pola “*file?.dat*” akan menghasilkan beberapa contoh sebagai berikut :

- *file1.dat*
- *file2.dat*
- *file3.dat*
- *file.dat*

• *fileN.dat*

Menggunakan karakter "*" akan menghasilkan lebih banyak hasil pencarian. "*file*.dat*" akan menghasilkan :

• *file1.dat*

• *file2.dat*

• *file12.dat*

• *filex.dat*

• *fileXYZ.dat*

Kedua contoh di atas adalah salah satu penggunaan *regular expression.* Bayangkan saja jika pencarian *file* hanya menggunakan metode nama *file* tanpa menyertakan *regular expression.* Tentunya saat proses pencarian menggunakan pola "*file*.dat*" hasilnya adalah harus "*file*.dat*". Hal inilah yang menyebabkan *regex* menjadi sangat *powerfull*. Dengan menggunakan sebuah pola didapatkan hasil yang luas.

Beberapa pola yang umum digunakan dalam *regex* tampak pada tabel 2.7 berikut.

Tabel 2.7 Pola Umum Pada *Regex*

Pola	Penjelasan
[ ]	ekspresi kurung. cocok dengan satu karakter yang berada dalam kurung, misal: pattern "a[bcd]i" cocok dengan string "abi", "aci", dan "adi". penggunaan range huruf dalam kurung diperbolehkan, misal : *pattern* "[a-z]" cocok dengan salah satu karakter diantara *string* "a" sampai "z". *pattern* [0-9] cocok dengan salah satu angka. Jika ingin mencari karakter "-" juga, karakter tersebut harus diletakkan di depan atau di belakang kelompok, misal: "[abc-]".
[^]	cocok dengan sebuah karakter yang tidak ada dalam kurung, berlawanan dengan yang diatas. misal: *pattern* "[^abc]" cocok dengan satu karakter apa saja kecuali "a", "b", "c".
?	cocok dengan nol atau satu karakter sebelumnya. misal: *pattern* "*died*?" cocok dengan *string "die"* dan *"died"*.
+	cocok dengan satu atau lebih karakter sebelumnya. misal: "yu+k" cocok dengan "yuk", "yuuk", "yuuuk", dan seterusnya.
*	cocok dengan nol atau lebih karakter sebelumnya. misal: *pattern* "hu*p" cocok dengan *string* "hp", "hup", "huup" dan seterusnya.
{ x }	cocok dengan karakter sebelumnya sejumlah x karakter. misal: *pattern* "[0- 9]{3}" cocok dengan bilangan berapa saja yang berukuran 3 digit.
{x,y}	cocok dengan karakter sebelumnya sejumlah x hingga y karakter. misal:

	pattern "[a-z]{3,5}" cocok dengan semua susunan huruf kecil yang terdiri dari 3 sampai 5 huruf.
!	Jika diletakkan di depan *pattern*, maka berarti "bukan". misal *pattern* "!a.u" cocok dengan *string* apa saja kecuali *string* "alu", "anu", "abu", "asu", "aiu", dan seterusnya.
^	Jika diletakkan di depan *pattern*, akan cocok dengan awal sebuah *string*.
$	Jika diletakkan di belakang *pattern*, akan cocok dengan akhir sebuah *string*
( )	*Grouping*. digunakan untuk mengelompokkan karakter-karakter menjadi *single* unit. *string* yang cocok dalan *pattern* yang berada dalam tanda kurung dapat digunakan pada operasi berikutnya, semacam variabel.
\	*escape character* mengembalikan fungsi *metacharacter* menjadi karakter biasa. Pada beberapa sistem dapat berarti sebaliknya, yaitu *metacharacter* menggunakan *escape character* didepannya.

2.2.5 Konsep *File Based System*

Menurut Thomas Connolly dan Carolyn Begg (2002, p7), "*A collection of application programs that perform services for the end-users such as the production of reports. Each program defines and manages its own data*". Artinya adalah suatu kumpulan aplikasi program yamg menyediakan *service* bagi *end-user* seperti pembuatan laporan. Setiap program mendefinisikan dan mengatur datanya sendiri. Kelemahan *file based system* (Connolly dan Begg, 2002, p12):

1. Data terpisah dan terisolasi.
2. Duplikasi Data.
3. Memakan banyak waktu.
4. Memerlukan *storage* yang besar.
5. Data tidak lagi konsisten.
6. Data *Dependence*.
7. *Incompatible File* Format.
8. *Fixed Queries of Application* Program.

Suatu *form* dapat diuraikan menjadi empat bagian kamus data yang menerangkan isi dari sistem yang berbentuk seperti tabel dibawah ini :

Tabel 2.8 Contoh Kamus Data

Data Flow Dictionary Entry	*Data Store Dictionary Entry*
Data Structure Dictionary Entry	
Data Element Dictionary Entry	

Tabel di atas menunjukkan bahwa *data flow* dan *data store* ada pada level tertinggi. Di sini lebih baik menganggap *data flow* dan *data store* sebagai *file* dari data. Selanjutnya struktur data yang ada pada *data flow* dan *data store* terletak pada level kedua. Di sini struktur data dianggap sebagai *record data*.

Yang terakhir adalah data elemen yang terletak pada level terendah, karena data element merupakan bagian dari struktur data. Di sini data element dianggap sebagai *field.*