BAB III

LANDASAN TEORI

3.1. Aplikasi

Aplikasi berasal dari kata *application* yang artinya penerapan, lamaran, penggunaan. Aplikasi adalah suatu subkelas perangkat lunak komputer yang memanfaatkan kemampuan komputer langsung untuk melakukan suatu tugas yang diinginkan pengguna. Secara istilah aplikasi adalah program siap pakai yang dibuat untuk melaksanakan suatu fungsi bagi pengguna atau aplikasi yang lain dan dapat digunakan oleh sasaran yang dituju (Harahap, 2012).

Klasifikasi aplikasi ini dapat dibagi menjadi 2 (dua) yaitu:

1. Aplikasi *software* spesialis, sutu program dengan dokumentasi tergabung yang dirancang untuk menjalankan tugas tertentu.
2. Aplikasi paket, suatu program dengan dokumentasi tergabung yang dirancang untuk jenis masakah tertentu.

Macam-macam data yang digunakan untuk membuat aplikasi adalah :

1. Data Sumber (*source data*), ialah fakta yang disimpan di dalam basis data, misalnya: nama, tempat lahir, tanggal lahir, dan lain-lain.
2. *Meta Data*, digunakan untuk menjelaskan struktur dari basis data, *type* dan format penyimpanan data *item* dan berbagai pembatas (*constraint*) pada data.

3. *Data Dictionary* atau *Data Repository*, digunakan untuk menyimpan informasi katalog skema dan pembatas serta data lain seperti: pembakuan, deskripsi program aplikasi dan informasi pemakai.
4. *Overhead Data*, berisi *linked list*, indeks dan struktur data lain yang digunakan untuk menyajikan *relationship record.*

Klasifikasi aplikasi dapat digolongkan menjadi beberapa kelas, antara lain :

1. Perangkat Lunak Perusahaan (*Enterprise Software)*

   Perangkat Lunak Perusahaan (*Enterprise Software)* adalah aplikasi yang digunakan perusahaan untuk melakukan pengorganisasian kegiatan perusahaan.

2. Perangkat Lunak Infrastruktur Perusahaan (*Enterprise Infrastructure Software)*

   Perangkat Lunak Infrastruktur Perusahaan (*Enterprise Infrastructure Software)* adalah aplikasi yang dibuat untuk menyediakan kemampuan-kemampuan umum yang dibutuhkan untuk membantu perangkat lunak perusahaan (*enterprise software).*

3. Perangkat Lunak Informasi Kerja (*Information Worker Software)*

   Perangkat Lunak Informasi Kerja (*Information Worker Software)* adalah aplikasi yang biasa dipakai untuk menunjukan kebutuhan individual untuk membuat dan mengolah informasi. Umumnya untuk tugas-tugas individu dalam sebuah departemen.

4. Perangkat Lunak Media dan Hiburan *(Content Acces Software)*

   Perangkat Lunak Media dan Hiburan *(Content Acces Software)* adalah aplikasi yang biasa digunakan untuk mengakses konten tanpa *editing*, tapi bisa saja termasuk *software* yang memungkinkan mengedit konten. Seperti *software* yang menunjukan kebutuhan individu dan grup untuk mengkonsumsi hiburan digital dan mempublikasikan konten digital.

5. Perangkat Lunak Pendidikan (*Education Software*)

   Perangkat Lunak Pendidikan (*Education Software*) adalah aplikasi yang hampir sama dengan Perangkat Lunak Media dan Hiburan (*Content access software)* tapi biasanya menampilkan konten yang berbeda.

6. Perangkat Lunak Pengembangan media (*Media Development Software)*

   Perangkat Lunak Pengembangan media (*Media Development Software)* adalah aplikasi yang digunakan untuk menunjukan kebutuhan individu untuk menghasilkan media cetak dan elektronik, umumnya pada bidang komersial atau pendidikan.

7. Perangkat Lunak Pengembangan Produk (*Project Engineering Software)*

   Perangkat Lunak Pengembangan Produk (*Project Engineering Software)* adalah aplikasi yang biasa dilakukan untuk pengembangan produk *hardware* dan *software.*

3.2. Aplikasi *Mobile*

Aplikasi *mobile* dapat diartikan sebagai sebuah produk dari sistem komputasi *mobile*, yaitu sistem komputasi yang dapat dengan mudah dipindahkan secara fisik dan yang komputasi kemampuan dapat digunakan saat mereka sedang dipindahkan. Contohnya seperti *personal digital assistant* (*PDA*), *smartphone* dan ponsel (Ramadhan dan Utomo, 2014). Berdasarkan jenisnya, aplikasi *mobile* menjadi beberapa kelompok yaitu :

1. *Short Message Service (SMS)*

   Merupakan aplikasi *mobile* paling sederhana, dirancang untuk berkirim pesan dan berguna ketika terintegrasi dengan jenis aplikasi *mobile* lainnya.

2. *Mobile Website* (Situs *Web Mobile)*

   Merupakan situs *web* yang dirancang khusus untuk perangkat *mobile.* Situs *web mobile* sering memiliki desain yang sederhana dan biasanya bersifat memberikan informasi.

3. *Mobile Web Application* (Aplikasi *Web Mobile)*

   Aplikasi web *mobile* merupakan aplikasi *mobile* yang tidak perlu diinstal atau dikompilasi pada perangkat target. Menggunakan *XHTML, CSS, dan JavaScript*, aplikasi ini mampu memberikan pengguna pengalaman layaknya aplikasi *native*/asli.

4. *Native Application* (Aplikasi Asli)

   Merupakan aplikasi *mobile* yang harus diinstal pada perangkat target. Aplikasi ini dapat disebut aplikasi *platform*, karena aplikasi ini harus

dikembangkan dan disusun untuk setiap *platform mobile* secara khusus.

3.3. *Java*

Java merupakan bahasa pemrograman yang bersifat umum/non-spesifik (*general purpose*), dan secara khusus didisain untuk memanfaatkan dependensi implementasi seminimal mungkin. Karena fungsionalitasnya yang memungkinkan aplikasi *java* mampu berjalan di beberapa *platform* sistem operasi yang berbeda, *java* dikenal pula dengan slogannya, "*Tulis sekali, jalankan di mana pun*" (Kodir, 2012).

Java adalah bahasa pemrograman yang dapat dijalankan di berbagai komputer termasuk telepon genggam. Bahasa ini awalnya dibuat oleh James Gosling saat masih bergabung di *Sun Microsystems* saat ini merupakan bagian dari *Oracle* dan dirilis tahun 1995. Bahasa ini banyak mengadopsi sintaksis yang terdapat pada *C* dan *C++* namun dengan sintaksis model objek yang lebih sederhana serta dukungan rutin-rutin aras bawah yang minimal. Aplikasi-aplikasi berbasis *java* umumnya dikompilasi ke dalam *p-cod*e (*bytecode*) dan dapat dijalankan pada berbagai Mesin *Virtual Java* (*JVM*).

3.4. *JDK*

JDK (Java Development Kit) merupakan implementasi dari salah satu dari *Java Platform Standard Edition, Java Platform Enterprise Edition* atau

Java Platform Micro Edition yang dirilis oleh *Oracle Corporation* (JDK, 2017). *JDK* berjalan diatas sebuah *virtual machine* yang dinamakan *JVM* (*Java Virtual Machine*). Dokumentasi *JDK* berisi spesifikasi *API*, deskripsi fitur, panduan pengembang, referensi halaman untuk perkakas *JDK* dan *utilitas*, demo, dan *link* ke informasi terkait. *JDK* harus diinstal sebelum menginstal *Android SDK*. *JDK* yang dapat digunakan untuk mengkompilasi aplikasi *android* adalah *Java SE development Kit 8* atau versi terbarunya

3.5. *Android SDK*

Menurut Safaat (2015) *Android* SDK adalah *tools Android API* (*Application Programming Interface*) yang diperlukan untuk mulai mengembangkan aplikasi pada *platform Android* menggunakan bahasa pemrograman *Java*. *Android* merupakan *subset* perangkat lunak untuk ponsel meliputi sistem operasi, *middleware* dan aplikasi kunci yang dirilis oleh *Google*. Saat ini disediakan *Android* SDK (*Software Development Kit*) sebagai alat bantu dan *API* untuk mulai mengembangkan aplikasi pada *platform Android* menggunakan bahasa *Java*. Sebagai *platform* aplikasi netral, *Android* memberi kesempatan untuk membuat aplikasi yang dibutuhkan yang bukan merupakan aplikasi bawaan *Handphone*/*Smartphone.*

3.6. *Android*

3.6.1 Pengertian *Android*

Android adalah sebuah sistem operasi untuk perangkat *mobile* berbasis *linux* yang mencakup sistem operasi, *middleware* dan aplikasi. Android menyediakan *platform* terbuka bagi para pengembang untuk menciptakan aplikasi mereka. Kemudian untuk mengembangkan *Android,* diperlukan *Open Handset Alliance, konsorsium* dari 34 perusahaan piranti keras, piranti lunak, dan telekomunikasi, termasuk *Google, HTC, Intel, Motorola, Qualcomm, T-Mobile,* dan *Nvidia* (Rispurwadi dkk, 2016).

Sistem operasi *Android* dibangun berdasarkan kernel Linux, dan memiliki arsitekur sebagai berikut :

1. *Applications,* Lapisan ini adalah lapisan aplikasi, serangkaian aplikasi akan terdapat pada perangkat *mobile*. Aplikasi inti yang telah terdapat pada Android termasuk kalender, kontak, SMS (*Short Message Service),* dan lain sebagainya. Aplikasi - aplikasi ini ditulis dengan bahasa pemrograman *Java.*
2. *Application Framework,* Pengembang aplikasi memiliki akses penuh ke Android sama dengan aplikasi inti yang telah tersedia. Pengembang dapat mudah mengakses informasi lokasi, mengatur *alarm*, menambah pemberitahuan ke status *bar* dan lainnya sebagainya. Arsitektur aplikasi ini dirancang untuk menyederhanakan penggunaan kembali komponen, aplikasi apa pun yang dapat memusblikasikan kemampuan dan aplikasi lainnya dapat menggunakan kemampuan sesuai batasan keamanan.

Dasar dari aplikasi adalah seperangkat layanan sistem, yaitu berbagai *view* yang digunakan untuk membangun *user interface, content provider* yang memungkinkan aplikasi berbagi data, *Resource Manage* menyediakan akses bukan kode seperti grafik, *string*, dan *layout*, *Notification Manager* yang akan membuat aplikasi dapat menampilkan tanda pada status bar dan *Activity Manager* yang berguna mengatur daur hidup dari aplikasi.

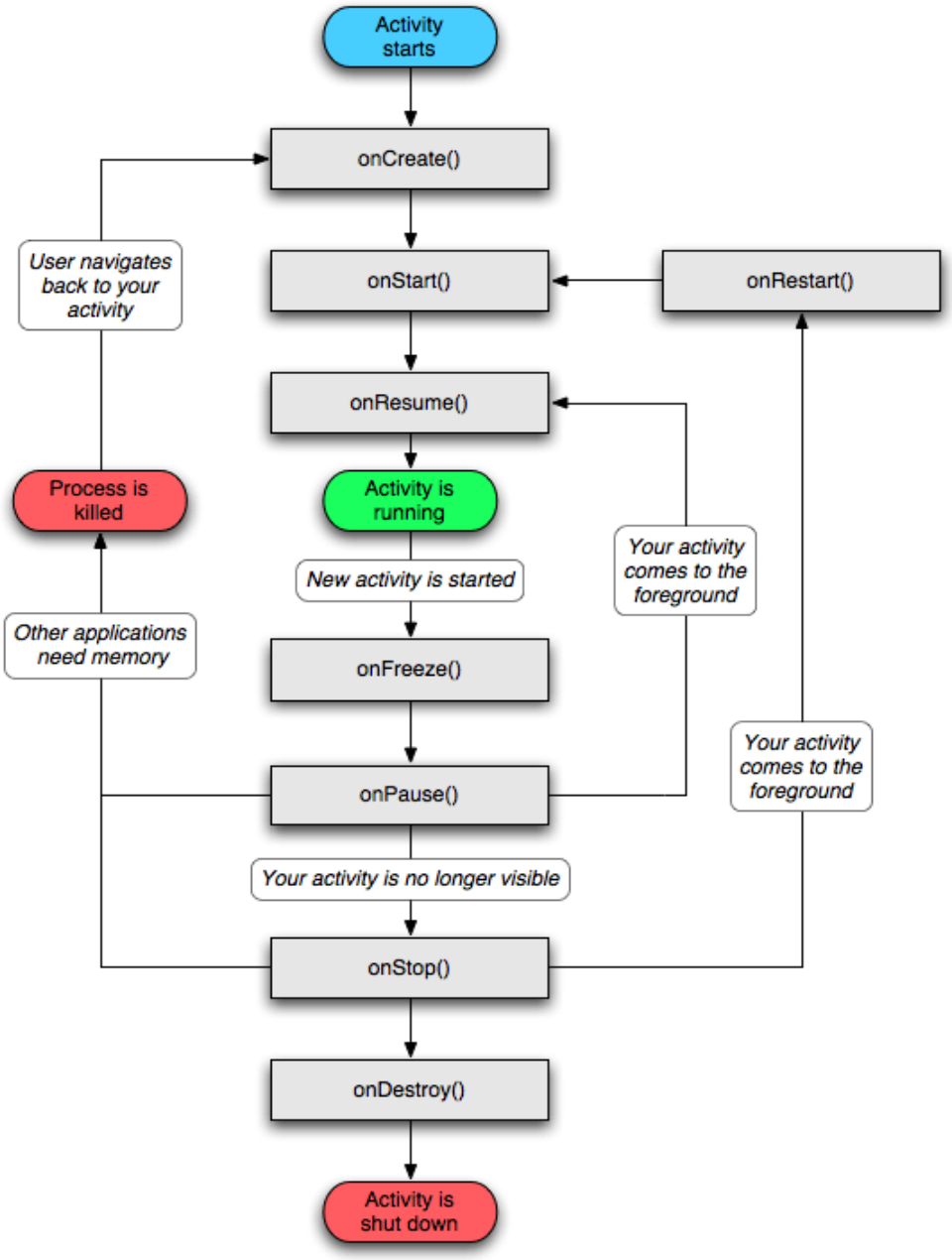
3. *Libraries,* Satu set *libraries* dalam bahasa *C/C++* yang digunakan oleh berbagai komponen pada sistem *Android.*
4. *Android Runtime,* Satu set *libraries* inti yang menyediakan sebagian besar fungsi yang tersedia di *libraries* inti dari bahasa pemrograman Java. Setiap aplikasi akan berjalan sebagai proses sendiri pada *Dalvik Virtual Machine.*
5. *Kernel Linux* dimana *Android* bergantung pada Linux versi 2.6 untuk layanan sistem initi seperti keamanan, manajemen memori, manajemen proses, *network stack*, dan model *driver.* Kernel juga bertindak sebagai lapisan antara *hardware* dan seluruh *software.*

3.6.2 Siklus Hidup *Android*

Untuk membangun atau membuat aplikasi berbasia *Android,* terdapat dua cara. Pertama, memiliki perangkat telepon seluler yang berbasis *Android* langsung. Kedua, menggunakan emulator yang sudah disediakan oleh *Google*. Aplikasi *Android* biasanya terdiri dari *activity-activity* yang saling terikat yang disebut juga *life cycle* atau siklus hidup. *Activity*

merupakan sebuah komponen yang terdapat di dalam aplikasi dan menyediakan *user interface/*antarmuka pada layar sehingga pengguna dapat melakukan interaksi dengan aplikasi yang sedang dijalankan (Elian dan Mazharuddin, 2014).

Penjelasn dari siklus hidup *Android* dapat dilihat pada gambar 3.1.



Gambar 3.1. Siklus Hidup Android

3.7. Android Studio

Gambar 3.2. Tampilan Muka Android Studio

Menurut Evitarina (2015) *Android Studio* adalah sebuah lingkungan pengembangan terpadu (*IDE*) untuk mengembangkan pada *platform Android*. *Android Studio* merupakan IDE resmi untuk pengembangan aplikasi android. Sebagai media yang menjadi pengembangan dari *Eclipse,* Android Studio tentunya sudah dilengkapi dengan gaya baru serta mempunyai banyak fitur-fitur baru dibandingkan dengan *Eclipse* IDE. Berbeda dengan *Eclipse* yang menggunakan *Ant, Android Studio* menggunakan *Gradle* sebagai *build environment.*

Android studio ini berbasis pada *IntelliJ IDEA*, sebuah *IDE* untuk bahasa pemrograman *Java*. Bahasa pemrograman utama yang digunakan adalah *Java*, sedangkan untuk membuat tampilan atau layout, digunakan bahasa *XML*. *Android studio* juga terintegrasi dengan *Android Software Development Kit (SDK*) untuk *deploy* ke perangkat *android*. *Android Studio*

menawarkan lebih banyak fitur yang meningkatkan produktivitas ketika membangun aplikasi *Android*, seperti :

1. *Build system* berbasis *Gradle* yang fleksibel.
2. *Emulator* yang cepat dan kaya fitur.
3. Lingkungan terpadu yang dapat digunakan untuk mengembangkan aplikasi untuk semua perangkat *Android*.
4. Alat pengujian dan *framework* yang ekstensif.
5. *Instant Run* untuk menggabungkan perubahan pada aplikasi yang sedang berjalan tanpa membangun *Application Package File (APK)* baru.
6. *Lint* untuk menangkap kinerja, kegunaan, kompatibilitas versi dan masalah lainnya.
7. *Template* kode dan integrasi *GitHub* untuk membantu membangun fitur aplikasi umum dan *import* contoh kode.
8. Mendukung *C++* dan *Native Development Kit (NDK).*
9. *Built-in support* untuk *Google Cloud Platform.*

3.8. MEmu App Player

MEmu App Player adalah sebuah mesin virtual (*emulator*) *Android* yang kuat dan bertujuan untuk memberikan pengalaman terbaik bermain game dan aplikasi android di PC (Memuplay, 2017).

Gambar 3.3. Ikon MEmu App Player

Dengan *MEmu App Player* pengguna dapat menguji aplikasi *Android* pada berbagai perangkat *virtual* untuk keperluan pengembangan, pengujian, dan demonstrasi. *Memu* bekerja dengan cepat, mudah dalam instalasi, *sensor widget* dan fitur interaksi yang *user-friendly*. *Emulator* ini dapat dijalankan pada sistem operasi *Windows* 7, *Windows* 8.1 dan *Windows* 10.

3.9. Pemodelan *Unified Modeling Language* (*UML*)

3.9.1 Pengertian *UML*

UML adalah bahasa spesifikasi standar yang dipergunakan untuk mendokumentasikan, menspesifikasikan dan membangun perangkat lunak. *UML* merupakan metodologi dalam mengembangkan sistem beroriantasi objek dan juga merupakan alat untuk mendukung pengembangan sistem (Urva dan Siregar, 2015).

UML saat ini sangat banyak dipergunakan dalam dunia industri yang merupakan standar bahasa pemodelan umum dalam industri perangkat lunak pengembangan sistem. Alat bantu yang digunakan dalam

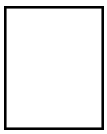
perancangan berorientasi objek berbasiskan *UML* adalah *use case diagram, activity diagram, class diagram,* dan *sequence diagram*

3.9.2 Use Case Diagram

Use case diagram merupakan pemodelan untuk kelakuan (*behavior*) sistem informasi yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Dapat dikatakan *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi tersebut. Simbol-simbol yang digunakan dalam *use case diagram* yaitu seperti pada tabel 3.1.

Tabel 3.1 Simbol Use Case Diagram

No	Simbol	Nama	Keterangan
1		*Actor*	Menspesifikasikan himpunan peran yang pengguna mainkan ketika berinteraksi dengan *use case*.
2		*Dependency*	Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri *(independent)* akan mempengaruhi elemen yang bergantung padanya elemen yang tidak mandiri (*independent*).

No	Simbol	Nama	Keterangan
3		*Generalization*	Hubungan dimana objek anak (*descendent*) berbagi perilaku dan struktur data dari objek yang ada di atasnya objek induk (*ancestor*)
4		*Include*	Menspesifikasikan bahwa *use case* sumber secara *eksplisit.*
5		*Extend*	Menspesifikasikan bahwa *use case* target memperluas perilaku dari *use case* sumber pada suatu titik yang diberikan.
6		*Association*	Apa yang menghubungkan antara objek satu dengan objek lainnya.
7		*System*	Menspesifikasikan paket yang menampilkan sistem secara terbatas.
8		*Use case*	Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu aktor.

No	Simbol	Nama	Keterangan
9		*Collabotation*	Interaksi aturan-aturan dan elemen lain yang bekerja sama untuk menyediakan perilaku yang lebih besar dari jumlah dan elemen-elemennya (sinergi).
10		*Note*	Elemen fisik yang eksis saat aplikasi dijalankan dan mencerminkan suatu sumber daya komputasi.

3.9.3 Activitty Diagram

Activity Diagram menggunakan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis. Simbol-simbol yang digunakan dalam *activity diagram* yaitu seperti dalam tabel 3.2.

Tabel 3.2 Simbol Activity Diagram

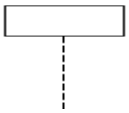
No	Simbol	Nama	Keterangan
1.		*Activity*	Memperlihatkan bagaimana masing-masing kelas antarmuka saling berinteraksi satu sama lain.

No	Simbol	Nama	Keterangan
2.		*Action*	State dari sistem yang mencerminkan eksekusi dari suatu aksi.
3.		*Initial Node*	Bagaimana objek dibentuk atau diawali.
4.		*Activity Final Node*	Bagaimana objek dibentuk dan dihancurkan.
5.		*Fork Node*	Satu aliran yang pada tahap tertentu berubah menjadi beberapa aliran.

3.9.4 Squence Diagram

Squence diagram menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan pesan yang dikirimkan dan diterima antar objek. Simbol-simbol yang digunakan dalam *squence diagram* yaitu seperti dalam tabel 3.3.

Tabel 3.3 Simbol Squence Diagram

No	Simbol	Nama	Keterangan
1		*LifeLine*	Objek *entity*, antarmuka yang saling berinteraksi.

No	Simbol	Nama	Keterangan
2		*Message*	Spesifikasi dari komunikasi antar objek yang memuat informasi-informasi tentang aktivitas yang terjadi
3		*Message*	Spesifikasi dari komunikasi antar objek yang memuat informasi-informasi tentang aktivitas yang terjadi

3.10. *Black Box Testing*

Black Box Testing adalah tipe *testing* yang memperlakukan perangkat lunak yang tidak diketahui kinerja internalnya. Pengujian *black box* dimaksudkan untuk mengethaui apakah fungsi-fungsi, *input* dan *output* dari perangkat lunak sesuai dengan spesifikasi yang dibutuhkan. Para *tester* memandang perangkat lunak seperti layaknya sebuah kotak hitam yang tidak penting dilihat isinya, tapi cukup dikenai proses *testing* dibagian luar. Jenis *testing* ini memandang perangkat lunak dari sisi spesifikasi dan kebutuhan yang telah didefinisikan pada saat awal perancangan (Wijaya dan Sari, 2015).

Metode ujicoba *black box* memfokuskan pada keperluan fungsional dari *software*. Karena itu ujicoba *black box* memungkinkan pengembang *software* untuk membuat himpunan kondisi *input* yang akan melatih seluruh

syarat-syarat fungsional suatu program. Ujicoba *black box* bukan merupakan alternatif dari ujicoba *white box*, tetapi merupakan pendekatan yang melengkapi untuk menemukan kesalahan lainnya, selain menggunakan metode *whitebox*. Ujicoba *black box* berusaha untuk menemukan kesalahan dalam beberapa kategori, diantaranya :

1. Fungsi yang tidak benar atau tidak ada.
2. Kesalahan antarmuka (*interface errors*).
3. Kesalahan pada struktur data dan akses basis data.
4. Kesalahan performa (*performance errors*).
5. Kesalahan inisialisasi dan terminasi.

3.11. *White Box Testing*

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 Test*ing antara lain:

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.*