BAB II
TINJAUAN PUSTAKA

2.1 *Augmented Reality*

2.1.1 Pengertian *Augmented Reality*

Augmented Reality (AR) merupakan variasi dari lingkungan *virtual* atau yang biasa disebut *Virtual Environtment* (VE). Pada teknologi *VE*, pengguna benar-benar dibawa kedalam suatu lingkungan *virtual* yang akan menyebabkan pengguna tidak dapat lagi melihat dunia nyata yang berada di sekitarnya. Sedangkan *Augmented Reality (AR)* tetap memungkinkan pengguna untuk melihat dunia nyata dengan penambahan objek-objek *virtual* didalam suatu ruang yang sama. Kesimpulannya *AR* merupakan penggabungan antara objek-objek *virtual* dengan dunia nyata di dalam suatu ruang yang sama[2].

Teknologi *Augmented Reality* memiliki 3 karakteristik utama yaitu[2]:

1. Kombinasi antara dunia nyata dengan objek *virtual*.
2. Menyediakan interaksi dengan objek secara *real time*.
3. Berbentuk 3D.

Manfaat utama dari teknologi *AR* adalah memungkinkan pengguna untuk mendapatkan informasi secara lebih menarik dan interaktif. Dengan adanya teknologi AR, pengguna akan memperoleh informasi dengan tampilan teks atau gambar *virtual* beserta dengan objek *virtual* yang digabungkan kedalam dunia nyata. Selain memunculkan informasi *virtual*, pengguna juga dapat berinteraksi dengan objek yang ditampilkan. Berbeda dengan objek 3D konvensional, teknologi *AR* memungkinkan pengguna untuk berinteraksi secara langsung terhadap objek dalam bentuk 3D yang diproyeksikan ke dalam lingkungan nyata. Dengan adanya interaksi ini pengguna dapat memilih sendiri informasi yang telah disediakan didalam suatu aplikasi *Augmented Reality*.

Penerapan teknologi *Augmented Reality* memiliki 4 tugas utama dalam menambahkan objek 3D kedalam lingkungan nyata [6] :

1. Scene Capture

Untuk menambahkan citra digital kedalam lingkungan nyata, maka dibutuhkan sebuah perangkat fisik yang mampu mengenali lingkungan nyata sehingga citra digital dapat ditambahkan kedalam lingkungan nyata tersebut. Setelah objek *virtual* ditangkap, maka objek tersebut dapat dimunculkan ke lingkungan nyata.

2. *Scene Identification Techniques*

Scene identification merupakan suatu proses utama untuk menambahkan citra digital yang berupa objek 3D kedalam lingkungan nyata. *Scene identification* dapat mengklasifikasikan skenario yang terdapat pada suatu aplikasi *Augmented Reality*. Ada 2 tipe dasar dari *scene identification*:

a) *Scene identification* dengan *marker*: *scene identification* yang menggunakan marker sebagai image target untuk memunculkan objek diatas permukaan marker tersebut.

b) *Scene identification* tanpa *marker*: sistem AR yang tidak menggunakan *marker* pada saat perangkat mengidentifikasi *scene*. AR tanpa penggunaan marker biasanya menggunakan fitur *GPS* dari *smartphone* untuk mencari dan berinteraksi dengan sumber daripada *AR*.

3. *Scene Processing*

Setelah menghitung titik koordinat yang telah ditentukan pada permukaan *marker* yang berada pada lingkungan nyata yang sesuai dengan parameter luar dan parameter dalam dari kamera, sistem akan mencari model virtual yang telah didaftarkan pada marker.

4. *Visualization Scene*

Tahap ini merupakan proses akhir dimana sistem akan menghasilkan gambar dari objek 3D yang diproyeksikan pada lingkungan nyata sehingga terdapat suatu adegan yang mencampur antara realitas dan virtualitas dengan penggunaan marker sebagai media untuk menopang keberadaan objek 3D tersebut. Serta menghadirkan informasi digital apabila menggunakan teknik *non marker scene identification*.

2.1.2 *Vuforia SDK*

Vuforia merupakan *Software Development Kit* yang dapat digunakan untuk pengembangan teknologi AR yang dapat diterapkan pada *smartphone* ataupun perangkat *mobile* lainnya yang memungkinkan aplikasi *AR* dijalankan dalam bentuk

video secara *real time*[7]. Sebuah aplikasi berbasis *Vuforia SDK*, menggunakan *display* dari perangkat *mobile* sebagai "lensa ajaib" yang memungkinkan pengguna masuk kedalam suatu dunia dimana dunia nyata dan dunia virtual dapat saling berdampingan. Aplikasi tersebut akan membuat gambar yang ditampilkan pada *preview live camera* untuk mewakili pandangan dunia nyata. Objek virtual berbentuk 3D kemudian ditumpangkan pada preview *live camera* dan objek *virtual* akan tampak bersatu dengan dunia nyata[8].

2.1.2.1 Arsitektur *Vuforia*

Sebuah aplikasi *AR* yang berbasis *SDK Vuforia* terdiri atas beberapa komponen utama, yaitu[7][8]:

1. ***Camera***

Komponen kamera harus dapat memastikan bahwa setiap *frame preview* telah ditangkap dan diteruskan secara efisien kepada *tracker*. *Developer* hanya cukup memberi perintah kapan kamera harus mulai dan berhenti menangkap. *Frame* kamera akan secara otomatis disampaikan pada perangkat tergantung pada format dan ukuran gambar.

2. ***Image Converter***

Format *pixel converter* akan mengkonversi dari format kamera (misalnya *YUV12*) ke dalam format yang sesuai untuk *OpenGL ES* (misalnya, *RGB565*) dan untuk melacak (misalnya *luminance*/pencahayaan) internal.

3. ***Tracker***

Merupakan komponen pelacak yang berisikan algoritma *computer vision* yang mendeteksi dan melacak objek dunia nyata dalam suatu *frame camera video*. Berdasarkan pada gambar kamera, algoritma yang berbeda memiliki tugas untuk mendeteksi target baru atau *marker* dan mengevaluasi *virtual button*. Hasilnya akan disimpan dalam sebuah keadaan objek yang digunakan pada *video background renderer* dan dapat diakses dari kode aplikasi tersebut. *Tracker* dapat melacak beberapa *dataset* pada saat yang bersamaan dan mengaktifkan mereka.

4. ***Video Background Renderer***

Modul *video background renderer* akan me-*render* gambar pada kamera yang tersimpan pada *state* objek. Performa dari *background video rendering* sangat tergantung terhadap sepsifikasi perangkat yang digunakan.

5. *Application Code*

Pengembang aplikasi harus menginisialisasikan seluruh komponen diatas dan melakukan 3 langkah utama didalam pengkodean aplikasi. Untuk setiap *frame* yang akan diproses, *state* objek akan diperbarui dan akan terjadi pemanggilan *render* terhadap aplikasi. Pengembang aplikasi harus:

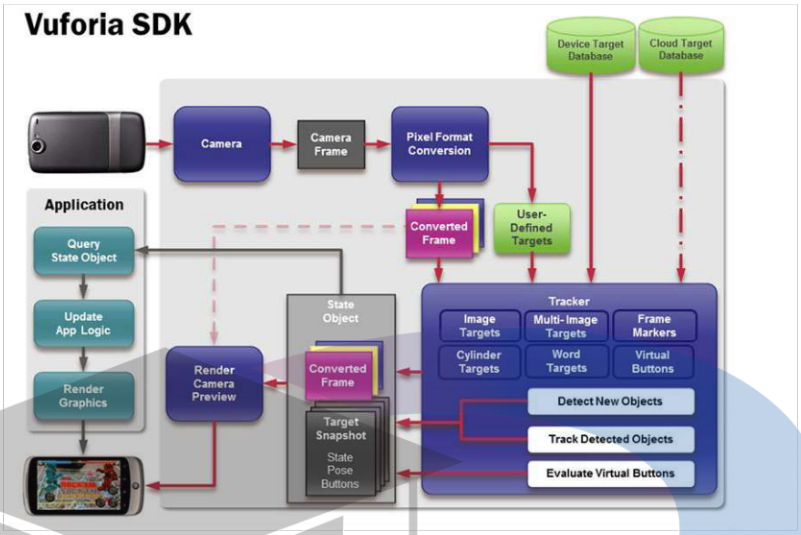
a) *Query state* objek untuk target yang baru dideteksi, *marker* ataupun *state* yang telah diperbarui dari elemen ini.
b) Memperbarui logika aplikasi dengan *input data* yang baru.
c) *Render* lapisan daripada gambar *augmented*.

6. *User Defined Targets*

Sebuah pendekatan yang sangat berbeda adalah fitur *user defined target*. Dibandingkan mempersiapkan target yang disediakan oleh pengembang, fitur ini memungkinkan pembuatan target *"on-the-fly"* dari gambar kamera saat aplikasi dijalankan. Sebuah komponen pembangun dipanggil untuk memicu pembuatan daripada *user-target* yang baru. Target akan dikembalikan dalam bentuk *cache*, tetapi akan mempertahankan sesi *AR* yang akan diberikan.

7. *Target Resources*

Target resources dibuat menggunakan *on-line Target Management System*. Dataset tersebut berisikan konfigurasi dari file *XML* yang memungkinkan pengembang untuk mengkonfigurasi fitur pelacak tertentu dan sebuah *binary file* yang berisikan *trackable database*. Asset ini di-*compile* oleh pengembang aplikasi menjadi sebuah *app installer package* dan digunakan secara *run-time* oleh *Vuforia SDK*.



Gambar2.1 *Data Flow Diagram* pengembangan aplikasi *Vuforia SDK [8]*

2.1.2.2 Image Target

Image target merupakan gambar yang dapat dideteksi dan dilacak oleh *vuforia SDK*. Gambar tersebut tidak harus berupa area hitam-putih yang spesial ataupun kode untuk dikenali. *Vuforia SDK* menggunakan sebuah algoritma untuk melacak fitur yang ada menjadi pengenalan gambar dengan membandingkan fitur ini terhadap objek yang dikenal oleh *database*. Setelah terdeteksi, *Vuforia* akan melacak gambar sejauh jarak pandang kamera[7]. Berikut ini adalah contoh *Image Target* yang telah disediakan oleh *Vuforia*.

Gambar 2.2 *Sample Image Target* yang telah disediakan oleh *Vuforia* [8]

Gambar 2.3 Representasi Objek 3D kedalam lingkungan nyata dengan memanfaatkan *Image Target [8]*

2.1.2.3 User Defined Target

User Defined Target adalah *Image Target* yang dibuat pada saat *runtime* dari *frame* kamera yang dipilih oleh pengguna. *User Defined Target* berbagi sebagian besar

kapabilitas dari sebuah standar *Image Target* dengan pengecualian bahwa *User Defined Target* tidak mendukung *virtual button*[9].

Proses menangkap, membangun, dan melacak daripada *User Defined Target* dikelola oleh logika aplikasi menggunakan *API Vuforia*. Dikarenakan *Image Target* dipilih oleh pengguna, maka penting untuk mengkomunikasikan kepada para pengguna apa jenis gambar yang dapat digunakan dan bagaimana gambar tersebut harus ditangkap untuk mendapatkan *user experience* yang terbaik[9].

Untuk *User Defined Target*, aplikasi memiliki tanggung jawab sebagai berikut[9]:

1) Memulai proses pemindaian target.
2) Memicu proses untuk membangun target.
3) Menambahkan target yang baru diperoleh kedalam *database* untuk pelacakan.

Dalam aplikasi, pengembang harus berkomunikasi kepada pengguna tentang kualitas target yang baik sehingga pengguna akhir dapat berhasil memperoleh target. Target yang ditetapkan pengguna yang ideal mencakup atribut berikut[9]:

Tabel 2.1 Attribut Rekomendasi *User Defined Target* [9]

Attribut	Contoh
Kaya akan detail	Sebuah pemandangan jalan, sekelompok orang, campuran *item*, dan pemandangan olahraga
Memiliki kontras yang baik	.
Tidak ada pola yang berulang	Sebuah lapangan rumput, bagian depan gedung rumah modern dengan jendela yang identik, dan kotak-kotak
Kemudahan ketersediaan	Kartu nama, majalah, dan memo

Gambar 2.4 Contoh instruksi yang digunakan dalam *User Defined Target* [9]

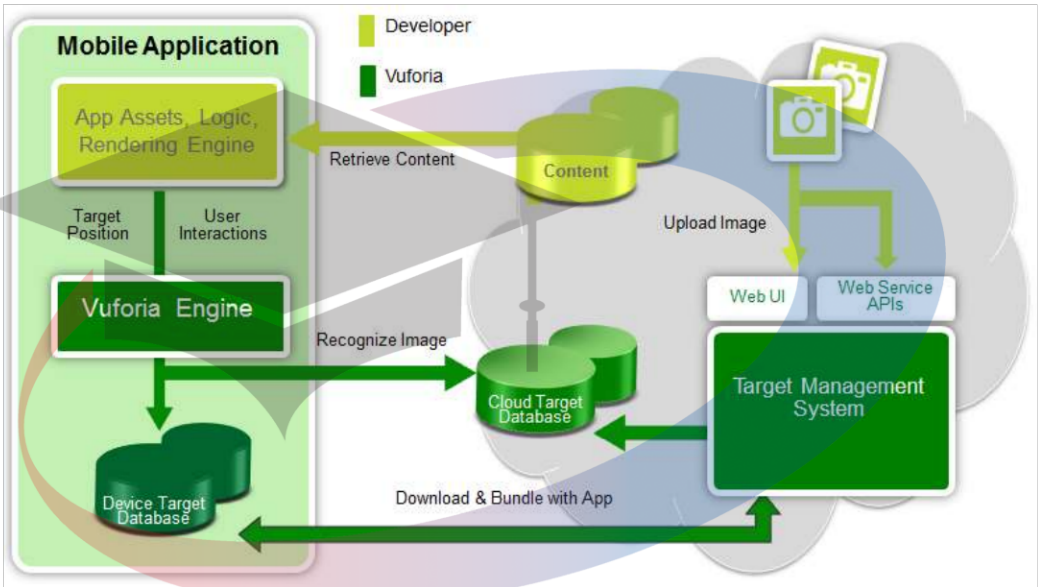
2.1.2.4 Kelebihan Vuforia SDK

Pengembangan aplikasi menggunakan *Vuforia* SDK memiliki beberapa kelebihan, antara lain[8]:

a. Deteksi yang lebih cepat dari target lokal.
b. *Cloud Recognition* hingga 1 juta target secara simultan.
c. *User Defined Target* untuk pembuatan target secara *run-time.*
d. *Cylinder target* – deteksi dan pelacakan gambar pada permukaan silinder.
e. *Text Recognition* – Deteksi dan pelacakan dari teks yang dicetak, dan meliputi urutan *alpha-numeric*.
f. *Robust Tracking* – augmentasi menempel pada target dan tidak mudah menghilang walaupun *device* bergerak.
g. Dapat melacak secara simultan hingga 5 target.
h. Hasil yang lebih baik dalam lingkungan nyata walaupun dengan pencahayaan yang cukup rendah.
i. Optimasi yang menjamin grafis yang lebih baik dan lebih realistis yang ditampilkan pada target.

j. Kemampuan pelacakan yang diperluas, yang memungkinkan aplikasi untuk tetap melacak target dan tetap mempertahankan konsistensi daripada augmentasi walaupun target tidak lagi berada pada wilayah pelacakan kamera.

Berikut ini adalah diagram yang memberikan gambaran tentang proses pengembangan aplikasi dengan platform *vuforia*.



Gambar 2.5 Diagram pengembangan aplikasi dengan platform *Vuforia* [8]

2.2 Sistem Operasi *Android*

Android merupakan sebuah sistem operasi pada perangkat *mobile* seperti *Smartphone* dan *PC Tablet* yang merupakan versi modifikasi dari *Linux*. Pada awalnya, *Android* dikembangkan oleh perusahaan *startup* yang bernama *Android, Inc* pada tahun 2005. Sebagai salah satu strategi untuk memasuki industri *mobile*, maka *Google* membeli *Android* dan mengambil alih pekerjaan dalam pengembangan *Android*. *Google* menginginkan *Android* menjadi sitem operasi yang terbuka dan gratis. Maka, sebagian besar *code Android* dirilis dibawah *Apache License open-source* yang berarti bahwa siapa saja yang ingin menggunakan *Android*, dapat mengunduh *source code Android* secara kesuluruhan. Selain itu, *vendor* (biasanya produsen *Hardware*) dapat menambahkan ekstensi mereka sendiri untuk *Android* dan menyesuaikan *Android* untuk membedakan produk mereka dari produsen lainnya[10].

Keuntungan utama dari mengadopsi sistem operasi *Android* adalah *Android* menawarkan pendekatan yang terpadu untuk pengembangan aplikasi. *Developer*

hanya perlu mengembangkan untuk *Android* dan aplikasi mereka dapat dijalankan pada beberapa perangkat yang berbeda selama perangkat tersebut menggunakan *Android*[10].

1. Versi *Android*

Android telah mengalami beberapa perubahan sejak pertama kalinya diluncurkan. Tabel berikut ini akan menunjukkan beberapa versi Android yang telah diluncurkan.

Tabel 2.2 Perkembangan Versi *Android* [11]

Versi Android	Code Name
1.0	Tanpa *Code Name*
1.5	Cupcake
1.6	Donut
2.0 - 2.1	Eclair
2.2	Froyo
2.3 - 2.3.7	Gingerbread
3.0 – 3.2	Honeycomb
4.0.1 – 4.0.4	Ice Cream Sandwich
4.1 – 4.3	Jelly Bean
4.4 – 4.4.4	KitKat
5.0 – 5.1	Lollipop
6.0	Marshmallow

2. Fitur-fitur *Android*

Android mendukung fitur-fitur sebagai berikut[10]:

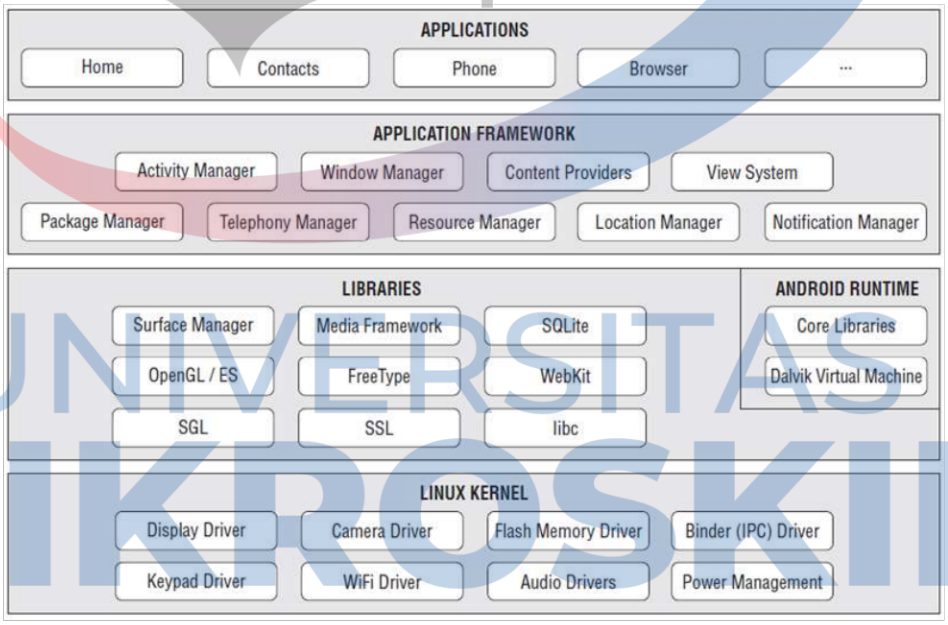
1) **Penyimpanan -** menggunakan *SQL*, sebuah database relasional yang ringan untuk penyimpanan data.
2) **Konektivitas** - Mendukung *GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, WiFi, LTE,* dan *WiMAX*.
3) **Pengolah Pesan** – Mendukung *SMS* dan *MMS*.
4) ***Web Browser***- Berdasarkan *WebKit OpenSource* bersama dengan *Chrome*.
5) **Dukungan Media** – Meliputi media berikut ini: *H.263, H.264*(dalam kategori *3GP* dan *MP4*), *MPEG-4 SP, AMR, AMR-WB* (dalam kategori *3GP*), *AAC, HE-AAC*

(dalam kategori *3GP* dan *MP4*), *MP3, MIDI, Oog Vorbis, WAV, JPEG, PNG, GIF,* dan *BMP*.

6) **Dukungan *Hardware*** – Sensor *Accelerometer*, kamera, kompas digital, sensor jarak, dan *GPS*.
7) ***Multi Touch*** – Mendukung layar *Multi Touch*.
8) ***Multi-Tasking*** – Mendukung aplikasi *Multi-Tasking*.
9) **Mendukung *Flash*** – *Android* 2.3 mendukung *Flash* 10.1.
10) ***Tethering*** – Mendukung pembagian jaringan internet sebagai *hotspot* nirkabel mapupun dengan menggunakan kabel.

3. Arsitektur *Android*

Gambar berikut ini akan menjelaskan beberapa lapisan yang yang membentuk sistem operasi *Android*[10].



Gambar 2.6 Arsitektur Sistem Operasi *Android* [10]

Sistem operasi *Android* dibagi menjadi 5 lapisan utama yaitu[10]:

1) ***Linux Kernel*** – Lapisan ini merupakan kernel dasar dari *Android*. Lapisan ini berisi semua *driver* tingkat rendah untuk berbagai komponen *hardware* dari sebuah perangkat *Android*.

2) ***Libraries*** – Lapisan ini mengandung seluruh *code* yang menyediakan fitur utama dan sebuah sistem operasi *Android*. Sebagai contoh, *library SQLite* menyediakan dukugan database untuk penyimpanan data.
3) ***Android Runtime*** - Pada lapisan yang sama seperti *Library*, *Android Runtime* menyediakan satu set *library* inti yang memungkinkan pengembang aplikasi untuk mengembangkan aplikasi *Android* menggunakan bahasa pemrograman *Java*.
4) ***Application Framework*** – Memperlihatkan berbagai kemampuan daripada sistem operasi *Android* kepada para pengembang aplikasi sehingga dapat dimanfaatkan untuk pengembangan aplikasi.
5) ***Applications*** – Pada lapisan teratas ini anda akan menemukan aplikasi yang berada perangkat *Android* (seperti Telepon, Kontak, *Browser* dll.) beserta dengan aplikasi yang telah anda unduh dan install sendiri dari *Android Market*. Seluruh aplikasi yang anda miliki pada perangkat *Android* berada pada lapisan ini.

4. *Android SDK (Software Development Kit)*

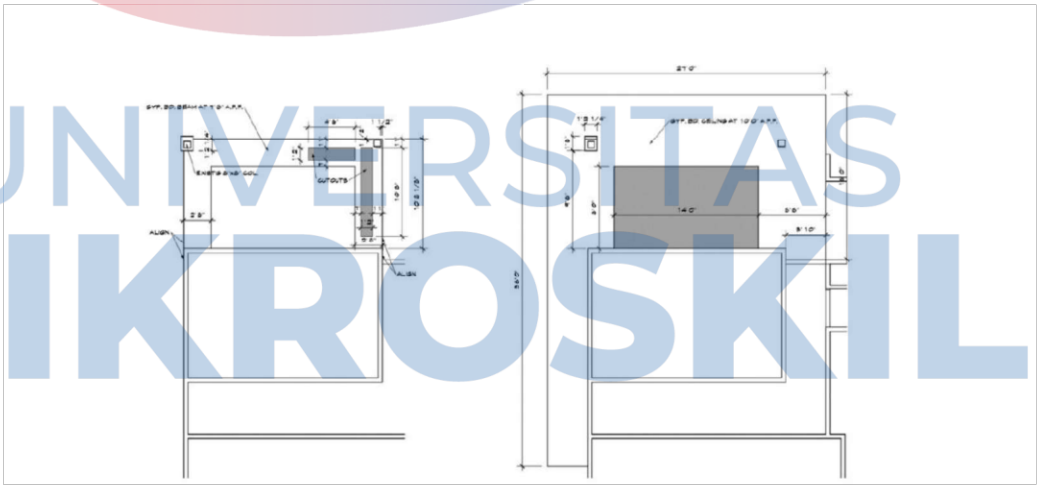
Merupakan sebuah alat pengembangan aplikasi yang memungkinkan pengguna untuk membuat aplikasi pada *platform Android. Android SDK* meliputi *debugger, library, emulator,* dokumentasi, *sample code,* dan *Tutorial*[10].

2.3 *Building Information Modelling*

Building Information Modeling (BIM) merupakan representasi digital dari karakteristik fisik dan fungsional dari sebuah fasilitas. Sebuah *BIM* adalah model cerdas yang dapat mengintegerasikan desain, visualisasi, simulasi, dan kolaborasi kedalam satu proses. Model merupakan representasi fisik, tetapi model tersebut juga dapat menjadi media penyampaian informasi. Model tidak hanya menunjukkan kepada pengguna seperti apa bentuk bangunan tersebut, tapi model tersebut juga dapat memberikan pengguna dan desainer pengertian yang lebih tentang kegunaan dari bangunan tersebut. Sebuah *BIM* pada dasarnya merupakan *prototype* bangunan digital yang membantu semua orang di dalam tim desain membuat keputusan yang lebih baik[12].

BIM merupakan sebuah konsep, bukan sebuah *software*. Akan tetapi, ada beberapa *software* yang mengunakan konsep *BIM* untuk melakukan proses desain. Salah satu software yang dapat melakukan proses ini adalah *SketchUp*. Adapun beberapa keunggulan dari penggunaan *SketchUp* adalah sebagai berikut[12]:

1) *SketchUp* merupakan pemodelan permukaan, yang artinya seluruh objek yang dibuat menggunakan *SketchUp* tersusun atas garis dan pernukaan. Proses penggambaran di *SketchUp* sangat mirip dengan proses penggambaran garis yang digunakan pada *2D CAD*. *SketchUp* dapat didefinisikan sebagai program *2D CAD* yang beroperasi dalam lingkungan 3D.
2) Ketika membangun model 3D menggunakan *SketchUp*, maka pengguna akan secara simultan menciptakan dokumen konstruksi. Seluruh perencanaan dalam bentuk 2D, bagian dari konstruksi, dan peningkatan secara dinamis terkait dengan model 3D. *SketchUp* memungkinkan pengguna untuk berpikir dan mendesain dalam bentuk 3D. Hal ini sangat kontras dengan penggunaan *software BIM* lainnya dimana pengguna akan menyusun dokumen konstruksi yang pada akhirnya akan menciptakan sebuah model 3D.



Gambar 2.7 Pemodelan 2D dari sebuah konstruksi [12]

Gambar 2.8 Pemodelan kembali dari model 2D menjadi model 3D dengan penggunaan SketchUp [12]

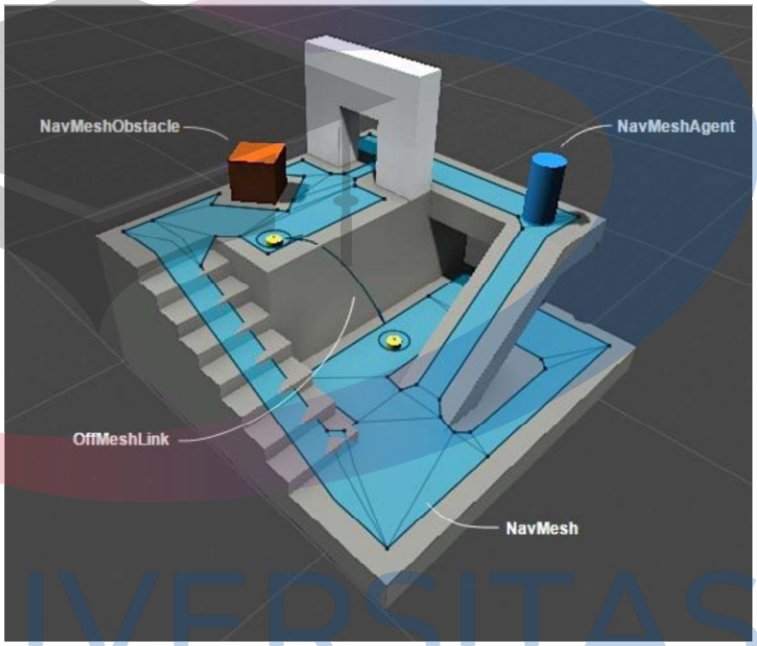
3) Penggunaan *SketchUp* yang sangat sederhana memungkinkan pengguna untuk membuat dan mengatur model secara cepat dan tanpa interupsi. Misalnya, untuk menambahkan dinding di beberapa paket *software BIM* lainnya, pengguna harus menetapkan beberapa properti, seperti tinggi, ketebalan, bahan, warna, dan isolasi. Untuk menambahkan dinding di SketchUp, Anda hanya perlu menggambar persegi panjang kemudian menarik objek tersebut.
4) *SketchUp* menawarkan *real-time rendering* yang akan memberikan informasi lebih baik sehingga pengguna dapat membuat keputusan desain yang lebih baik dibandingkan dengan *software* lainnya.

2.4 *Unity3D*

Unity 3D merupakan sebuah *game engine* yang dapat digunakan pada berbagai *platform* yang dikembangkan oleh *Unity Technologies*. *Unity3D* pada umumnya digunakan oleh para pengembang untuk menciptakan *games*, tetapi selain pengembangan *games,* Unity juga dapat digunakan untuk pembuatan konten 3D dan 2D yang interaktif seperti simulasi pelatihan dan juga visualisasi di bidang medis dan arsitektur serta pembuatan aplikasi *Augmented Reality* yang dikombinasikan dengan penggunaan *Library Vuforia* untuk menghasilkan aplikasi *Augmented Reality* di berbagai platform seperti perangkat *mobile*, *desktop*, dan *web*.[13]

1. Sistem Navigasi pada *Unity*

Sistem navigasi *(NavMesh)* yang tersedia di *Unity* memungkinkan pengembang untuk membuat karakter yang dapat menjelajahi dunia virtual yang telah dibuat oleh si pengembang aplikasi. Sistem navigasi tersebut memberikan karakter kemampuan untuk memahami bahwa karakter tersebut perlu melalui suatu rangkaian jalur untuk mencapai tujuan yang telah ditentukan, dan untuk mencapai tujuan tersebut karakter harus menghindari berbagai rintangan agar tujuan yang telah ditentukan dapat tercapai. Sistem Unity NavMesh terdiri dari beberapa bagian:[14]



Gambar 2.9 Komponen sistem navigasi pada *Unity*[14]

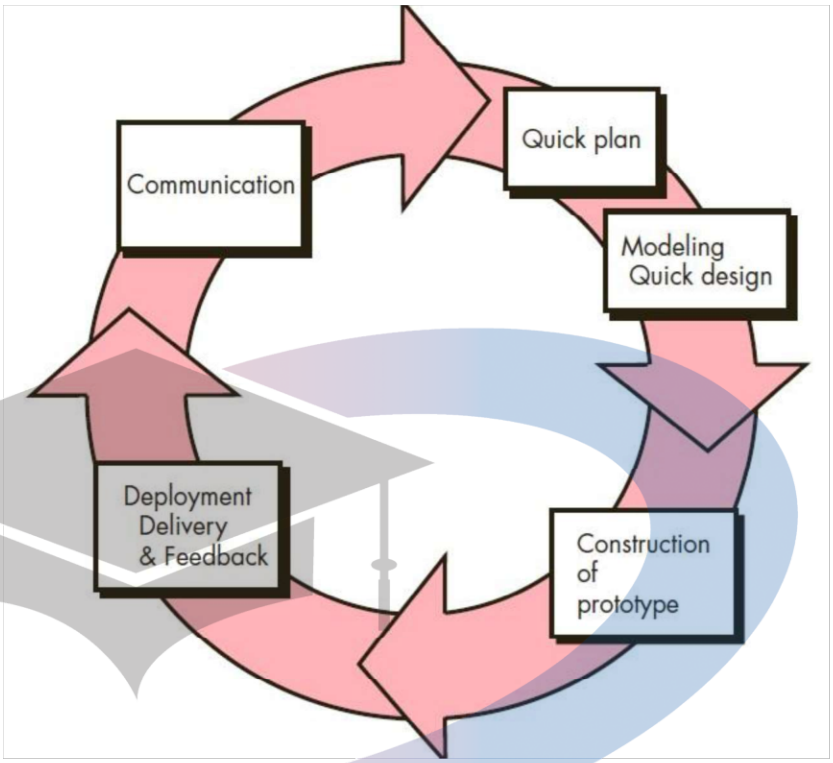
1. *NavMesh* (singkatan dari *Navigation Mesh*) adalah struktur data yang menggambarkan permukaan yang dapat dijalani dari dunia virtual dan memungkinkan *NavMesh Agent* untuk menemukan suatu jalan dari satu lokasi ke lokasi yang lain didalam dunia virtual.
2. Komponen *NavMesh Agent* membantu pengembang untuk membuat karakter yang dapat menghindari objek penghalang selagi bergerak menuju tujuan dari karakter tersebut. *NavMesh Agent* tahu bagaimana cara untuk menghindari hambatan satu sama lain serta hambatan gerak dari karakter tersebut.
3. Komponen *Off-Mesh link* memungkinkan pengembang untuk menggabungkan jalan pintas navigasi yang tidak dapat diwakili menggunakan permukaan yang

dapat dijalani. Misalnya, melompati parit atau pagar, atau membuka pintu sebelum berjalan melalui objek tersebut.

2.5 *Prototyping*

Pada beberapa kasus pengembangan *software,* pelanggan seringkali mendefinisikan serangkaian tujuan umum untuk pengembangan tersebut, tetapi tidak mendefinisikan persyaratan rinci untuk fungsi serta fitur dari *software* tersebut. Di sisi lain pengembang sistem menjadi kesulitan dalam memilih efisiensi algoritma, adaptasi dari sistem operasi, ataupun interaksi antara manusia terhadap mesin dari *software* yang akan dikembangkan. Dalam kasus seperti ini paradigma prototyping menawarkan pendekatan yang terbaik. Paradigma *prototyping* akan membantu para pengembang *software* dan para *stakeholder* untuk mendapatkan pemahaman lebih mengenai apa yang harus dibangun ketika seluruh persyaratan pengembangan *software* masih belum jelas sepenuhnya.[15]

Paradigma *prototyping* dimulai dengan komunikasi. Pengembang akan bertemu dengan para *stakeholder* untuk menentukan tujuan secara keselurahan dari *software* yang akan dikembangkan, mengidentifikasi keseluruhan persyaratan, serta menjabarkan keseluruhan garis besar dimana pengembangan akan dilakukan secara bertahap merupakan hal yang wajib. Perulangan perencanaan dan pemodelan *prototyping* dilakukan secara cepat. Desain cepat tersebut berfokus pada representasi dari aspek-aspek perangkat lunak yang akan dilihat oleh para pengguna (tata letak antarmuka yang akan digunakan oleh para pengguna ataupun tampilan format *output*) Desain cepat tersebut mengarah kepada pembangunan dari sebuah *prototype. Protoype* dikembangkan dan dievaluasi oleh para *stakeholder* yang akan memberikan umpan balik berupa perbaikan persyaratan untuk kedepannya. Kegiatan ini akan terus berulang hingga segala kebutuhan dari para *stakeholder* telah terpenuhi seiring para pengembang sistem akan lebih memahami apa saja yang diperlukan dalam pengembangan tersebut.[15]



Gambar 2.10 Pengembangan perangkat lunak dengan metodologi *Prototyping* [15]

2.6 *Unified Modelling Languange (UML)*

Unified Modeling Language (UML) adalah tujuan umum bahasa pemodelan visual yang digunakan untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan artefak dari sistem perangkat lunak. UML akan menangkap keputusan serta pemahaman tentang sistem yang harus dibangun. UML digunakan untuk memahami, merancang, menelisik, mengkonfigurasi, memelihara dan mengontrol informasi dari perangkat lunak tersebut.[16]

1. Konsep Dasar *UML*

View merupakan bagian dari pemodelan konstruksi *UML* yang merupakan salah satu aspek dari suatu sistem. Pada tingkat paling atas, *view* dapat dibagi menjadi beberapa area: *structural classification*, *dynamic behavior*, *physical layout*, dan *model management*. Tabel berikut ini akan menjabarkan lebih jauh tentang *view UML* beserta

dengan diagram yang digunakan, dan juga konsep utama yang terkait dengan setiap *view*.[16]

Tabel 2.3 Konsep Dasar *UML* [16]

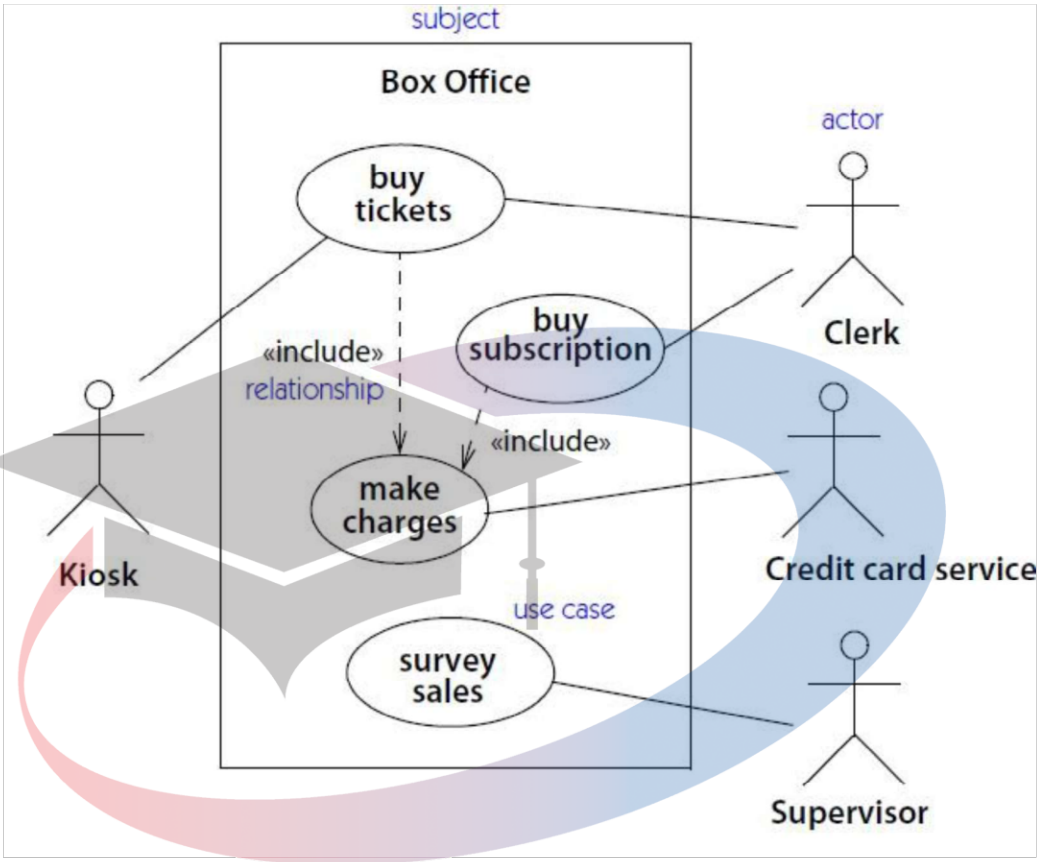
Major Area	View	Diagram	Main Concepts
Structural	Static view	Class diagram	Association, class, dependency, generalization, interface, realization
	Design view	Internal structure	Connector, interface, part, port, provided interface, role, required interface
		Collaboration diagram	Connector, collaboration, collaboration use, role
		Component diagram	Component, dependency, port, provided interface, realization, required interface, subsystem
	Use case view	Use case diagram	Actor, association, extend, include, use case, use case generalization
Dynamic	State machine view	State machine diagram	Completion transition, do activity, effect, event, region, state, transition, triger
	Activity view	Activity diagram	Action, activity, control flow, control node, data flow. Exception, expansion region, fork, join, object node, pin
	Interaction view	Sequence diagram	Occurence specification, execution specification, interaction,

			interaction fragment, interaction operand, lifeline, message, signal
		Communication diagram	Collaboration, guard condition, message, role, sequence number
Physical	Deployment view	Deployment diagram	Artifact, dependency, manifestation, node
Model management	Model management view	Package diagram	Import, model, package
	Profile	Package diagram	Constraint, profile, stereotype, tagged value

Berdasarkan konsepsi dasar yang telah dijabarkan pada tabel diatas, berikut ini penjelasan lebih terperinci mengenai pendefinisian diagram-diagram yang digunakan dalam *UML*

1. *Use Case Diagram*

Use case memodelkan fungsionalitas dari sebuah subjek seperti yang dirasakan oleh agen luar, yang disebut sebagai *actor*, yang berinteraksi dengan beberapa subyek dari sudut pandang tertentu. Sebuah *use case* adalah satuan fungsi yang dinyatakan sebagai transaksi antara *actor* dan subyek. Tujuan utama dari sebuah *use case* adalah untuk menyusun daftar yang menggambarkan *actor* dan *use case* dan menunjukkan *actor* yang berpartisipasi di dalam tiap-tiap *use case*. Perilaku *use case* dinyatakan menggunakan tampilan dinamis, terutama pada pandangan interaksi. Berikut ini merupakan sebuah contoh dari sebuah diagram *use case*. [16]



Gambar 2.11 Contoh diagram *use case* [16]

Use case akan menangkap perilaku sistem, subsistem, *class*, atau komponen yang muncul untuk pengguna sistem. *Use case* membagikan fungsionalitas sistem kedalam transaksi untuk *actor* ideal dari pengguna sistem. Potongan-potongan fungsi interaktif disebut *use case*. *Use case* menggambarkan interaksi dengan *actor* sebagai urutan pesan antara sistem dengan satu atau lebih *actor*. *Actor* tersebut meliputi manusia serta sistem komputer.

Use Case memiliki 2 fitur utama yaitu:

1) Actor

Actor merupakan idealisasi dari peran yang dimainkan oleh orang, proses, atau hal eksternal yang berinteraksi dengan sistem, subsistem, ataupun *class*. *Actor* menggambarkan interaksi yang dimiliki oleh para pengguna luar dengan sistem yang ada. Pada waktu berjalan, salah satu pengguna fisik dapat terikat ke beberapa *actor* yang terdapat dalam sistem. Setiap aktor berpartisipasi dalam satu atau lebih *use case*.

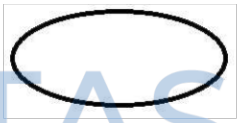
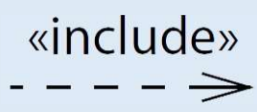
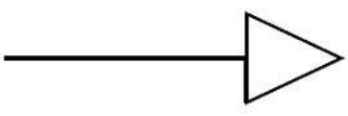
Actor melakukan interaksi dengan *use case* dengan cara bertukar pesan. Seorang aktor mungkin manusia, sistem komputer, ataupun beberapa proses yang dapat dieksekusi. [16]

2) Use Case

Sebuah *use case* adalah satuan yang berhubungan dari fungsi eksternal terlihat yang disediakan oleh *classifier* dan diungkapkan oleh urutan pesan yang dipertukarkan oleh subjek dan satu atau lebih *actor* dari unit sistem. Tujuan dari *use case* adalah untuk menentukan bagian dari perilaku yang yang saling berhubungan tanpa mengungkapkan struktur internal subjek. [16]

Selain kedua fitur utama tersebut, *use case* juga memiliki simbol-simbol lainnya seperti yang dijabarkan pada tabel berikut.

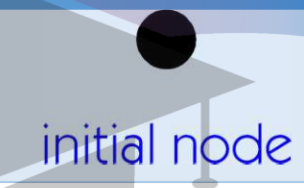
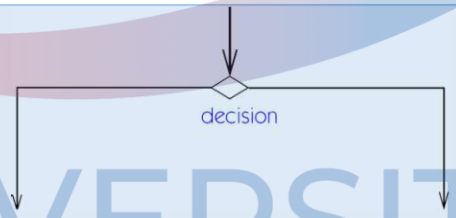
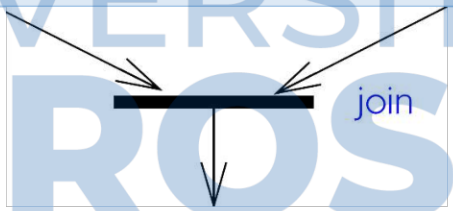
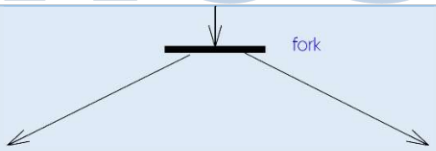
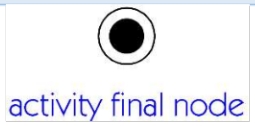
Tabel 2.4 Simbol-simbol yang digunakan pada diagram *Use Case* [16]

Hubungan	Fungsi	Simbol/Notasi
Actor	Seseorang, proses, ataupun sesuatu yang memiliki interaksi dengan sistem, subsistem, ataupun *class*	
Use Case	Menggambarkan kegiatan-kegiatan yang dapat dilakukan oleh *actor*	
Association	Digunakan untuk menggambarkan hubungan antara *actor* dan *use case*	
Extend	Penyisipan perilaku tambahan kepada sebuah use case	
Include	Penyisipan perilaku tambahan ke dalam *use case* dasar yang juga secara eksplisit menjelaskan penyisipan tersebut	
Use Case Generalization	Keterhubungan antara *use case* umum dan *use case* yang lebih spesifik	

2. *Activity Diagram*

Activity diagram merupakan sebuah diagram yang menggambarkan aliran kerja dari sebuah sistem atau proses bisnis. *Activity diagram* memvisualisasikan aliran kerja yang berisikan aktivitas, tindakan, pilihan, serta pengulangan. Beberapa notasi yang digunakan pada class diagram adalah sebagai berikut:[16]

Tabel 2.5 Jenis-jenis notasi yang digunakan pada *Activity Diagram [16]*

Nama	Simbol	Keterangan
Initial		Merupakan titik awal dari aktivitas sebuah sistem.
Activity		Merepresentasikan pelaksanaan dari pernyataan dalam prosedur atau pelaksanaan langkah dalam alur kerja.
Decision		Merupakan percabangan apabila adanya aktivitas yang memiliki pilihan lebih dari satu.
Join		Menunjukkan aktivitas yang digabungkan menjadi satu.
Fork		Menunjukkan suatu kegiatan yang dilakukan secara pararel.
Flow Final		Merupakan titik akhir yang dilakukan oleh sistem

3. *Class Diagram*:

Elemen utama dari *class diagram* adalah *class*, yang menyimpan dan mengelola informasi di dalam sistem. Selama proses analisis, *class* merujuk kepada orang, tempat, peristiwa, dan hal-hal lain yang nantinya informasi tersebut akan ditangkap oleh sistem dari waktu ke waktu. Kemudian, selama proses perancangan dan implementasi, *class* dapat merujuk pada artefak implementasi khusus seperti *windows*, *form*, dan benda-benda lain yang digunakan untuk membangun sistem. Setiap kelas digambarkan dengan menggunakan persegi panjang yang dibagi menjadi tiga bagian dengan nama *class* berada di atas, atribut di tengah, dan method berada di bagian paling bawah. [17]

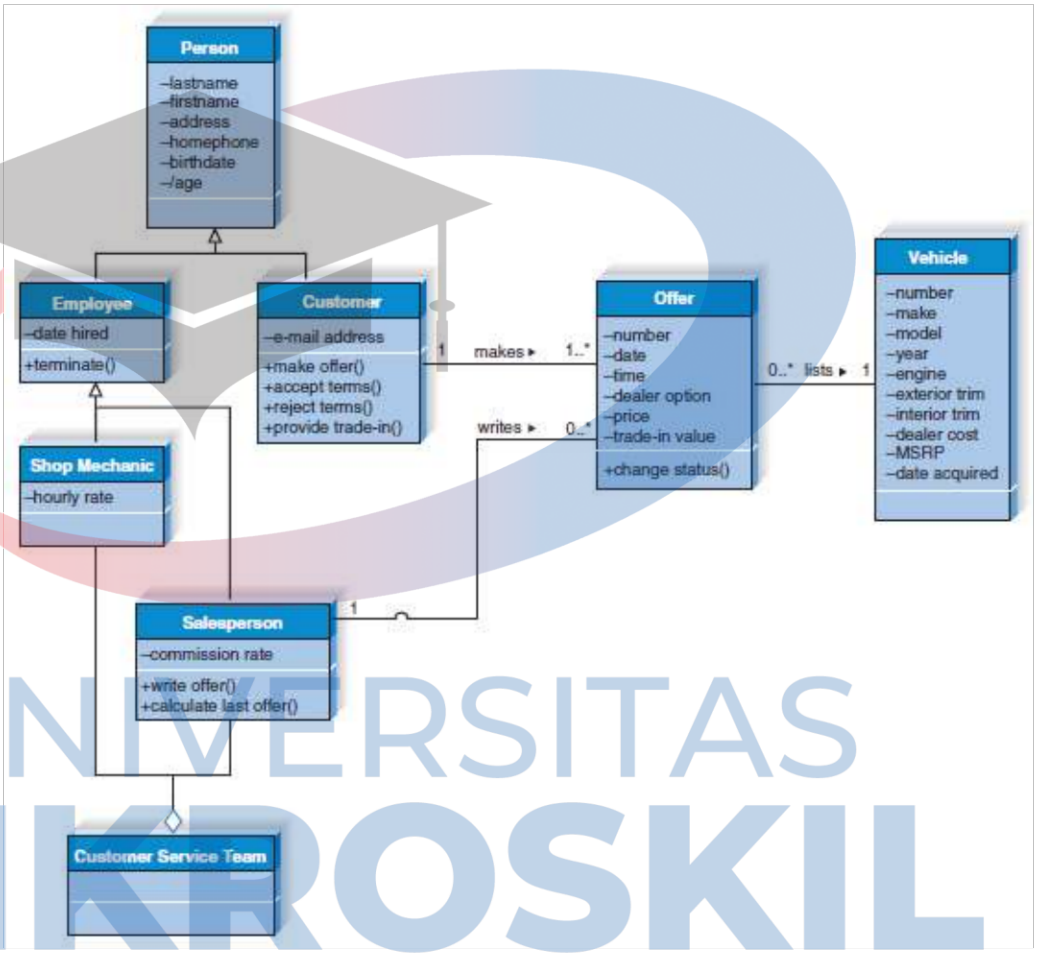
Atribut merupakan properti *class* yang kita inginkan untuk menangkap informasi. Perhatikan bahwa *class Person* pada gambar 2.12 berisi atribut nama belakang, nama depan, alamat, telepon, tanggal lahir, dan usia. Hal ini juga memungkinkan untuk menunjukkan visibilitas atribut pada diagram. Visibilitas berkaitan dengan tingkat menyembunyikan informasi harus ditegakkan untuk atribut. Visibilitas atribut baik dapat berupa *public* (+), *protected* (#), atau *private* (-). Atribut *public* merupakan salah satu atribut yang tidak tersembunyi dari objek lain. Dengan demikian, objek lain dapat memodifikasi nilainya. Sebuah atribut *protected* merupakan salah satu atribut yang tersembunyi dari semua kelas lain kecuali *subclass* terdekatnya. Sebuah atribut private adalah salah satu atribut yang tersembunyi dari semua kelas-kelas lain. Visibilitas standar untuk atribut biasanya *private*. [17]

Operation merupakan tindakan atau fungsi yang dapat dilakukan oleh *class*. Fungsi-fungsi yang tersedia untuk semua kelas (misalnya, membuat contoh baru, mengembalikan nilai untuk atribut tertentu, menetapkan nilai untuk atribut tertentu, atau menghapu) tidak secara eksplisit ditampilkan dalam *class* tersebut. Sebaliknya, hanya operasi yang unik untuk kelas yang disertakan, seperti "menghentikan" dan "membuat tawaran" operasi pada gambar 2.12. Perhatikan bahwa kedua operasi diikuti dengan tanda kurung. Operasi harus ditampilkan dengan tanda kurung yang baik kosong, atau diisi dengan beberapa nilai yang merupakan parameter bahwa operasi itu perlu untuk bertindak. Seperti dengan atribut, visibilitas operasi dapat ditunjuk sebagai *public*, *protected*, atau *private*. Visibilitas standar untuk operasi biasanya *public*. [17]

Class dapat berisi 3 jenis operasi: *constructor*, *query*, dan *update*. Sebuah operasi *constructor* menciptakan contoh baru dari sebuah *class*. Sebuah operasi *query*

membuat informasi tentang kondisi suatu objek yang tersedia untuk objek lainnya, tapi hal itu tidak akan mengubah objek tersebut. Sebuah operasi *update* akan mengubah nilai dari beberapa atau seluruh atribut objek, yang dapat mengakibatkan perubahan kondisi dari objek tersebut.

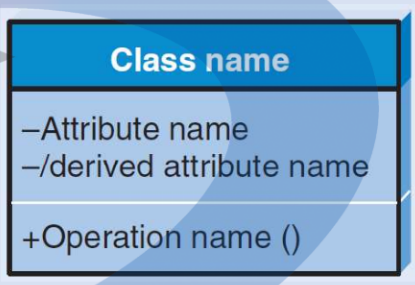
Berikut ini merupakan contoh gambaran dari sebuah *class diagram* [17]:



Gambar 2.12 Contoh dari *Class Diagram*

Tabel berikut akan menjelaskan secara lebih rinci mengenai berbagai simbol yang digunakan pada *class diagram*. [17]

Tabel 2.6 Sintaks *Class Diagram* [17]

Istilah dan Definisi	Simbol
Class ▪ Mewakili orang, tempat, peristiwa, dan hal-hal lain yang nantinya informasi tersebut akan ditangkap oleh sistem. ▪ Memiliki nama diketik dalam huruf tebal dan berada pada kompartemen paling atas. ▪ Memiliki daftar attribut pada kompartemen tengah ▪ Memiliki daftar operasi di kompartemen bawah. ▪ Tidak secara eksplisit menunjukkan operasi yang tersedia untuk semua kelas.	
Attribute ▪ Menunjukkan properti yang menggambarkan keadaan suatu objek. ▪ Dapat diturunkan dari atribut lainnya, ditunjukkan dengan menempatkan garis miring sebelum nama atribut.	Attribute Name /derived attribute name
Method ▪ Menunjukkan tindakan atau fungsi yang dapat dilakukan oleh *class*.	Operation Name ()

▪ Dapat diklasifikasikan sebagai konstruktor, query, atau pembaharuan operasi. ▪ Termasuk tanda kurung yang mungkin berisi parameter khusus atau informasi yang dibutuhkan untuk melakukan operasi.	
Association ▪ Menunjukkan hubungan antara beberapa *class*, atau *class* dengan dirinya sendiri. ▪ Diberi label oleh frase verbal atau nama peran, yang lebih baik untuk mencerminkan hubungan. ▪ Bisa ada di antara satu atau lebih *class*. ▪ Mengandung beberapa simbol, yang mewakili minimum dan maksimum berapa kali contoh *class* dapat dikaitkan dengan contoh *class* terkait.	1..* Verb Phrase 0..1