BAB 2
TINJAUAN PUSTAKA

2.1. Landasan Teori

Landasan teori menjelaskan beberapa teori-teori dan penjelasan yang berkaitan dengan aplikasi atau media yang akan dibangun. Landasan teori yang digunakan dalam penyusunan aplikasi freelancer untuk bidang otomotif pada *platform android*, meliputi Pengertian Aplikasi, Pengertian *Mobile, Android, Geotagging, GPS, Google* Maps, Pengertian Java, UML, dan lain-lain.

2.2. Aplikasi

Menurut (Kadir, 2003, h.204) aplikasi adalah suatu program yang dibuat oleh pemakai yang ditujukan untuk melakukan suatu tugas khusus. Berdasarkan definisi di atas maka dapat disimpulkan bahwa aplikasi adalah program yang dibuat untuk melakukan tugas khusus dalam perusahaan. [7].

2.3. *Android*

Menurut Teguh Arifianto (2011 : 1), *android* merupakan perangkat bergerak pada sistem operasi untuk telepon seluler yang berbasis linux. Menurut Hermawan (2011 : 1), *Android* merupakan OS (Operating System) *Mobile* yang tumbuh ditengah OS lainnya yang berkembang dewasa ini. OS lainnya seperti Windows *Mobile*, *i-Phone OS, Symbian*, dan masih banyak lagi. Akan tetapi, OS yang ada ini berjalan dengan memprioritaskan aplikasi inti yang dibangun sendiri tanpa melihat potensi yang cukup besar dari aplikasi pihak ketiga. Oleh karena itu, adanya keterbatasan dari aplikasi pihak ketiga untuk mendapatkan data asli ponsel, berkomunikasi antar proses serta keterbatasan distribusi aplikasi pihak ketiga untuk *platform* mereka [8].

2.4. Sistem

Sistem adalah sekumpulan unsur / elemen yang saling berkaitan dan saling mempengaruhi dalam melakukan kegiatan bersama untuk mencapai suatu tujuan.

Contoh :

- Sistem Komputer terdiri dari : *Software, Hardware*, *Brainware*.
- Sistem Akuntansi

LUDWIG VON BARTALANFY.

Sistem merupakan seperangkat unsur yang saling terikat dalam suatu antar relasi diantara unsur-unsur tersebut dengan lingkungan.

ANATOL RAPOROT.

Sistem adalah suatu kumpulan kesatuan dan perangkat hubungan satu sama lain.

L. ACKOF.

Sistem adalah setiap kesatuan secara konseptual atau fisik yang terdiri dari bagian-bagian dalam keadaan saling tergantung satu sama lainnya[9].

2.5. Informasi

Menurut Gordon B. Davis, Informasi merupakan data yang telah diolah menjadi sebuah bentuk yang berarti bagi penerimanya dan bermanfaat dalam mengambil keputusan saat ini atau saat akan mendatang [9].

2.6. Sistem Informasi

Sistem adalah kumpulan dari komponen-komponen peralatan model r*equirements*, *function* dan *interface*. Sistem informasi merupakan kombinasi teratur apapun dari sumber daya manusia, *Hardware*, *Software*, data, dan jaringan[9].

Daur hidup suatu sistem meliputi:

1. Investigasi
2. Analisis
3. Desain
4. Implementasi
5. Pemeliharaan

2.7. *Global Positioning System* (*GPS*)

Sistem Pemosisi Global (bahasa Inggris: *Global Positioning System* (*GPS*)) adalah sistem untuk menentukan letak di permukaan bumi dengan bantuan penyelarasan (*synchronization*) sinyal satelit. Sistem ini menggunakan 24 satelit yang mengirimkan sinyal gelombang mikro ke Bumi. Sinyal ini diterima oleh alat penerima di permukaan, dan digunakan untuk menentukan letak, kecepatan, arah,

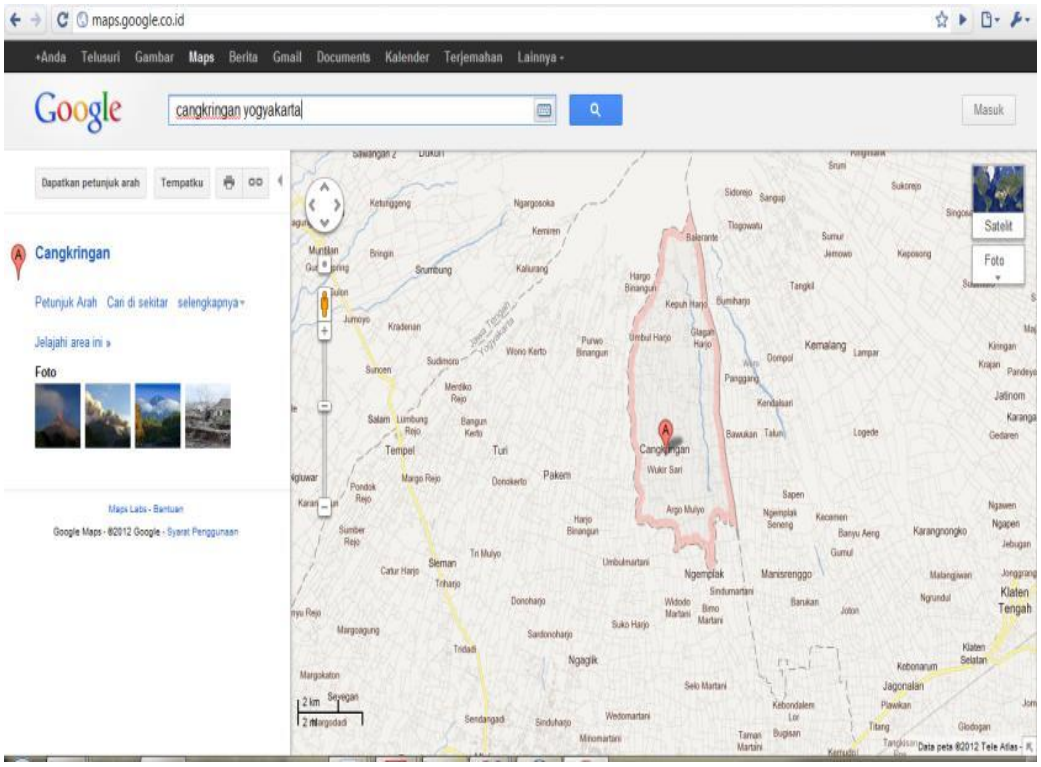
dan waktu. Sistem yang serupa dengan *GPS* antara lain GLONASS Rusia, Galileo Uni Eropa, IRNSS India.

Sistem ini dikembangkan oleh Departemen Pertahanan Amerika Serikat, dengan nama lengkapnya adalah NAVSTAR *GPS* (kesalahan umum adalah bahwa NAVSTAR adalah sebuah singkatan, ini adalah salah, NAVSTAR adalah nama yang diberikan oleh John Walsh, seorang penentu kebijakan penting dalam program *GPS*). Kumpulan satelit ini diurus oleh 50th Space Wing Angkatan Udara Amerika Serikat. Biaya perawatan sistem ini sekitar US$750 juta per tahun, termasuk penggantian satelit lama, serta riset dan pengembangan[10].

2.8. *Google Maps Service*

Google Map Service adalah sebuah jasa peta global *virtual* gratis dan *online* yang disediakan oleh perusahaan *Google. Google Maps yang* dapat ditemukan di alamat *http://maps.google.com*.*Google Maps* menawarkan peta yang dapat diseret dan gambar satelit untuk seluruh dunia.*Google Maps* juga menawarkan pencarian suatu tempat dan rute perjalanan.

Google Maps API adalah sebuah layanan (*service*) yang diberikan oleh *Google* kepada para pengguna untuk memanfaatkan *Google Map* dalam mengembangkan aplikasi. *Google Maps API* menyediakan beberapa fitur untuk memanipulasi peta, dan menambah konten melalui berbagai jenis *services* yang dimiliki, serta mengijinkan kepada pengguna untuk membangun aplikasi *enterprise* di dalam websitenya [10].



Gambar 2. 1 Tampilan *Google* Maps [11]

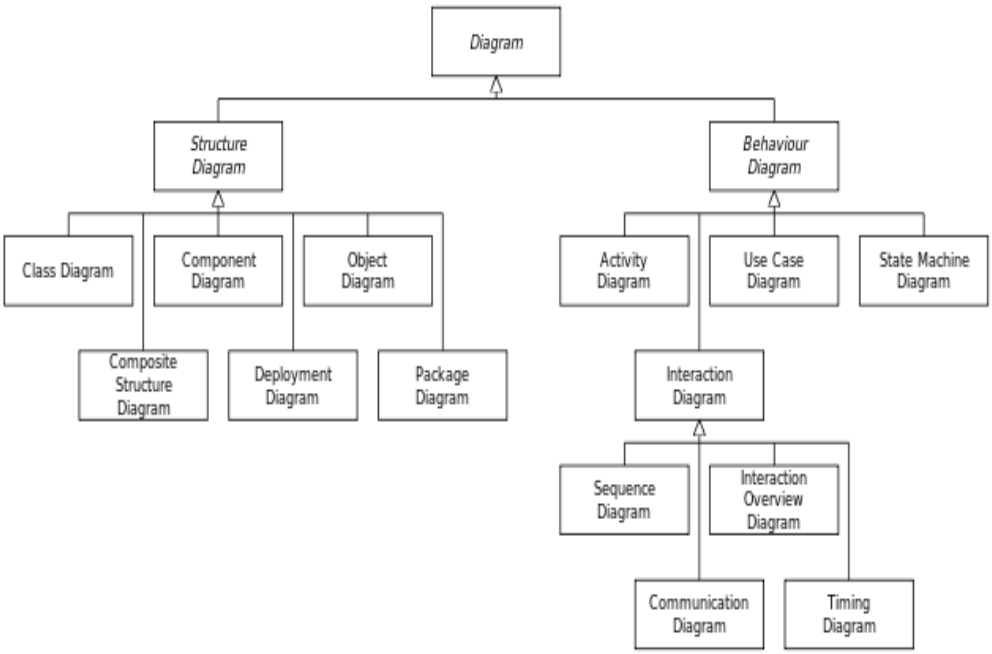
Pengguna dapat memanfaatkan layanan-layanan yang ditawarkan oleh *Google Maps* setelah melakukan registrasi dan mendapatkan *Google Maps API Key*.*Google* menyediakan layanan ini secara gratis kepada pengguna di seluruh dunia.

2.9. UML

Unified Modeling Language (UML) adalah himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya. UML adalah metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat *tool* untuk mendukung pengembangan sistem tersebut. UML mulai diperkenalkan oleh *Object Management Group*, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP. UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM.

UML adalah suatu bahasa yang digunakan untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan suatu sistem informasi. UML dikembangkan sebagai suatu alat untuk analisis dan desain berorientasi

objek oleh Grady Booch, Jim Rumbaugh, dan Ivar Jacobson. Namun demikian UML dapat digunakan untuk memahami dan mendokumentasikan setiap sistem informasi. Penggunaan UML dalam industri terus meningkat. Ini merupakan standar terbuka yang menjadikannya sebagai bahasa pemodelan yang umum dalam industri peranti lunak dan pengembangan sistem[11].



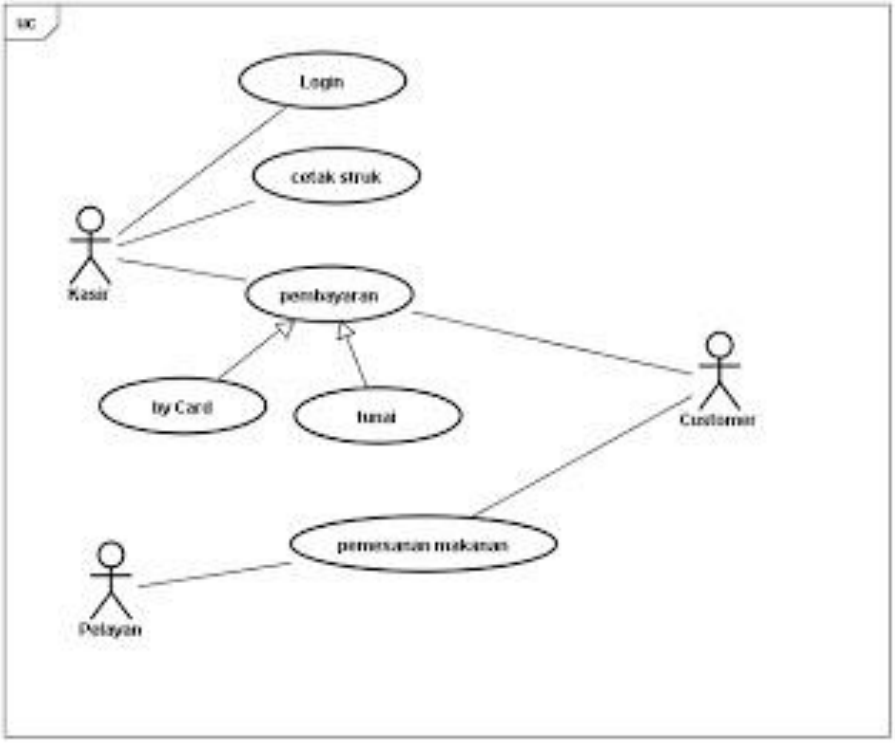
Gambar 2. 2 *Diagram* UML [12]

Diagram yang menggambarkan aktor, *use case* dan relasinya sebagai suatu urutan tindakan yang memberikan nilai terukur untuk aktor. sebuah *use case* digambarkan sebagai elips horizontal dalam suatu *diagram* UML *use case* [12].

Use case memiliki dua istilah:

1.System *use case*; interaksi dengan sistem.

2. Business *use case*; interaksi bisnis dengan konsumen atau kejadian nyata



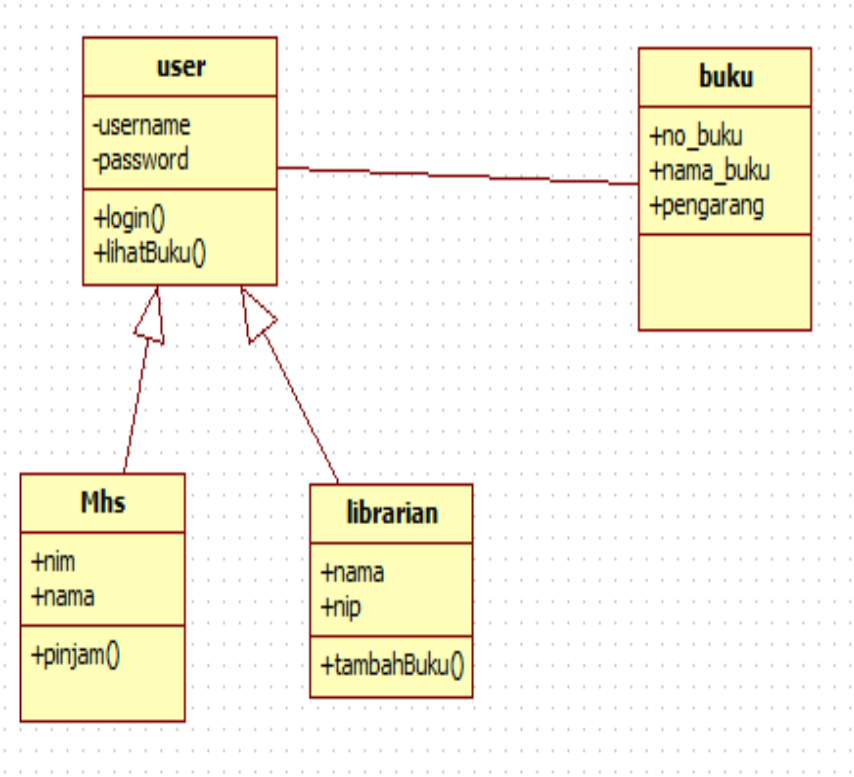
Gambar 2. 3 *Use case* Diagram

2.9.1. *Class* Diagram

Class diagram menggambarkan struktur statis dari kelas dalam sistem anda dan menggambarkan atribut, operasi dan hubungan antara kelas. *Class diagram* membantu dalam memvisualisasikan struktur kelas-kelas dari suatu sistem dan merupakan tipe *diagram* yang paling banyak di pakai. Selama tahap desain, *Class diagram* berperan dalam menangkap struktur dari semua kelas yang membentuk arsitektur sistem yang dibuat [12].

Class memiliki tiga area pokok :

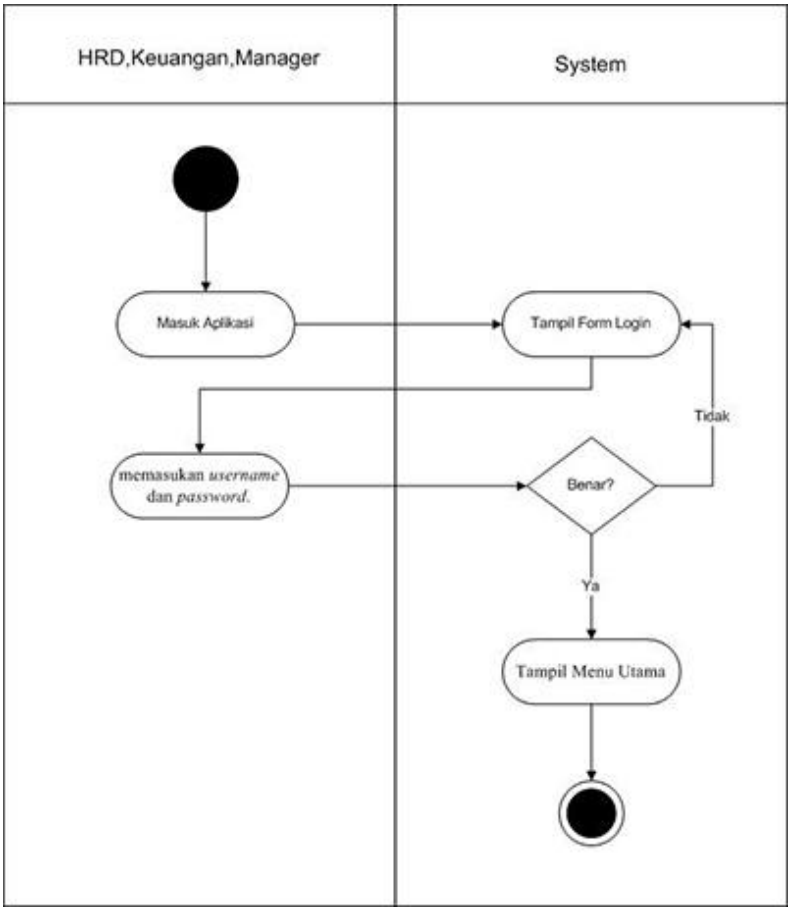
1. Nama (dan stereotype)
2. Atribut
3. Metoda



Gambar 2. 4 *Class* Diagram

2.9.2. *Activity* Diagram

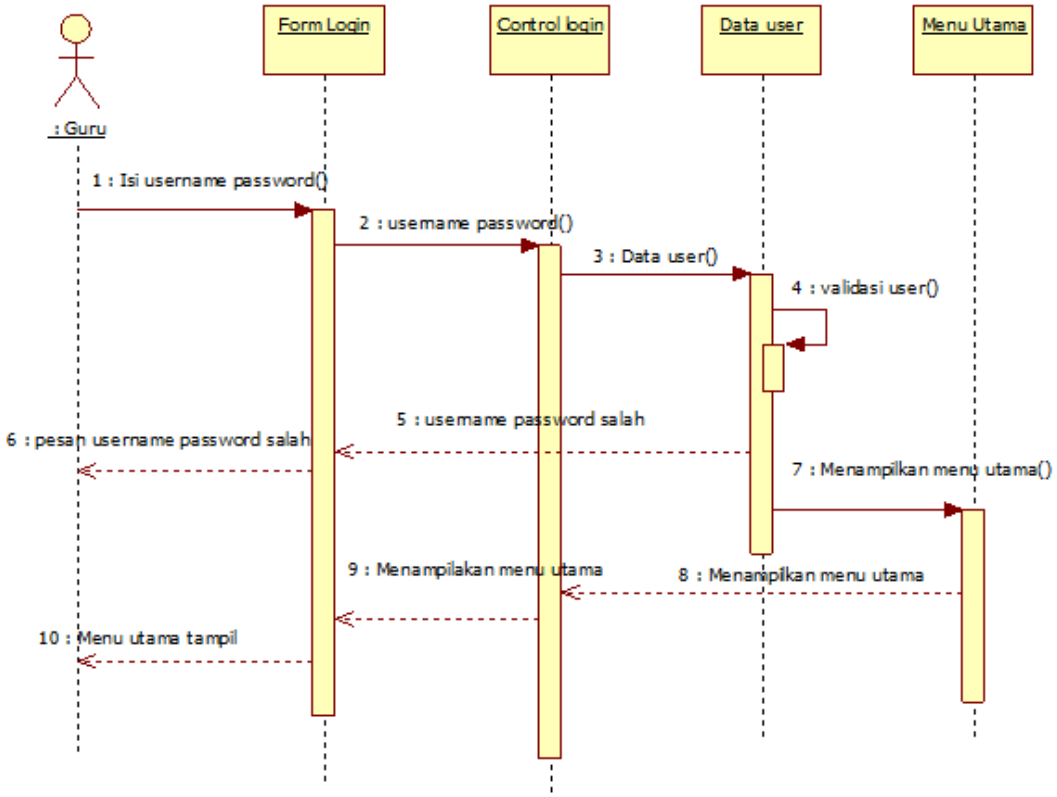
Menggambarkan aktifitas-aktifitas, objek, *state*, transisi state dan *event*. Dengan kata lain kegiatan *diagram* alur kerja menggambarkan perilaku sistem untuk aktivitas [12].



Gambar 2. 5 *Activity* Diagram

2.9.3. *Sequence* Diagram

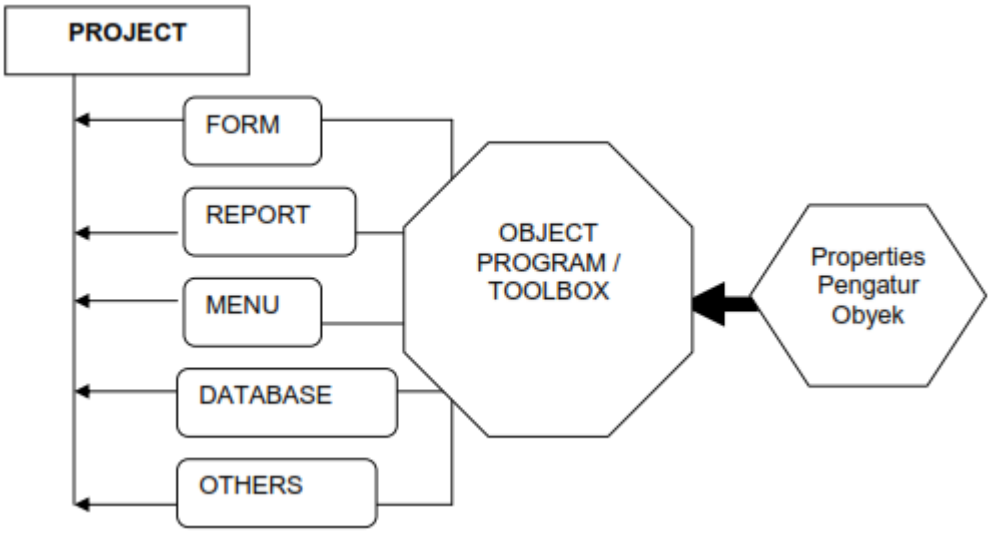
Sequence diagram menjelaskan interaksi objek yang disusun berdasarkan urutan waktu. Secara mudahnya sequence *diagram* adalah gambaran tahap demi tahap, termasuk kronologi (urutan) perubahan secara logis yang seharusnya dilakukan untuk menghasilkan sesuatu sesuai dengan *use case diagram* [12].



Gambar 2. 6 Sequence Diagram

2.10. Object Oriented Programming (OOP)

OOP (*Object Oriented Programming*) adalah sebuah istilah yang diberikan kepada bahasa pemrograman yang menggunakan tehnik berorientasi atau berbasis pada sebuah obyek dalam pembangunan program aplikasi, maksudnya bahwa orientasi pembuatan program tidak lagi menggunakan orientasi linear melainkan berorientasi pada objek-objek yang terpisah-pisah. Suatu perintah dalam bahasa ini diwakili oleh sebuah Obyek yang didalamnya berisi beberapa perintah-perintah standar sederhana. Obyek ini dikumpulkan dalam Modul form atau *Report* atau modul lain dan disusun didalam sebuah *project* [13]. Gambaran tentang pemrograman ini seperti dibawah ini :



Gambar 2. 7 Model Pemrograman OOP

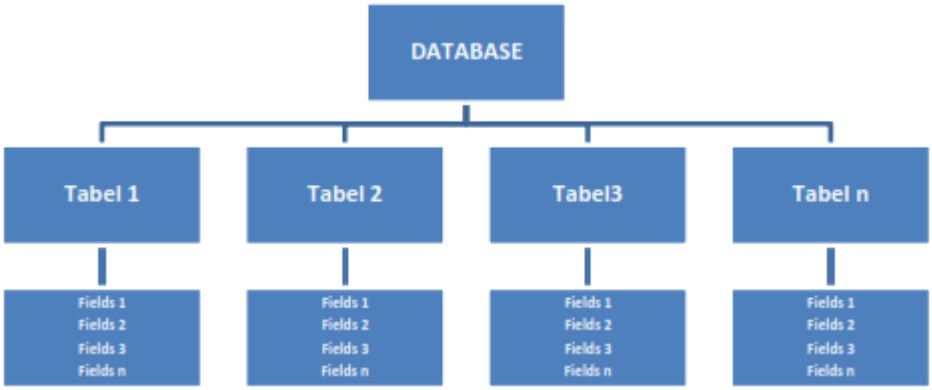
2.11. *Database*

Database adalah sebuah fasilitas pengolahan *Database* didalam program. Banyak sekali fasilitas yang disediakan untuk melakukan pengelolaan *Database* seperti melakukan pengurutan dengan kunci yang bertingkat, hubungan antar table (relasi), melakukan filter dengan SQL, membuat grid dengan cepat dan beberapa fasilitas *Database* yang lain[13].

Susunan *Database* dalam pemrograman ini sudah terstruktur yang terdiri dari :

1. File *Database*, yang berisi tabel-tabel yang diperlukan program
2. Didalam tabel terdapat struktur dari tabel tersebut atau dikenal dengan nama *Field*
3. Masing-masing *Field* menyimpan data-data tersendiri

Adapun gambaran tentang *Database* pemrograman ini seperti pada gambar dibawah ini :



Gambar 2. 8 Struktur *Database* Pemrograman OOP[13]

2.12. *GPS* (*Global Positioning System*)

Global Positioning System adalah sistem navigasi berbasis satelit terdiri dari jaringan 24 satelit ditempatkan ke orbit oleh Departemen Pertahanan Amerika Serikat yang pertama kali diperkenalkan mulai tahun 1978. Layanan *GPS* dahulu hanya dipergunakan untuk keperluan militer namun mulai terbuka untuk publik. 24 satelit *GPS* tersebut berada sekitar 12.000 mil di atas bumi bergerak mengelilingi bumi 12 jam dengan kecepatan 7.000 mil per jam. Satelit *GPS* berkekuatan energi sinar matahari, memiliki baterai cadangan untuk menjaga agar tetap berjalan pada saat gerhana matahari atau pada saat tidak ada energi matahari dan memiliki roket penguat kecil pada masing-masing satelit agar dapat mengorbit tepat pada tempatnya.

Satelit-satelit *GPS* harus selalu berada pada posisi orbit yang tepat untuk menjaga akurasi data yang dikirim ke *GPS receiver*, sehingga harus selalu dipelihara agar posisinya tepat. Posisi satelit-satelit tersebut selalu dipantau oleh stasiun pengendali. Stasiun-stasiun pengendali di bumi ada di Hawaii, Ascension Islan, Diego Garcia, Kwajalein dan Colorado Spring. Untuk dapat mengetahui posisi seseorang maka diperlukan alat yang diberi nama *GPS receiver* yang berfungsi untuk menerima sinyal yang dikirim dari satelit *GPS*. *GPS receiver* mengambil informasi tersebut danmelakukan perhitungan triangulation untuk menentukan lokasi pengguna dengan tepat. *GPS receiver* membandingkan waktu sinyal dikirim dengan waktu sinyal tersebut diterima untuk mengetahui jarak satelit. Dengan mengetahui jarak tersebut, *GPS receiver* dapat melakukan

perhitungan dan menentukan posisi pengguna dan menampilkan dalam peta elektronik. Setelah menentukan posisi pengguna, selanjutnya *GPS* dapat menghitung informasi lain, seperti kecepatan, arah yang dituju, jalur, tujuan perjalanan, jarak tujuan, matahari terbit dan matahari terbenam dan masih banyak lagi.

Satelit *GPS* sangat presisi dalam mengirim informasi karena satelit tersebut menggunakan jam atom. Jam atom yang ada pada satelit merupakan partikel atom yang diisolasi, sehingga dapat menghasilkan jam yang akurat dibandingkan dengan jam biasa. Keistimewaan *GPS* adalah mampu bekerja dalam berbagai kondisi cuaca, siang atau malam. Keakuratan sebuah perangkat *GPS* bisa mencapai 15 meter, bahkan model terbaru yang dilengkapi teknologi Wide Area Augmentation System (WAAS) keakuratannya sampai 3 meter.Karena *GPS* bekerja mengandalkan satelit, maka penggunaannya disarankan di tempat terbuka. Penggunaan di dalam ruangan, atau di tempat yang menghalangi arah satelit (di angkasa), maka *GPS* tidak akan bekerja secara akurat dan maksimal. Perhitungan waktu yang akurat sangat menentukan akurasi perhitungan untuk
menentukan informasi lokasi. Semakin banyak sinyal satelit yang dapat diterima maka akan semakin presisi data posisi yang dihasilkan. Selain itu, ketinggian juga mempengaruhi proses kerja *GPS*, Karena semakin tinggi maka semakin bersih atmosfer, sehingga gangguan semakin sedikit (Hasanuddin Z. Abidin , 1997) [14].

2.13. *Google Geocoding*

Google Maps sudah memiliki fitur yang cukup lengkap, termasuk mencari lokasikota pada peta (*Geocoding*). *Geocoding* adalah proses menemukan koordinat geografis, yaitu bujur dan lintang, dari data geografis lain, misalnya alamat (AmriShodiq, 2009). Fitur *Geocoding* dapat dimanfaatkan untuk menambahkan informasi *latitude* dan *longitude* pada foto yang tidak memiliki data *Geotagging* [15].

2.14. Pengertian Eclipse

Eclipse adalah sebuah IDE *(Integrated Development Environment)* untuk mengembangkan perangkat lunak dan dapat dijalankan di semua *platform* (platform-independent) [4]. Berikut ini adalah sifat dari Eclipse:

a. *Multi-platform*: Target sistem operasi *Eclipse* adalah *Microsoft Windows,Linux, Solaris, AIX, HP-UX* dan *Mac OS X*.

b. *Mulit-language: Eclipse* dikembangkan dengan bahasa pemrograman *Java*, akan tetapi *Eclipse* mendukung pengembangan aplikasi berbasis Bahasa pemrograman lainnya, seperti *C/C++, Cobol, Python, Perl, PHP*, dan lain sebagainya.

c. Multi-role: Selain sebagai IDE untuk pengembangan aplikasi, *Eclipse* pun bisa digunakan untuk aktivitas dalam siklus pengembangan perangkat lunak, seperti dokumentasi, test perangkat lunak, pengembangan web, dan lain sebagainya [16].

2.15. Arsitektur Eclipse

Sejak versi 3.0, *Eclipse* pada dasarnya merupakan sebuah kernel, yang mengangkat *plug-in*. Apa yang dapat digunakan di dalam *Eclipse* sebenarnya adalah fungsi dari *Plug-in* yang sudah diinstal. Ini merupakan basis dari *Eclipse* yang dinamakan Rich Client *Platform* (RCP). Berikut ini adalah komponen yang membentuk RCP:

1. Core *platform*
2. OSGi
3. SWT (Standard Widget Toolkit)
4. JFace
5. *Eclipse Workbench*

Secara standar *Eclipse* selalu dilengkapi dengan JDT (Java *Development Tools*), *Plug-in* yang membuat *Eclipse* kompatibel untuk mengembangkan program Java, dan PDE (*Plug-in Development* Environment) untuk mengembangkan *Plug-in* baru. *Eclipse* beserta plug-in-nya diimplementasikan dalam bahasa pemrograman Java.

Konsep *Eclipse* adalah IDE yang terbuka (*open*), mudah diperluas (*extensible*) untuk apa saja, dan tidak untuk sesuatu yang spesifik. Jadi, *Eclipse* tidak saja

untuk mengembangkan program Java, akan tetapi dapat digunakan untuk berbagai macam keperluan, cukup dengan menginstal *Plug-in* yang dibutuhkan. Apabila ingin mengembangkan program C/C++ terdapat *Plug-in* CDT (C/C++ *Development Tools*). Selain itu, pengembangan secara visual bukan hal yang tidak mungkin oleh Eclipse, *Plug-in* UML2 tersedia untuk membuat *diagram* UML. Dengan menggunakan PDE setiap orang bisa membuat *Plug-in* sesuai dengan keinginannya [16].

2.16. Perkembangan Eclipse

Sejak tahun 2006, *Eclipse* Foundation mengkoordinasikan peluncuran *Eclipse* secara rutin dan simultan yang dikenal dengan nama *Simultaneous Release*. Setiap versi peluncuran terdiri dari *Eclipse Platform* dan juga sejumlah proyek yang terlibat dalam proyek Eclipse. Tujuan dari sistem ini adalah untuk menyediakan distribusi *Eclipse* dengan fitur-fitur dan versi yang terstandarisasi. Hal ini juga dimaksudkan untuk mempermudah deployment dan maintenance untuk sistem enterprise[16]. Adapun versi *Eclipse* yang telah diluncurkan yaitu :

1. *Eclipse* 3.0
2. *Eclipse* 3.1
3. Callisto
4. Europa
5. Ganymede
6. Galileo
7. Helios
8. Indigo
9. Juno
10. Kepler

2.17. *Software Development* Kit (SDK)

Software Development Kit (SDK) adalah suatu kit atau *library* dari Bahasa pemrograman untuk pengembangan atau pembangunan suatu perangkat lunak dan biasanya SDK terdiri dari kumpulan *Tools* yang dibutuhkan. Misalnya Bahasa pemrograman java, mempunyai SDK yang berisi suatu *library* yang dapat digunakan untuk membuat suatu aplikasi berbasis java [16].

2.18. Java Development Kit (JDK)

Java *Development* Kit (JDK) adalah sekumpulan perangkat lunak yang dapat kamu gunakan untuk mengembangkan perangkat lunak yang berbasis Java, Sedangkan JRE adalah sebuah implementasi dari Java *Virtual Machine* yang benar-benar digunakan untuk menjalankan program java.Biasanya, setiap JDK berisi satu atau lebih JRE dan berbagai alat pengembangan lain seperti sumber kompiler java, bundling, *debuggers*, *Development* libraries dan lain sebagainya. Perbedaan JDK dengan SDK (*Software Development* Kit) yaitu JDK adalah sebuah SDK tetapi sebuah SDK tidak harus menjadi sebuah JDK [16].

2.19. *Android Development Tools* (ADT)

Android Development Tools (ADT) adalah plugin untuk *Eclipse* Intergrated *Development Environment* (IDE) yang dirancang untuk memberikan lingkungan yang terpadu di mana untuk membangun aplikasi *Android*. ADT memperluas kemampuan *Eclipse* untuk membiarkan para *developer* lebih cepat dalam membuat proyek baru *Android*, membuat aplikasi UI, menambahkan komponen berdasarkan *Android Framework* API, debug aplikasi dalam pengunaan *Android* SDK, dan membuat file APK untuk mendistribusikan aplikasi. Mengembangkan aplikasi di *Eclipse* dengan ADT sangat dianjurkan dan merupakan cara tercepat untuk memulai membuat aplikasi *Android*, karena banyak kemudahan-kemudahan sebagai *Tools* yang terintegrasi seperti, custom XML editor, dan debug panel *ouput*. Selain itu ADT memberikan dorongan luar biasa dalam mengembangkan aplikasi *Android* [16].

2.20. Java

Java adalah bahasa pemrograman berorientasi objek murni yang dibuat berdasarkan kemampuan-kemampuan terbaik bahasa pemrograman objek sebelumnya (C++, Ada, Simula). Java diciptakan oleh James Gosling, *developer* dari Sun Microsystems pada tahun 1991.

Java pertama kali diluncurkan sebagai bahasa pemrograman umum (*general purpose programming language*) dengan kelebihan dia bisa dijalankan di web *browser* sebagai *applet*. Sejak awal, para pembuat *Java* telah menanamkan visi mereka ke dalam *Java* untuk small embedded customer device)seperti TV,

telepon, radio, dan sebagainya supaya dapat berkomunikasi satu sama lain. Langkah pertama yang diambil oleh Sun Microsistem adalah dengan membuat JVM (*JavaVirtual machine*) yang kemudian diimplementasikan dalam bentuk JRE (*Java Runtime Environment*). JVM adalah lingkungan tempat eksekusi program *Java* berlangsung dimana para obyek saling berinteraksi satu dengan yang lainnya. *Virtual Machine* inilah yang menyebabkan *Java* mempunyai kemampuan penanganan memori yang lebih baik, keamanan yang lebih tinggi serta portabilitas yang besar. Apabila kita hanya ingin menjalankan program *Java*, maka kita cukup memiliki JRE saja.Tapi seandainya kita ingin mengembangkan perangkat lunak sendiri, JRE saja tidak cukup.

Untuk lebih meningkatkan produktivitas pengembang perangkat lunak, Sun juga meluncurkan SDK (*Standard Development Kit*) yang berisi kakas dan API untuk membuat program aplikasi berbasis *Java*. Pada tahun 1999 Sun meluncurkan J2EE (*Java 2 Enterprise Edition*) sebagai *Framework* untuk membuat aplikasi *enterprise* berskala besar. Pada tahun 2001, Sun meluncurkan J2ME yang kelak menjadi salah satu standar pemrograman di dalam PDA maupun *handphone*.

Sederhana, semudah C dan seampuh C++: berlawanan dengan anggapan orang-orang bahwa bahasa *Java* sulit untuk dipelajari, *Java* gampang untuk dipelajari terutama untuk orang yang sudah mengenal pemrograman tapi belum terlalu terikat pada paradigma pemrograman prosedural. Tentu saja ini berarti bahwa kita harus siap mempelajari salah satu teknologi yang berkembang paling cepat di dunia dalam dua tahun terakhir ini dengan banyak membaca tentunya baik dari buku maupun melalui web.

Sangat berorientasi obyek (OOP) dengan implementasi yang sangat baik sehingga kita bukan hanya belajar bagaimana membuat program yang baik (*reusable, scalable*, dan *maintanable*) tetapi juga kita belajar bagaimana cara berfikir yang baik untuk mengenali struktur masalah yang sedang kita hadapi dan memecahkannya secara sistematis dengan pola-pola tertentu (*patterns OpenPlatform, Write Once Run Anywhere* (WORA), *portable* atau *multiplatform*, program yang kita buat dapat dijalankan di Windows, Linux/Unix, Solaris, dan

Macintosh tanpa perlu diubah maupun di kompilasi ulang. *Java* adalah juga bahasa yang paling sesuai digunakan bersama dengan XML yang membuat data menjadi *portable*, ini karena kelahiran XML tidak terlepas dari dukungan parser-parser berbahasa *Java*.

Arsitekturnya yang kokoh dan pemrograman yang aman. Dalam *Java* program yang kita buat tidak mudah untuk "*hang*" karena konflik pada memori biasanya diselesaikan dengan mengumpulkan obyek-obyek yang sudah tak terpakai lagi secara otomatis oleh *garbage collector*. Penanganan kesalahan juga dipermudah dalam *Java* dengan konsep *Exception*[17].

1. Bukan sekedar bahasa tapi juga *platform* sekaligus arsitektur. *Java* mempunyai portabilitas yang sangat tinggi. Ia dapat berada pada *smartcard*, *pager*, POS (*Point of Service*), *handphone*, PDA, palm, TV, *Embedded device* (PLC, *micro controller*), laptop, pc, dan bahkan server). Menyadari akan hal ini Sun membagi arsitektur *Java* membagi tiga bagian, yaitu:

a. *Enterprise Java* (J2EE) untuk aplikasi berbasis web, aplikasi sistem tersebar dengan beraneka ragam klien dengan kompleksitas yang tinggi. Merupakan superset dari Standar *Java*

b. *Standard Java* (J2SE), ini adalah yang biasa kita kenal sebagai bahasa *Java*, dan merupakan fokus kita sekarang.

c. *Micro Java* (J2ME) merupakan subset dari J2SE dan salah satu aplikasinya yang banyak dipakai adalah untuk *wireless device*/*Mobile device*

2. Program *Java* dijalankan menggunakan *interpreter* melalui *Java Virtual Machine* (JVM). Hal ini menyebabkan *source codeJava* yang telah dikompilasi menjadi *Java bytecodes*dapat dijalankan pada *platform* yang berbeda-beda.

3. Fitur-fitur utama yang lain:

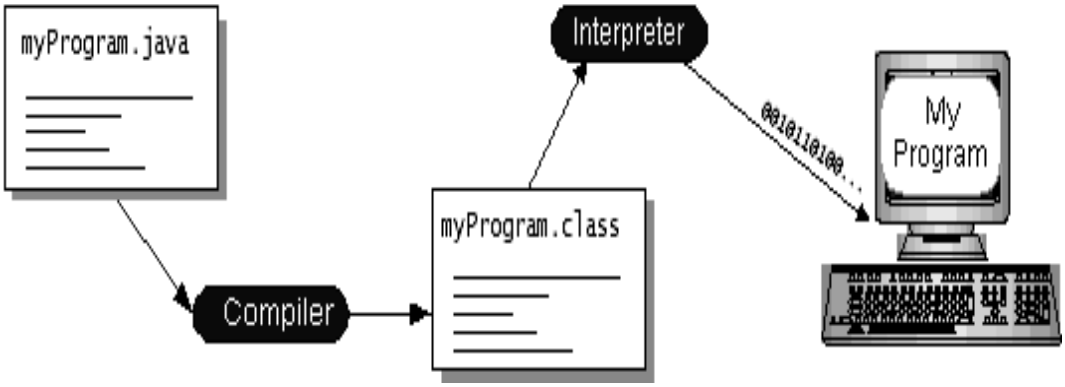
   a. Mendukung *multithreading.*

   b. Selalu memeriksa tipe obyek pada saat *runtime.*

   c. Mempunyai *automatic garbage collection* untuk membersihkan obyek yang tidak terpakai dari memori

   d. Mendukung *exception* sebagai salah satu cara penanganan kesalahan

   Berikut Mekanisme kompilasi Bahasa Java Bekerja



Gambar 2. 9 Mekanisme kompilasi dan eksekusi program Java

2.21. Pengujian *Software*

Pengujian *Software* sangat diperlukan untuk memastikan *Software*/aplikasi yang sudah/sedang dibuat dapat berjalan sesuai dengan fungsionalitas yang diharapkan. Pengembang atau penguji *Software* harus menyiapkan sesi khusus untuk menguji program yang sudah dibuat agar kesalahan ataupun kekurangan dapat dideteksi sejak awal dan dikoreksi secepatnya. Pengujian atau *testing* sendiri merupakan elemen kritis dari jaminan kualitas perangkat lunak dan merupakan bagian yang tidak terpisah dari siklus hidup pengembangan *Software* seperti halnya analisis, desain, dan pengkodean. (Shi, 2010).

Pengujian *Software* haruslah dilakukan dalam proses rekayasa perangkat lunak atau *Software engineering*. Sejumlah strategi pengujian *Software* telah diusulkan dalam literatur. Semuanya menyediakan *template* untuk pengujian bagi pembuat *Software*. Dalam hal ini, semuanya harus memiliki karakteristik umum berupa (Bhat and Quadri, 2015) :

1. *Testing* dimulai pada level modul dan bekerja keluar ke arah integrasi pada sistem berbasiskan komputer
2. Teknik *testing* yang berbeda sesuai dengan poin-poin yang berbeda pada waktunya
3. *Testing* diadakan oleh pembuat/pengembang *Software* dan untuk proyek yang besar oleh group *testing* yang independent
4. *Testing* dan *Debugging* adalah aktivitas yang berbeda tetapi *debugging* harus diakomodasikan pada setiap strategi *testing*

Pengujian *Software* adalah satu elemen dari sebuah topik yang lebih luas yang sering diartikan sebagai Verifikasi dan Validasi (V&V)

- Verifikasi: menunjuk kepada kumpulan aktifitas yang memastikan bahwa *Software* telah mengimplementasi sebuah fungsi spesifik.
- Validasi: menunjuk kepada sebuah kumpulan berbeda dari aktivitas yang memastikan bahwa *Software* yang telah dibangun dapat ditelusuri terhadap kebutuhan customer.

Definisi V&V meliputi banyak aktifitas SQA (*Software quality assurance*), termasuk review teknis formal, kualitas dan audit konfigurasi, monitor performance. Terdapat beberapa tipe yang berbeda dalam pengujian *Software* yang meliputi studi kelayakan dan simulasi. (Bhat and Quadri, 2015):

1. Metode *Software engineering* menyediakan dasar dari mutu yang mana yang akan dipakai.
2. Metode *Analysis, design and Construction* berupa tindakan untuk meningkatkan kualitas dengan menyediakan teknik yang seragam dan hasil yang sesuai dengan keinginan.
3. Metode *Formal Technical Reviews* menolong untuk memastikan kualitas kerja produk merupakan hasil konsekuensi dari setiap langkah *Software engineering*.
4. Metode *Measurement* diberlakukan pada setiap elemen dari konfigurasi *Software*
5. Metode *Standards and Procedures* membantu untuk memastikan keseragaman dan formalitas dari SQA untuk menguatkan dasar "filosofi kualitas total"
6. Metoda *Testing* menyediakan cara terakhir dari tingkat kualitas mana yang dapat dicapai dan dengan praktis dapat mengetahui letak error.

Davids menyarankan satu set prinsip pengujian:

1. Semua test harus dapat dilacak ke kebutuhan pelanggan.
2. Test harus direncanakan dengan baik sebelum pengujian mulai.

a. Prinsip Pareto berlaku untuk pengujian

b. 80% dari semua kesalahan yang terungkap selama pengujian akan mudah dapat dilacak dari 20% semua modul program.

3. Pengujian seharusnya mulai "dari yang kecil" dan pengujian perkembangan ke arah "yang besar".

4. Pengujian menyeluruh adalah tidak mungkin. Paling efektif, pengujian harus diselenggarakan oleh suatu pihak ketiga mandiri.

Langkah-langkah pengujian *Software* ada 4 yaitu:

1. *Unit testing-testing* per unit yaitu mencoba alur yang spesifik pada struktur modul kontrol untuk memastikan pelengkapan secara penuh dan pendeteksian error secara maksimum
2. *Integration testing – testing* per penggabungan unit yaitu pengalamatan dari isu-isu yang diasosiasikan dengan masalah ganda pada verifikasi dan konstruksi program
3. *High-order test* yaitu terjadi ketika *Software* telah selesai diintegrasikan atau dibangun menjadi satu –tidak terpisah-pisah
4. *Validation test* yaitu menyediakan jaminan akhir bahwa *Software* memenuhi semua kebutuhan fungsional, kepribadian dan performa.

Tom Gilb menyatakan bahwa prosedur yang harus digunakan jika ingin mengimplementasikan strategi *testing Software* yang sukses (Bhat and Quadri, 2015):

1. Menetapkan seluruh kebutuhan produk *Software* dalam perhitungan sebelum memulai *testing*
2. Status obyek *testing* harus jelas
3. Memahami pengguna *Software* dan mengembangkan sebuah profil untuk setiap kategori user
4. Mengembangkan rencana *testing* yang menekankan pada "*rapid cycle testing*"
5. Membangun *Software* yang sempurna yang didesain untuk menguji dirinya sendiri
6. Menggunakan tinjauan ulang yang formal sebagai filter sebelum pengujian
7. Melakukan tinjauan ulang secara formal untuk menilai strategi tes dan kasus tes itu sendiri

8. Mengembangkan pendekatan peningkatan yang berkelanjutan untuk proses testing

Ada beberapa jenis pengujian perangkat lunak antara lain (Khan, 2011):

1. Pengujian *white box* adalah pengujian yang didasarkan pada pengecekan terhadap detail perancangan, menggunakan struktur control dari desain program secara prosedural untuk membagi pengujian ke dalam beberapa kasus pengujian. Secara sekilas dapat diambil kesimpulan *white box testing* merupakan petunjuk untuk mendapatkan program yang benar secara 100%,
2. *Black-Box Testing* merupakan pengujianyang berfokus pada spesifikasi fungsional dari perangkat lunak, tester dapat mendefinisikan kumpulan kondisi input dan melakukan pengetesan pada spesifikasi fungsional program.

2.21.1. Pengujian *Blackbox*

White Box Testing adalah salah satu cara untuk menguji suatu aplikasi atau *Software* dengan cara melihat modul untuk dapat meneliti dan menganalisa kode dari program yang di buat ada yang salah atau tidak. Kalau modul yang telah dan sudah di hasilkan berupa output yang tidak sesuai dengan yang di harapkan maka akan dikompilasi ulang dan di cek kembali kode-kode tersebut hingga sesuai dengan yang diharapkan (Nidhra and Dondetti, 2012).

Kasus yang sering menggunakan *white box testing* akan di uji dengan beberapa tahapan yaitu:

1. Pengujian seluruh keputusan yang menggunakan logikal.
2. Pengujian keseluruh loop yang ada sesuai batasan-batasannya.
3. Pengujian pada struktur data yang sifatnya internal dan yang terjamin validitasnya.

Kelebihan *White Box Testing* antara lain (Nidhra and Dondetti, 2012) :

1. Kesalahan Logika

   Menggunakan sintax ‘if’ dan sintax pengulangan. Langkah selanjutnya metode*white box testing* ini akan mencari dan mendeteksi segala kondisi yang di percaya tidak sesuai dan mencari kapan suatu proses perulangan di akhiri.

2. Ketidaksesuaian Asumsi

Menampilkan dan memonitor beberapa asumsi yang diyakini tidak sesuai dengan yang diharapkan atau yang akan diwujudkan, untuk selanjutnya akan dianalisa kembali dan kemudian diperbaiki.

3. Kesalahan Pengetikan

Mendeteksi dan mencaribahasa-bahasa pemograman yang di anggap bersifat *case sensitif*.

Kelemahan *White Box Testing* adalah pada perangkat lunak yang jenisnya besar, metode *white box testing* ini dianggap boros karena melibatkan banyak sumberdaya untuk melakukannya. (Nidhra and Dondetti, 2012)