BAB II

TINJAUAN PUSTAKA

II.1. Definisi Perancangan

Perancangan adalah suatu kegiatan yang memiliki tujuan untuk mendesign sistem baru yang dapat menyelesaikan masalah-masalah yang dihadapi perusahaan yang diperoleh dari pemilihan alternatif sistem yang terbaik. Jadi kesimpulannya perancangan itu adalah suatu pola yang dibuat untuk mengatasi masalah yang dihadapi perusahaan atau organisasi setelah melakukan analisis terlebih dahulu.

II.2. Animasi

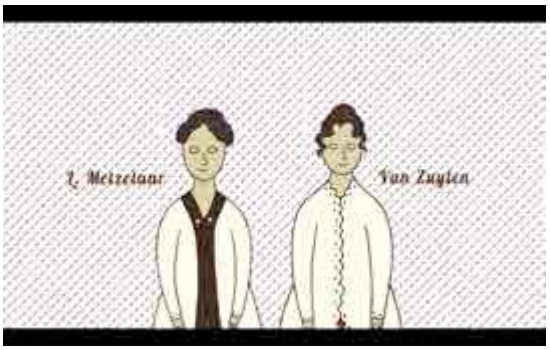
Animasi merupakan kumpulan gambar yang diolah sedemikian rupa sehingga menghasilkan gerakan. Animasi mewujudkan ilusi (illusion) bagi pergerakan dengan memaparkan atau menampilkan satu urutan gambar yang berubah sedikit demi sedikit (Progressively) pada kecepatan yang tinggi. Animasi digunakan untuk memberi gambaran pergerakan bagi suatu objek (Agus Suheri, 2006; 2-3).

II.2.1. Jenis-jenis Animasi

Menurut Yunita Syahfitri (2011, 3), animasi yang dulunya mempunyai prinsip yang sederhana, sekarang telah berkembang menjadi beberapa jenis, yaitu animasi 2D, animasi 3D dan animasi tanah liat.

1. Animasi 2D (Dua Dimensi)

Animasi ini yang paling akrab dengan keseharian kita. Biasa disebut juga dengan film kartun. Kartun sendiri berasal dari *Cartoon*, yang berarti gambar yang lucu. Memang, film kartun ini kebanyakan film yang baru.



Gambar II.1. Animasi 2D

(Sumber : Riri Rosyidah, dkk, 2014; 7)

2. Animasi 3D (Tiga Dimensi)

Perkembangan teknologi dan dunia komputer membuat teknik pembuatan animasi 3D semakin berkembang dan maju pesat. Animasi 3D adalah perkembangan dari animasi 2D. Dengan animasi 3D, karakter yang diperlihatkan semakin hidup dan nyata, mendekati wujud aslinya.

Gambar II.2. Animasi 3D

(Sumber : Tristiariena Utami, 2012; 11)

3. Animasi Tanah Liat (Clay Animation)

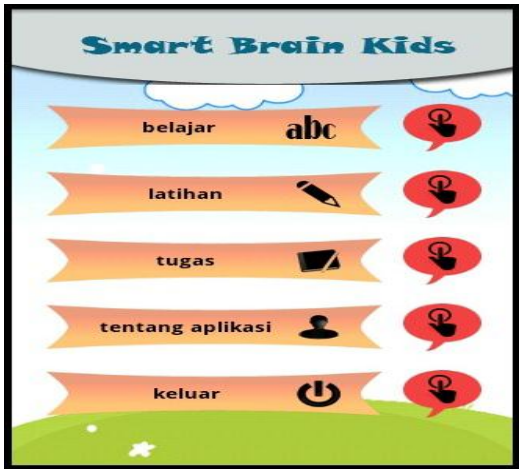
Meski namanya *Clay* (tanah liat), namun yang dipakai bukanlah tanah liat biasa. Animasi ini menggunakan *Palsticin*, bahan lentur seperti permen karet yang ditemukan pada tahun 1987. Tokoh-tokoh pada animasi *Clay* dibuat dengan menggunakan rangka yang khusus untuk kerangka tubuhnya. Film animasi *Clay* pertama kali dirilis bulan Februari 1908 berjudul, *A Sculptor's Web Rarebit Nightmare*. Umtuk beberapa waktu yang lalu juga, beredar film *Clay* yang berjudul *Chicken Run*.

II.3. Game

Dalam bahasa Indonesia "Game" berarti "permainan". Permainan yang dimaksud dalam game juga merujuk pada pengertian sebagai "kelincahan intelektual" *(Intellectual Playability)*. Sementara kata "game" bisa diartikan sebagai arena keputusan dan aksi pemainnya. Ada target-target yang ingin dicapai pemainnya. Kelincahan intelektual pada tingkat tertentu, merupakan ukuran sejauh mana game itu menarik untuk dimainkan secara maksimal (Suindarti, 2011; 2).

II.3.1. Game Edukasi

Menurut Andriansyah (2014; 2), Game edukasi adalah *game* digital yang dirancang untuk pengayaan pendidikan (mendukung pengajaran dan pembelajaran), menggunakan teknologi multimedia interaktif.



Gambar II.3. Game Edukasi Smart Brain Kids

(Sumber : Arif Dwi Sutanto, 2013; 11)

Menurut Hurd dan Jenuings, perancang yang baik haruslah memenuhi kriteria dari *education game* itu sendiri (Nelly Indriani Widiastuti, Irwan Setiawan. *Membangun Game Edukasi Walisongo*. 2012). Berikut ini adalah beberapa kriteria dari sebuah education game, yaitu :

1. Nilai keseluruhan (*Overall Value*)

   Nilai keseluruhan dari suatu *game* terpusat pada desain dan panjang durasi *game*. Aplikasi ini dibangun dengan desain yang menarik dan interaktif. Untuk penentuan panjang durasi, aplikasi ini menggunakan fitur timer.

2. Dapat digunakan (*Usability*)

   Mudah digunakan dan diakses adalah poin penting bagi pembuat *game*. Aplikasi ini merancang sistem dengan *interface* yang *user friendly* sehingga *user* dengan mudah dapat mengakses aplikasi.

3. Keakuratan (*Accuracy*)

   Keakuratan diartikan sebagai bagaimana kesuksesan model/gambaran sebuah game dapat dituangkan ke dalam percobaan atau perancangannya. Perancangan aplikasi ini harus sesuai dengan model *game* pada tahap perencanaan.

4. Kesesuaian (*Appropriateness*)

   Kesesuaian dapat diartikan bagaimana isi dan desain *game* dapat diadaptasikan terhadap keperluan user dengan baik. Aplikasi ini menyediakan menu dan fitur yang diperlukan *user* untuk membantu pemahaman user dalam menggunakan aplikasi.

5. Relevan (*Relevance*)

   Relevan artinya dapat mengaplikasikan isi *game* ke target *user*. Agar dapat relevan terhadap *user*, sistem harus membimbing mereka dalam pencapaian tujuan pembelajaran. Karena aplikasi ini ditujukan untuk anak-anak, maka desain antarmuka harus sesuai dengan nuansa anak-anak, yaitu menampilkan warna-warna yang ceria.

6. Objektifitas (*Objectives*)

   Objektifitas menentukan tujuan user dan kriteria dari kesuksesan atau kegagalan. Dalam aplikasi ini objektifitas adalah usaha untuk mempelajari hasil dari permainan.

7. Umpan Balik (*Feedback*)

   Untuk membantu pemahaman *user* bahwa permainan (*performance*) mereka sesuai dengan objek *game* atau tidak, *feedback* harus disediakan. Aplikasi ini

menyajikan animasi dan efek suara yang mengindikasikan kesuksesan atau kegagalan permainan.

II.3.2. Jenis-Jenis Game

Menurut Suindarti (2011, 3), jenis-jenis game sebagai berikut :

1. ***Shooting*** **(tembak-tembakan)** : Video game jenis sangat memerlukan kecepatan refleks, koordinasi mata-tangan, juga timing, inti dari game jenis adalah tembak, tembak dan tembak. Contoh : GTA, dan Crysis.
2. ***Fighting*** **(Pertarungan)** : Game yang permainannya memerlukan refleks dan koordinasi mata dan tangan dengan cepat, tetapi inti dari game ini adalah penguasaan hafalan jurus. Contoh : Mortal Kombat dan Tekken.
3. **Petualangan (*Adventure*)** : Game yang lebih menekankan pada jalan cerita dan kemampuan berfikir pemain dalam menganalisa tempat secara visual, memecahkan teka-teki maupun menyimpulkan berbagai peristiwa. Contoh : Kings Quest dan Space Quest.
4. **Simulasi, Konstruksi, Manajemen** : Video game jenis ini seringkali menggambarkan dunia di dalamnya sedekat mungkin dengan dunia nyata dan memperhatikan dengan detil berbagai faktor. Contoh : The Sims.
5. **Strategi** : Game jenis ini memerlukan koordinasi dan strategi dalam memainkan permainan ini. Kebanyakan game strategi adalah game perang. Contoh : Warcraft.

6. **Olahraga (*Sport*)** : Game ini merupakan adaptasi dari kenyataan, membutuhkan kelincahan dan juga strategi dalam memainkannya. Contoh : Winning Eleven dan NBA.
7. ***Puzzle*** : Game teka-teki, pemain diharuskan memecahkan teka-teki dalam game tersebut. Contoh : Minesweeper, Tetris dan Bejeweld.
8. ***Edugames*** **(Edukasi)** : Video game jenis ini dibuat dengan tujuan spesifik sebagai alat pendidikan, entah untuk belajar mengenal warna untuk balita, mengenal huruf dan angka, matematika, sampai belajar baha asing. Developer yang membuatnya , harus memperhitungkan berbagai hal agar game ini benar-benar dapat mendidik, menambah pengetahuan dan meningkatkan ketrampilan yang memainkannya. Target segmentasi pemain harus pula disesuaikan dengan tingkat kesuliatan dan design visual ataupun animasinya. Contoh edugames : Bobi bola, Dora the explorer, Petualangan Billy dan Tracy.

II.4. Linear Congruent Method

Karena komputer yang diprogram untuk melaksanakan instruksi set arithmatic, mungkin banyak yang bertanya-tanya bagaimana mungkin untuk mendapatkan nomor acak dari komputer. Sebenarnya, hal itu tidak mungkin. Namun, dengan kecerdikan dan perhatian, program dapat dirancang untuk menghasilkan bilangan acak yaitu, angka yang berperilaku, untuk tujuan praktis, seolah-olah mereka secara acak. Sebuah metode yang sangat umum menghasilkan bilangan acak pada komputer disebut linear congruent (Eric A. Suess, 2010).

Menurut Dian Sekarsari (2014; 2), *Linear Congruent Method* (LCM) merupakan metode pembangkit bilangan acak yang banyak digunakan dalam program komputer. LCM memanfaatkan model *linear* untuk membangkitkan bilangan acak yang didefinisikan dengan :

$$\boldsymbol{I(n+1) = (aI(n) + c)\ mod\ m}$$

Dimana :

n = adalah bilangan acak ke n.

a dan c adalah konstanta linear Congruent Method.

m adalah batas maksimum bilangan acak.

Keunggulan dari algoritma ini adalah kecepatannya yang baik, dikarenakan operasi yang dilakukan hanyalah beberapa operasi manipulasi bit saja.

II.4.1. Pencarian Linear Congruent Method

Linear Congruent Method (LCM) merupakan metode pembangkit bilangan acak yang banyak digunakan dalam program komputer. LCM memanfaatkan model linear untuk membangkitkan bilangan acak yang didefinisikan dengan :

$$\boldsymbol{X_n + 1 = (aX_n + c)(mod\ m)}$$

Dimana :

Xn = adalah bilangan acak ke n

a dan c adalah konstanta LCM

m adalah batas maksimum bilangan acak

Ketentuan-ketentuan pemilihan setiap parameter pada persamaan di atas adalah sebagai berikut :

a. m = modulus, $0 < m$
b. a = multiplier (pengganda), $0<a<m$
c. c = Increment (pertambahan nilai), $0 \leq c<m$
d. X_0 = nilai awal, $0 \leq X_0 < m$
e. c dan m merupakan bilangan prima relatif
f. a – 1 dapat dibagi oleh faktor prima dari m
g. a – 1 merupakan kelipatan 4 jika m juga kelipatan 4
h. a harus sangat besar

Ciri khas dari LCM adalah terjadi pengulangan pada periode waktu tertentu atau setelah sekian kali pembangkitan, hal ini adalah salah satu sifat dari metode ini, *pseudo random generator* pada umumnya. Penentuan konstanta LCM (a, c dan m) sangat menentukan baik tidaknya bilangan acak yang diperoleh dalam arti memperoleh bilangan acak yang seakan-akan tidak terjadi pengulangan.

II.5. Visual Basic

Visual Basic diturunkan dari bahasa *BASIC*, *visual basic* terkenal sebagai bahasa pemrograman yang mudah digunakan terutama untuk membuat aplikasi yang berjalan di atas *platform windows*.

Pada tahun 90an, *Visual Basic* menjadi bahasa pemrograman yang paling populer dan menjadi pilihan utama untuk mengembangkan program bahasa

Windows. Versi *Visual Basic* terakhir sebelum berjalan di atas *.NET Framework* adalah VB6 (Visual Basic 1998).

Visual Basic .NET dirilis pada bulan Februari tahun 2002 bersamaan platform *.NET Framework* 1.0. Kini sudah ada beberapa versi dari *Visual Basic* yang berjalan pada platform *.NET*, yaitu VB 2002 (VB7), VB 2005 (VB8). VB 2008 (VB9), dan yang terakhir adalah VB 2010 yang dirilis bersamaan dengan visual studio 2010 (Nurullah, 2012: 20).

II.5.1. .NET *Framework*

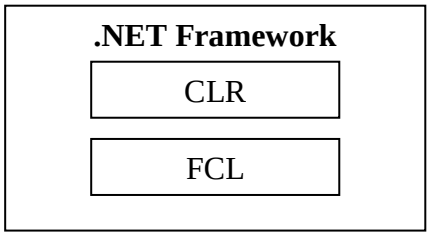
Pengertian dari .NET Framework adalah lingkungan komputasi baru (platform tunggal) yang menyederhanakan proses pembuatan aplikasi pada lingkungan terdistribusi di internet (Yuswanto, 2008 : 2).

Microsoft menciptakan *.NET Framework* dengan beberapa tujuan, antara lain :

- Lingkungan pengembangan program yang multibahasa : *.NET Framework* mendukung beberapa bahasa pemrograman seperti VB, C#, C++, *IronPhyton*, dan lain-lain. Namun semua bahasa tersebut akan digabungkan menjadi assembly yang bernama MSIL (*Microsoft Intermediate Language*).
- Mendukung penuh konsep *object oriented programming* : semua yang ada di *.NET Framework* adalah *object*.
- Menyederhanakan pemrograman pada platform windows.

- Menggunakan managed code sehingga penulisan code menjadi lebih aman. Sebagai contoh, anda tidak perlu lagi membuang *object* secara manual karena sudah ditangani oleh komponen *Garbage Collection*.

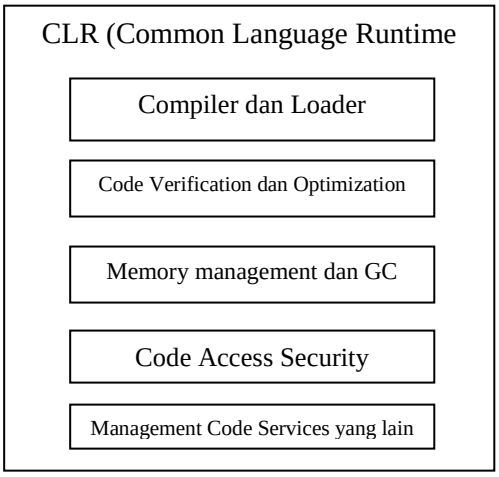
.*NET Framework* sebenarnya terdiri dari dua komponen utama, yaitu CLR *(Common Language Runtime)* dan FCL *(Framework Class Library)*.



Gambar II.4. Komponen Utama .NET Framework.

(Sumber : Kurniawan Erick, 2011 : 1-2)

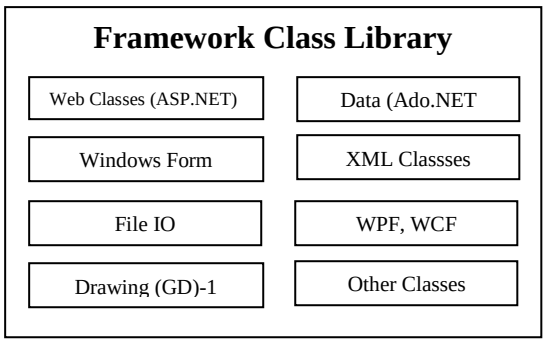
Common Language Runtime (CLR) adalah pondasi utama dari .NET Framework. CLR merupakan komponen yang bertanggung jawab terhadap berbagai macam hal, seperti melakukan managemen memory, melakukan eksekusi kode, melakukan verifikasi terhadap keamanan kode, dan berbagai layanan sistem lainnya.



Gambar II.5. Komponen CLR

(Sumber : Kurniawan Erick, 2011 : 2-3)

FCL (.*NET Framework Class Library*) atau sering disebut juga BCL (*Base Case Library*) adalah koleksi dari reusable types yang sangat banyak dan terintegrasi secara melekat dengan CLR. Kumpulan *Class Library* ini sangat berguna untuk pengembang aplikasi karena pengembang tidak perlu membuat semuanya dari awal karena sudah disediakan oleh .NET, misal *Class* untuk membuat aplikasi berbasis *windows*, *Class* untuk membuat objek-objek koleksi, *Class* untuk koneksi dengan database (ADO.NET), *Class* untuk mengembangkan aplikasi berbasis web, *Class* WPF (*Windows Presentation Foundation*), dan masih banyak lagi.



Gambar II.6. Komponen FCL

(Sumber : Kurniawan Erick, 2011 : 3)

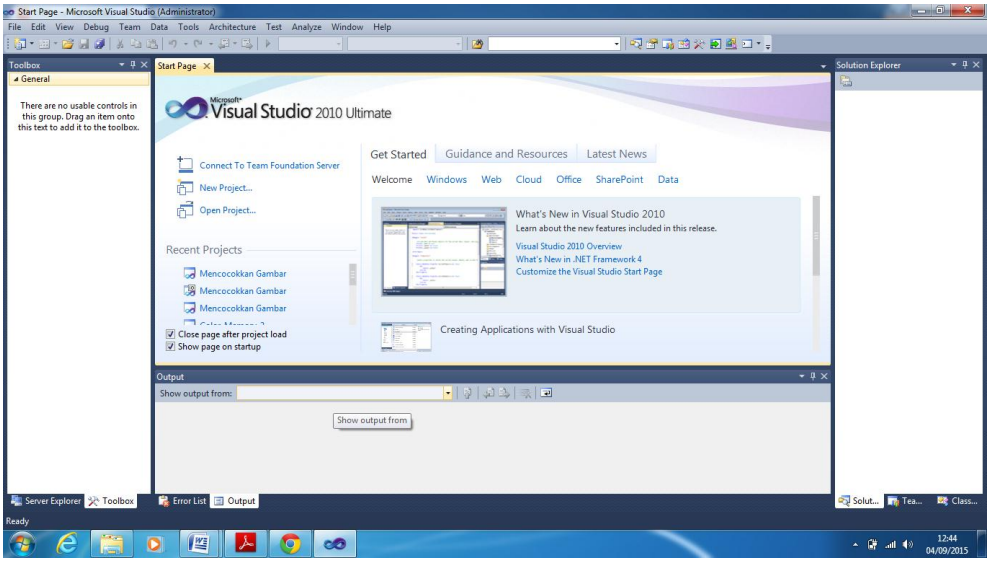
Saat ini, .NET Framework sudah mencapai versi 4.0. Pada versi yang keempat ini, .NET Framework menambahkan beberapa fitur, penambahan fitur-fitur tersebut meliputi :

- Penambahan fitur CLR : fitur baru pada VB10 dan C# 4.0.
- Penambahan language baru F#, Iron Phyton, dan Iron Ruby.
- *Data Access Improvement* (*Entity Framework 4.0* dan *Data Services 1.5*).
- *Services* dan *Workflow Technology*.
- Penambahan fitur pada ASP.NET : Web Form 4.0 (ClientII), Routing, Ajax 4.0 (*Client Templates*, *Data Context*), dan ASP.NET MVC.
- Penambahan fitur pada WPF : WPF 4.0 (*Datagrid*, *Ribbon*, *Multitouch Windows 7 Enhancements*), MEF (*Managed Extensibility Framework*).
- *Client Server* : WCF 4.0.
- *Paralel Computing* : PLINQ

II.5.2. Visual Studio 2010

Untuk menulis program menggunakan Visual Studio 2010 sebenarnya tidak harus menggunakan IDE (Integrated Development Environtment) seperti Visual Studio 2010 bertujuan untuk memudahkan dalam pengembangan aplikasi dengan cara yang lebih cepat sehingga dapat meningkatkan produktivitas.

Berikut adalah gambar tampilan dari Visual Studio 2010 saat pertama kali dijalankan.

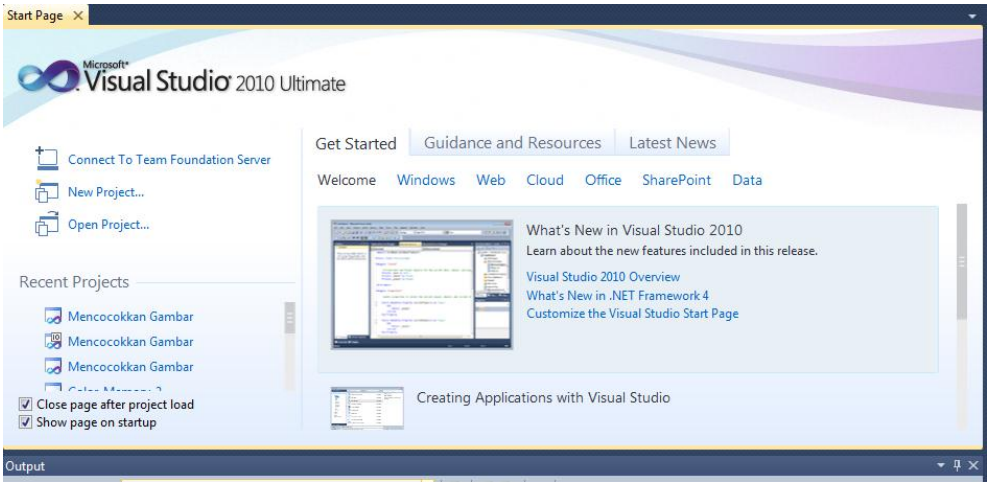


Gambar II.7. Halaman Start Up

(Sumber : Kurniawan Erick, 2011 : 9-10)

II.5.3. Menjalankan Visual Studio 2010

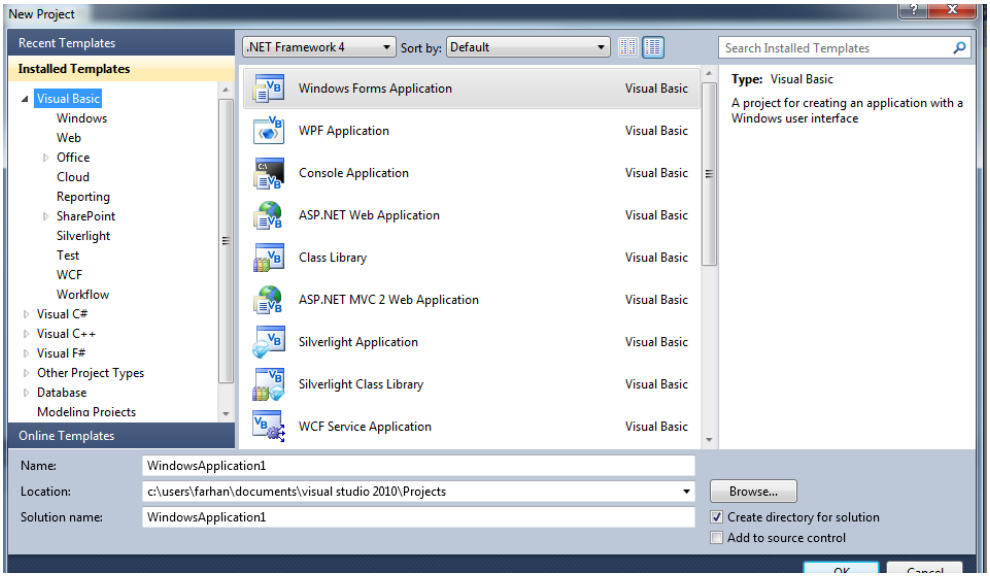
Ada beberapa tahap untuk dapat menjalankan Visual Studio 2010 menggunakan Visual Studio 2010 antara lain dengan memilih menu New Project pada halaman startup, atau dapat memilih menu File > New Project untuk menampilkan jendela baru berisi daftar pilihan dari projek yang dapat dibuat.



Gambar II.8. Membuka Proyek Baru di Visual Studio 2010

(Sumber : Kurniawan Erick, 2011 : 12)

Pilih *Visual Basic* sebagai bahasa pemrograman yang akan digunakan untuk mengembangkan program, kita juga dapat memilih bahasa lain untuk mengembangkan program di *Visual Studio* 2010.



Gambar II.9. Memilih Bahasa Pemrograman

(Sumber : Kurniawan Erick, 2011 : 12)

II.6. UML *(Unified Modelling Language)*

Unified Modelling Language (UML) adalah sebuah "bahasa" yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa-bahasa berorientasi objek seperti C++, Java, C# atau VB.NET. Walaupun demikian, UML tetap dapat digunakan untuk modeling aplikasi prosedural dalam VB atau C. Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan *syntax*/semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML *syntax* mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya: Grady Booch OOD (*Object-Oriented Design*), Jim Rumbaugh OMT (*Object Modeling Technique*), dan Ivar Jacobson OOSE (*Object-Oriented Software Engineering*). Sejarah UML sendiri cukup panjang. Sampai era tahun 1990 seperti kita ketahui puluhan metodologi pemodelan berorientasi objek telah bermunculan di dunia. Diantaranya adalah: *metodologi booch, metodologi coad, metodologi OOSE, metodologi OMT, metodologi shlaer-mellor, metodologi wirfs-brock*, dsb. Masa itu terkenal dengan

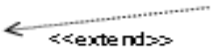
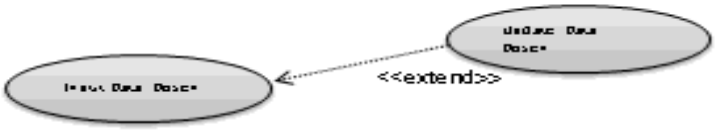
masa perang metodologi (*method war*) dalam pendesainan berorientasi objek. Masing-masing metodologi membawa notasi sendiri-sendiri, yang mengakibatkan timbul masalah baru apabila kita bekerjasama dengan group/perusahaan lain yang menggunakan metodologi yang berlainan. Dimulai pada bulan Oktober 1994 *Booch, Rumbaugh dan Jacobson*, yang merupakan tiga tokoh yang boleh dikata metodologinya banyak digunakan mempelopori usaha untuk penyatuan metodologi pendesainan berorientasi objek. Pada tahun 1995 direlease *draft* pertama dari UML (versi 0.8). Sejak tahun 1996 pengembangan tersebut dikoordinasikan oleh *Object Management Group* (OMG – http://www.omg.org). Tahun 1997 UML versi 1.1 muncul, dan saat ini versi terbaru adalah versi 1.5 yang dirilis bulan Maret 2003. *Booch, Rumbaugh* dan *Jacobson* menyusun tiga buku serial tentang UML pada tahun 1999. Sejak saat itulah UML telah menjelma menjadi standar bahasa pemodelan untuk aplikasi berorientasi objek. *(Yuni Sugiarti ; 2013 : 33)*

Dalam pembuatan skripsi ini penulis menggunakan diagram *Use Case* yang terdapat di dalam UML. Adapun maksud dari *Use Case* Diagram diterangkan dibawah ini.

1. *Use Case Diagram*

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah “apa” yang diperbuat sistem, dan bukan “bagaimana”. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login ke

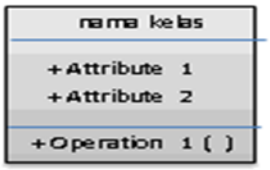
sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. *Use case diagram* dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem. Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal. Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*. Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain. *(Yuni Sugiarti ; 2013 : 41)*

Simbol	Deskripsi
Use Case 	fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit dan aktor; biasanya dinyatakan dengan menggunakan kata kerja di awal frase nama use case
Aktor  nama aktor	orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang; biasanya dinyatakan menggunakan kata benda di awal frase nama aktor
Asosiasi / association	komunikasi antara aktor dan use case yang berpartisipasi pada use case atau use case memiliki interaksi dengan aktor
Extend  <<extend>>	relasi use case tambahan ke sebuah use case dimana use case yang ditambahkan dapat berdiri sendiri walaupun tanpa use case tambahan itu; mirip dengan prinsip inheritance pada pemrograman berorientasi objek; biasanya use case tambahan memiliki nama depan yang sama dengan use case yang ditambahkan, arah panah menunjukan pada use case yang dituju contoh : 
Include <<include>> 	relasi use case tambahan ke sebuah use case dimana use case yang ditambahkan memerlukan use case ini untuk menjalankan fungsinya atau sebagai syarat dijalankan use case ini. Ada dua sudut pandang yang cukup besar mengenai include di use case, include berarti use case yang ditambahkan akan selalu dipanggil saat use case tambahan dijalankan, contoh : 

Gambar II.10. *Use Case* Diagram
(Sumber : Yuni Sugiarti, 2013 : 42)

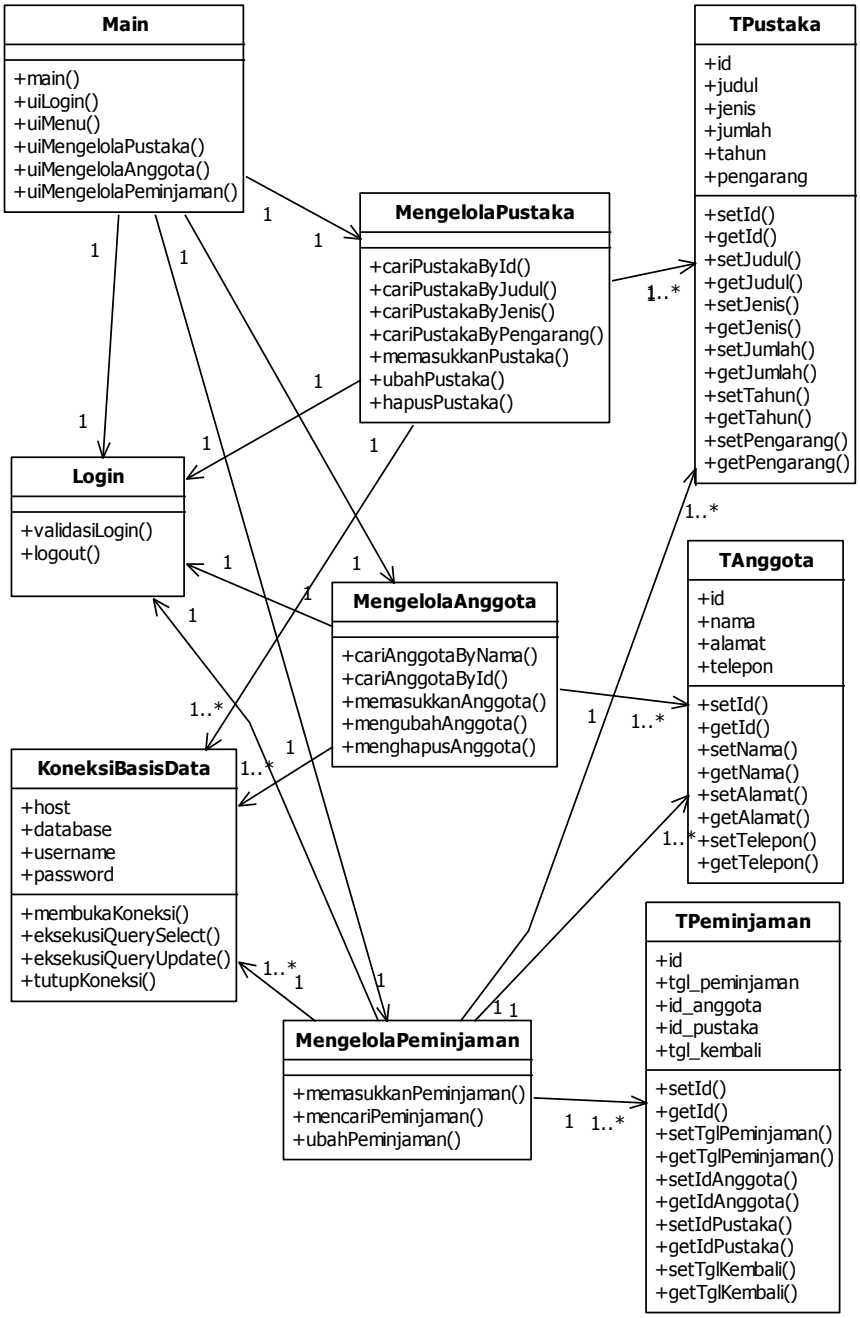
2. *Class Diagram*

Diagram kelas atau *class* diagram menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi. Berikut adalah simbol-simbol pada diagram kelas :

Simbol	Deskripsi
Package 	Package merupakan sebuah bungkusan dari satu atau lebih kelas
Operasi 	Kelas pada struktur sistem
Antarmuka / interface  interface	sama dengan konsep interface dalam pemrograman berorientasi objek
Asosiasi 	relasi antar kelas dengan makna umum, asosiasi biasanya juga disertai dengan multiplicity
Asosiasi berarah/directed asosiasi 	relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan multiplicity
Generalisasi 	relasi antar kelas dengan makna generalisasi-spesialisasi (umum-khusus)
Kebergantungan / defedency 	relasi antar kelas dengan makna kebergantungan antar kelas
Agregasi 	relasi antar kelas dengan makna semua-bagian (whole-part)

Gambar II.11. *Class* Diagram

(Sumber : Yuni Sugiarti, 2013 : 59)



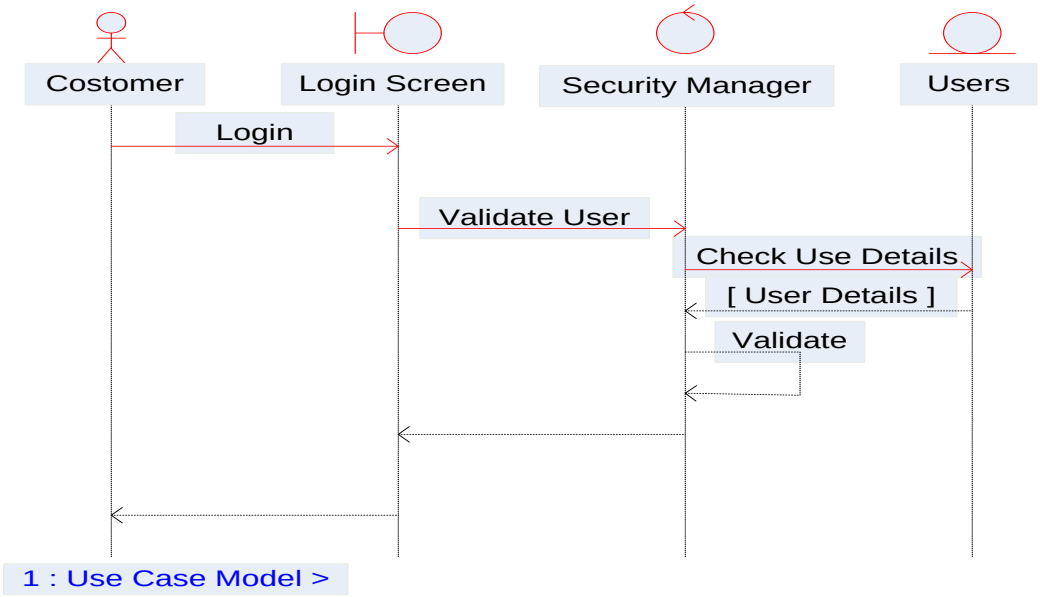
Gambar II.12. Contoh Class Diagram

(Sumber : Yuni Sugiarti, 2013 : 63)

3. *Sequence Diagram*

Diagram *Sequence* menggambarkan kelakuan/prilaku objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambarkan diagram *sequence* maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu.

Banyaknya diagram *sequence* yang harus digambar adalah sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram *sequence* sehingga semakin banyak *use case* yang didefinisikan maka diagram *sequence* yang harus dibuat juga semakin banyak.



Gambar II.13. Contoh *Sequence* Diagram

(Sumber : Yuni Sugiarti, 2013 : 63)

4. *Activity Diagram*

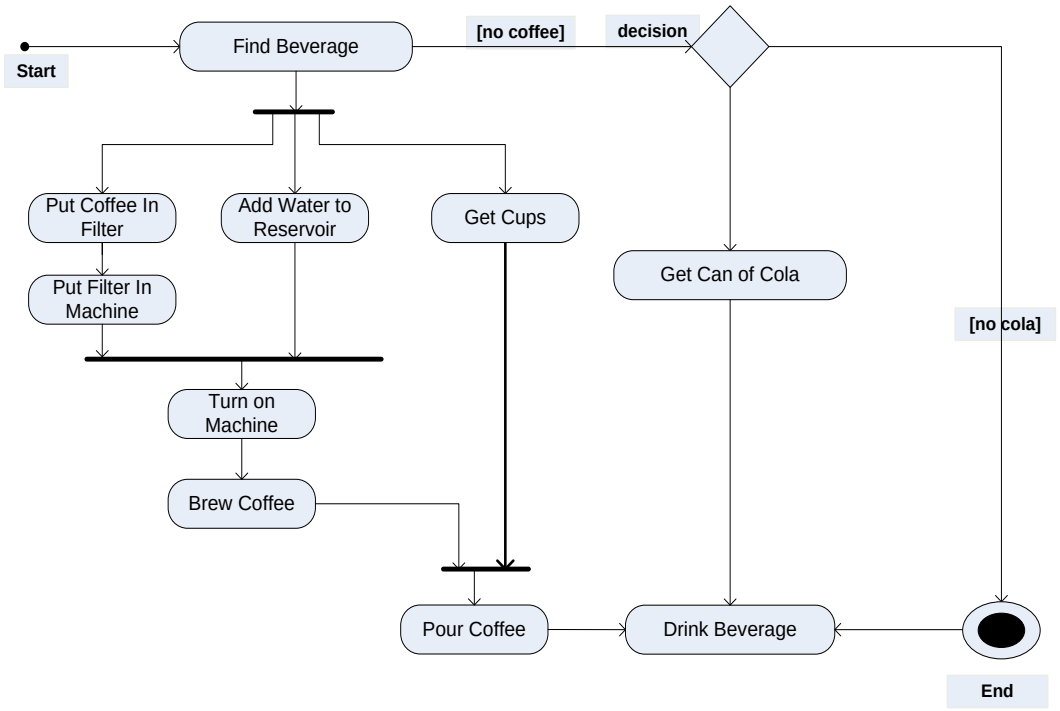
Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Activity diagram merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan behaviour internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

Sama seperti *state*, standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan behaviour pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal.

Activity diagram dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu. (*Yuni Sugiarti ; 2013 : 76)*



Gambar II.14. Activity Diagram

(Sumber : Yuni Sugiarti, 2013 : 76)