BAB 2
TINJAUAN PUSTAKA

2.1.Tempat Penelitian

Kendaraan Pribadi dengan jenis motor Yamaha V-ixion dengan tahun perakitan 2013, bertransmisi manual dan Fuel Injection.

2.2.Landasan Teori

Landasan teori berisi penjelasan berbagai konsep serta teori-teori yang berkaitan dalam melakukan penelitian, adapun beberapa teori-teori yang digunakan sebagai pendukung jalannya penelitian sebagai berikut.

2.2.1. Sistem Keamanan

Sistem keamanan merupakan sebuah upaya yang dilakukan untuk mengamankan kinerja dan sumber daya. Penerapan sistem keamanan dalam kehidupan sehari-hari berguna sebagai penjaga sumber daya agar tidak digunakan, modifikasi, interupsi, dan diganggu oleh orang yang tidak berwenang[6].

2.2.2. Sepeda Motor

Sepeda motor adalah kendaraan beroda dua yang digerakkan oleh sebuah mesin. Letak kedua roda sebaris lurus dan pada kecepatan tinggi sepeda motor tetap stabil disebabkan oleh gaya giroskopik. Sedangkan pada kecepatan rendah, kestabilan atau keseimbangan sepeda motor bergantung kepada pengaturan setang oleh pengendara. Penggunaan sepeda motor di Indonesia sangat populer karena harganya yang relatif murah, terjangkau untuk sebagian besar kalangan dan penggunaan bahan bakarnya serta serta biaya operasionalnya cukup hemat [7].

2.2.2.1.Kunci Kontak AC

Pada posisi *OFF* dan *LOCK*, kunci kontak membelokkan tegangan dari sumber tegangan (*alternator*) yang dibutuhkan oleh sistem pengapian ke massa melalui terminal IG dan E kunci kontak, sehingga sistem pengapian tidak dapat bekerja. Di sisi lain pada posisi *OFF* dan *LOCK* kunci kontak juga memutuskan hubungan tegangan (+) baterai (terminal BAT dan BAT 1) sehingga seluruh sistem

kelistrikan tidak dapat dioperasikan. Pada posisi *ON*, kunci kontak memutuskan hubungan terminal IG dan E, sehingga tegangan yang dihasilkan oleh *alternator* diteruskan ke sistem pengapian. Sistem pengapian dapat dioperasikan, disamping itu hubungan terminal BAT dan BAT 1 terhubung sehingga seluruh sistem kelistrikan dapat dioperasikan[8].

2.2.2.2.Kunci Kontak DC

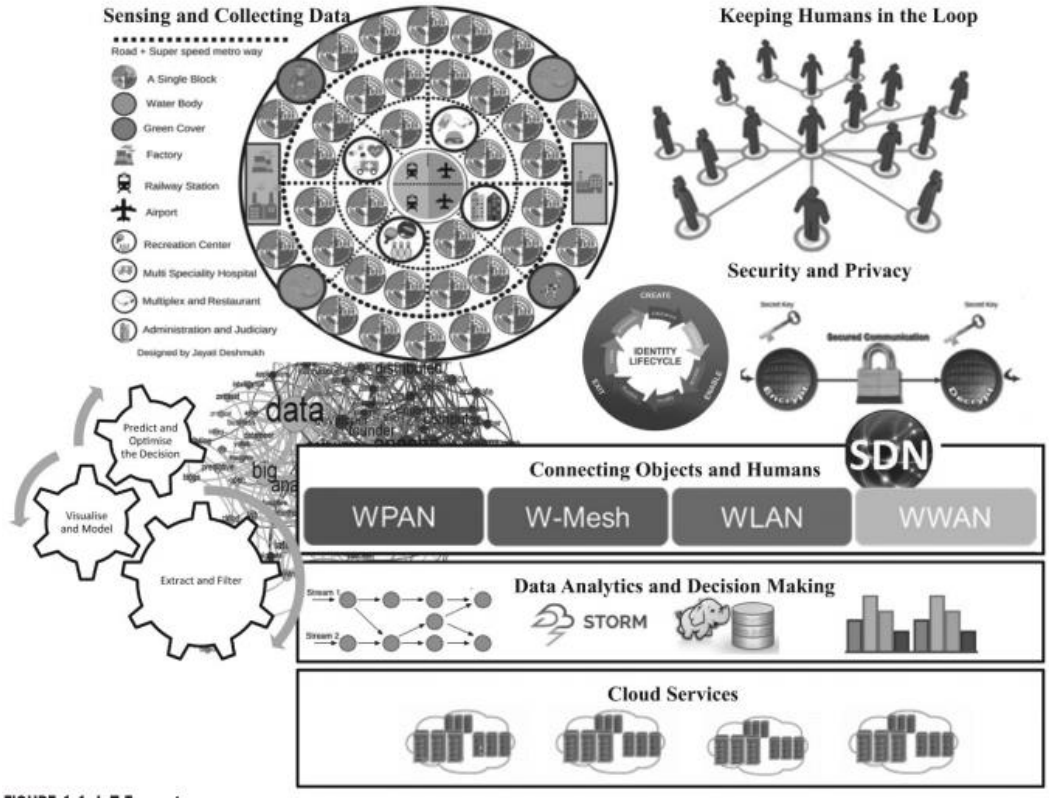
Pada posisi ON, kunci kontak menghubungkan tegangan (+) baterai ke seluruh sistem kelistrikan (termasuk sistem pengapian) untuk mengoperasikan seluruh sistem kelistrikan yang ada. Pada posisi OFF dan LOCK, kunci kontak memutuskan hubungan kelistrikan dari sumber tegangan (terminal (+) baterai) yang dibutuhkan oleh seluruh sistem kelistrikan, sehingga seluruh sistem kelistrikan tidak dapat dioperasikan[8]. Pada penelitian ini digunakan sistem kunci kontak agar pemutusan atau penyambungan ignition coil (kunci kontak) lebih mudah dilakukan.

2.2.3. Internet Of Things (IOT)

Setelah empat dekade sejak munculnya Internet oleh ARPANET, istilah internet mengacu pada kategori besar aplikasi dan protokol yang dibangun di atas jaringan komputer yang canggih dan saling berhubungan, melayani miliaran pengguna di seluruh dunia dalam 24/7. Fokusnya telah bergeser ke arah integrasi manusia dan perangkat tanpa adanya batasan untuk meliput ranah fisik dengan lingkungan manusia, dan terciptalah *Internet Of Things* (IOT).

Internet Of Things (IOT) terdiri dari 2 pilar utama yaitu “Internet” dan “*Things*”, jadi setiap obyek yang mampu terhubung ke internet akan masuk ke dalam kategori “*Things*” seperti mencakup seperangkat entitas yang lebih umum seperti *smartphone*, *sensors*, manusia dan objek lainnya. Konteksnya mampu berkomunikasi dengan entitas lain, membuatnya dapat diakses kapan saja, dimana saja. Secara garis besar dengan *Internet Of Things* (IOT) objek harus dapat diakses tanpa batasan waktu atau tempat.

Pada Gambar menunjukan *ecosystem Internet Of Things* (IOT).



Gambar 2.1 Internet Of Things

Internet Of Things (IOT) juga diidentifikasi sebagai enabler untuk interaksi mesin-ke-mesin, manusia-ke-mesin, dan manusia dengan lingkungan. Dengan meningkatnya jumlah smart device dan adopsi protocol baru seperti IPv6, *Internet Of Things* (IOT) diperkirakan akan bergeser kearah perpaduan *smart networking* dan *autonomous networks* dari objek-objek yang dilengkapi internet. *Internet Of Things* (IOT) menawarkan banyak keuntungan untuk berbagai aplikasi, termasuk *emergency management*, *smart farming*, perawatan kesehatan dan lain-lain. Peran penting lain dari *Internet Of Things* (IOT) adalah membangun sistem kolaboratif yang mampu merespons secara efektif terhadap peristiwa yang ditangkap melalui sensor dengan efektif[9]. Pada penelitian ini konsep *Internet Of Things* (IOT) digunakan sebagai konsep utama mikrokontroller dengan sepeda motor dengan menggunakan mikrokontroller *Arduino Uno /Atmega328,* sensor yang digunakan adalah sensor *Fingerprint* untuk mendapatkan informasi sidik jari pengguna, sensor

RFID untuk membaca *UID* dari *e-KTP* pengguna dan modul relay untuk melakukan pengontrolan kunci kontak dan *electical starter*.

2.2.3.1.Internet Of Things (IOT) Architectures

Block arsitektur *Internet Of Things* (IOT) adalah perangkat sensorik, *remote service invocation* dan komunikasi jaringan. Arsitektur sistem holistic untuk *Internet Of Things* (IOT) perlu menjamin secara sempurna pengoperasian komponennya dan menyatukan *hardware* dan *software* secara bersamaan, untuk mencapai hal ini, pertimbangan yang cermat diperlukan dalam merancang *failure recovery* dan *scalability[9]*. Model arsitektur *Internet Of Things* (IOT) sebagai berikut:

1. ***SOA-Based Architecture***

   Service-Oriented Architecture (SOA) mungkin penting untuk penyedia layanan dan pengguna, memastikan *interoperability* di antara perangkat heterogen. Terdapat 4 lapisan fungsi yang berbeda sebagai berikut:

   a. *Sensing Layer*, integerasi dengan objek hardware yang tersedia untuk mendapatkan status dari lingkungan.
   b. *Network Layer*, infrastruktur untuk mendukung koneksi *wireless* atau koneksi kabel.
   c. *Service Layer*, untuk membuat dan mengelola layanan yang diperlukan oleh pengguna atau applikasi.
   d. *Interfaces Layer*, terdiri dari metode interaksi dengan pengguna atau applikasi.

   Secara umum, dalam arsitektur SOA sistem yang kompleks dibagi menjadi beberapa subsistem yang modular sehingga memberikan cara mudah untuk mempertahankan keseluruhan sistem dengan merawat komponen individualnya, dengan ini dapat memastikan bahwa jika terjadi kegagalan komponen, sisa sistem masi dapat beroperasi secara normal, ini adalah nilai yang sangat besar untuk desain yang efektik dari arsitektur aplikasi *Internet Of Things (IOT)* dimana *reability* adalah parameter yang paling signifikan. SOA telah digunakan secara intesif di WSN, karena tingkat abstraksi yang sesuai dan keunggulan yang berkaitan dengan desain modularnya[9].

2. ***API-Oriented Architecture***

Pendekatan konvensional untuk mengembangkan solusi berorientasi layanan *Remote Method Invocation* (RMI) sebagai sarana untuk mengGambarkan, menemukan dan memanggil layanan, karena *overhead* dan *complexity* yang dipaksakan oleh tenik ini, *API* web dan *Representational State Transfer* (REST) sebagai metode yang memiliki solusi alternative yang menjanjikan. Sumber daya yang dibutuhkan berkisar dari bandwidth jarigan ke kapasitas komputasi dan penyimanan data, dipicu oleh permintaan konversi data permintaan *respons* yang terjadi secara teratur selama *service calls*. Format pertukaran data ringan seperti *JavaScript Object Notation* (JSON) dapat mengurangi overhead, terutama untuk smart devices dan sensor dengan sumber daya terbatas, dengan mengganti file XML berukuran besar, ini membantu dalam menggunakan *communication channel* dan pemprosesan di perangkat lebih efisien. Membangun API untuk applikasi Internet Of Things (IOT) membantu penyedia layanan menarik lebih banyak pelanggan sambal berfokus pada fungsionaltias produk dari pada presentasi. Selain itu, lebih mudah untuk mengaktifkan *multitenancy* dengan fitur keamanan API sepeti *OAuth*, API yang memang mampu meningkatkan eksposisi dan komersialisasi layanan. Dengan API menyediakan pemantauan layanan dan alat bisa lebih efisien dari pada berorientasi dengan layanan sebelumnya[9].

2.2.3.2.Monitoring and Actuating

Pemantauan perangkat melalui API dapat membantu menjadi lebih mudah. API dapat melaporkan penggunaa daya, kinerja peralatan dan status sensor sehingga alat dapat melakukan tindakan setelah mengirim perintah yang telah di tentukan. *Real-time application* dapat memanfaatkan fitur API untuk melaporkan status sistem saat ini, seadngkan manajer dan pengembang memiliki opsi untuk secara bebas memanggil API tanpa perlu mengakses perangkat secara fisik atau *remote devices*, dapat membatnu mengidentifikasi cata produski atau kinerja melalui penerapan deteksi *anomaly* pada data yang dikumpulkan dengan demikian dapat meingkatkanya produktivitas[9].

2.2.4. Internet

Internet adalah sekumpulan jaringan berbeda yang saling terhubung Bersama sebagai suatu kesatuan dengan menggunakan berbagai macam protokol, salahsatunya adalah protokol TCP/IP (*Transmission Control Protocol / InternetProtocol*). TCP/IP adalah protokol yang paling banyak digunakan di internet. Protokol TCP/IP merupakan cara standarduntuk memaketkan dan mengalamatkan data komputer (sinyal eletronik) sehingga data tersebut dapat dikirim ke komputer terdekat atau keliling dunia dan tiba dalam waktu cepat tanpa rusak atau hilang [16].

2.2.5. *Object Oriented* (OO)

Object oriented adalah sebuah obyek memiliki keadaan sesaat (*state*) dan perilaku (*behaviour*). State sebuah obyek adalah kondisi obyek tersebut yang dinyatakan dalam *attribute*/*properties*. State sebuah obyek adalah kondisi obyek tersebut yang dinyatakan dalam *attribute*/*properties*. Sedangkan perilaku suatu obyek mendefinisikan bagaimana sebuah obyek bertindak/beraksi dan memberikan reaksi. Perilaku sebuah objek dinyatakan dalam operation. Menurut schmuller, *attribute* dan *operation* bila disatukan akan memberikan *fitur*/*features*. Himpunan objek-objek yang sejenis disebut class. Objek adalah contoh *instance* dari sebuah class. Ada dua macam aplikasi yang tidak cocok dikembangkan dengan metode OO. Yang pertama adalah aplikasi yang sangat berorientasi ke database. Aplikasi yang sangat berorientasi ke penyimpanan dan pemanggilan data sagnat tidak cocok dikembangkan dengan OO karena akan banyak kehilangan manfaat dari penggunaan RDBMS (Relational Database Management System) untuk penyimpanan data. Akan tetapi RDBMS juga mempunyai keterbatasan dalam penyimpanan dan pemanggilan struktur data yang kompleks seperti multimedia, data spasial dan lain-lain. Bentuk aplikasi lain yang kurang cocok dengan pendekatan OO adalah aplikasi yang membutuhkan banyak algoritma. Beberapa applikasi yang melibatkan perhitungan yang besar dan kompleks[10]. Pada penelitian ini konsep OO digunakan untuk konsep pengkodean pada sistem yang akan dibangun. Berikut ini adalah konsep-konsep dalam pemrograman berorientasi objek :

1. ***Class***

Kelas (*Class*) merupakan pengGambaran satu set objek yang memiliki atribut yang sama. Kelas mirip dengan tipe data ada pemrograman non objek, akan tetapi lebih komprehensif karena terdapat struktur sekaligus karakteristiknya. Kelas baru dapat dibentuk lebih spesifik dari kelas ada umumnya.kelas merupakan jantung dalam pemrograman berorientasi objek.

2. ***Object***

   Objek merupakan teknik dalam menyelesaikan masalah yang kerap muncul dalam pengembangan perangkat lunak. Teknik ini merupakan teknik yang efektif dalam menemukan cara yang tepat dalam membangun sistem dan menjadi metode yang paling banyak dipakai oleh para pengembang perangkat lunak. Orientasi objek merupakan teknik pemodelan sistem riil yang berbasis objek.

3. ***Abstraction***

   Kemampuan sebuah program untuk melewati aspek informasi yang diolah adalah kemampuan untuk fokus pada inti permasalahan. Setiap objek dalam sistem melayani berbagai model dari pelaku abstrak yang dapat melakukan kerja, laporan dan perubahan serta berkomunikasi dengan objek lain dalam sistem, tanpa harus menampakkan kelebihan diterapkan.

4. ***Encapsulation***

   Encapsulation adalah proses memastikan pengguna sebuah objek tidak dapat menggantikan keadaan dari sebuah objek dengan cara yang tidak sesuai prosedur. Artinya, hanya metode yang terdapat dalam objek tersebut yang diberi izin untuk mengakses keadaan yang diinginkan. Setiap objek mengakses *interface* yang menyabutkan bagaimana objek lainnya dapat berintegrasi dengannya. Objek lainnya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut

5. ***Polimorfism***

   Polimorfism merupakan suatu fungsionalitas yang diimplikasikan dengan berbagai cara yang berbeda. Pada program berorientasi objek, pembuat

program dapat memiliki berbagai implementasi untuk sebagian fungsi tertentu.

6. ***Inheritance***

   Seperti yang sudah diuraikan di atas, objek adalah contoh / *instance* dari sebuah *class*. Hal ini mempunyai konsekuensi yang penting yaitu sebagai *instance* sebuah *class*, sebuah objek mempuyai semua karakteristik dari classnya. Inilah yang disebut dengan *Inheritance* (pewarisan sifat). Dengan demikian apapun *attribute* dan *operation* dari *class* akan dimiliki pula oleh semua obyek yang disinherit/diturunkan dari class tersebut. Sifat ini tidak hanya berlaku untuk objek terhadap *class*, akan tetapi juga berlaku untuk *class* terhadap *class* lainnya.

2.2.6. *Unified Modeling Language* (UML)

Unified Modeling Language (UML) adalah salah satu alat bantu yang sangat handal di dunia pengembangan sistem yang berorientasi objek. Hal ini disebabkan karena UML menyediakan bahasa permodelan visual yang memungkinkan bagi pengembang sistem untuk membuat cetak biru atau svisi mekanisme yang efektif untuk berbagi dan mengkomunikasikan rancagan mereka dengan yang lain. *Unified Modeling Language* (UML) merupakan kesatuan dari Bahasa permodelan yang dikembangkan booch, *Object Modeling Technique* (OMT) dan *Object Orented Software Engineering* (OOSE) Metode ini menjadikan proses analisis dan design kedalam empat tahapan iterative, yaitu identifikasi kelas-kelas, dan objek-objek, identifikasi semantic dari hubungan objek dan kelas tersebut, perincian *interface* dan implementasi. Keunggulan metode Booch adalah pada detail dan kayanya dengan notasi dan elemen, permodelan OMT yang dikembangkan oleh Rumbaugh didasarkan pada analisis terstruktur dan pemodelan entity-relationship. Tahapan utama dalam metodologi ini adalah nalisis, design sistem, design objek dan implementasi. Keunggulan metode ini adalah dalam penotasian yang mendukung konsep OO.

Metode OOSE dari Jacobson lebih memberi menekanan pada use case. OOSE memiliki tiga tahapan yaitu membuat model requirement dan analisis, design dan implementasi, dan model pengujian. Keunggulan metode ini adalah mudah dipelajari karena memiliki notasi yang sederhana namun mencakup seluruh tahapan

dalam rekayasa perangkat lunak. Dengan UML, metode Booch, OMT dan OOSE digabungkan dengan membuang elemen-elemen yang tidak praktis ditambah dengan elemen-elemen dari metode lain yang lebih efektif dan elemen-elemen baru yang belum ada pada metode terdahulu sehingga UML lebih ekspresif dan seragam daripada metode lainnya.

Sebagai sebuah notasi grafis yang relative sudah dibakukan (*open standard*) dan dikontrol oleh OMG (*Object Management Group*) mungkin lebih dikenal sebagai badan yang berhasil membakukan CORBA (Common Object Request Broker Architecture), UML menawarkan banyak keistimewaan. UML tidak hanya dominan dalam penotasian di lingkunan OO tetapi juga popular di luar lingkungan OO. Paling tidak ada tiga karakter penting yang melekat di UML yaitu sketsa, cetak biru dan Bahasa pemograman. Sebagai sebuah sketsa, UML bisa berfungsi sebagai jembatan dalam mengkomunikasikan beberapa aspek dari sistem. Dengan demikian semua anggota tim akan mempunyai Gambaran yang sama tentang suatu sistem. UML bisa juga berfungsi sebagai sebuah cetak biru karena sangat lengkap dan detil. Dengan cetak biru ini maka akan bisa diketahui informasi detail tentang coding program (*forward engineering*) atau bahkan membaca program dan menginterpretasikannya kembali kedalam diagram (*reverse engineering*). *Reverse engineering* sangat berguna pada situasi dimana code program yang tidak terdokumentasi asli hilang atau bahkan elum dibuat sama sekali. Sebagai bahasa pemograman, UML dapat menterjemahkan diagram yang ada di UML menjadi code program yang siap untuk di jalankan[10].

Struktur sebuah sistem dideskripsikan dalam 5 view diman asalah satu diantaranya scenario, scenario ini memegang peran khusus untuk mengintegrasikan content ke view yang lain. Kelima view tersebut berhubungan dengan diagram yang dideskripsikan di UML. Setiap view berhubungan dengan perspektif tertentu di mana sistem akan diuji. View yang erbeda akan menekankan pada aspek yang berbeda dari siste yang mewakili ketertarikan sekelompok stakeholder tertentu. Penjelasan lengkap tentang sistem bisa dibentuk dengan menggambungkan informasi-informasi yang ada pada kelima view tersebut sebagai berikut :

1. *Scenario*, mengGambarkan interaksi diantara objek dan diantara proses. *Scenario* ini digunakan untuk dientifikasi elemen arsitektur, ilustrasi dan validasi disain arsitektur serta sebagai titik awal untuk pengujian prototype arsitektur. *Scenario* ini bisa juga disebut dengan *use case view*. *Use case view* ini mendefinisikan kebutuhan sistem karena mengantdung semua *view* yang lain yang mendeskripsikan aspek-aspek tertentu dari rancangan sistem. Itulah sebabnya *use case view* menajdi pusat peran dan sering dikatakan yang mengdrive proses pengembangan perangkat lunak.
2. *Development View*, menjelaskan sebuah sistem dari perspektif programmer dan terkonsentrasikan ke manajemen perangkat lunak. *View* ini dikenal juga sebagai *implementation view*. Diagram UML yang termasuk dalam development *view* di antaranya adalah *component diagram* dan *package diagram*.
3. *Logical View*, terkait dengan fungsionalitas sistem yang dipersiapkan untuk pengguna akhir. *Logical view* mendeskripsikan struktur logika yang mendukung fungsi-fungsi yang dibutuhkan di *use case*. *Design view* ini berisi *object diagram, class diagram, state machine diagram* dan *composite structure diagram.*
4. *Physical View*, mengGambarkan sistem dari perspektif *sistem engineer*. Fokus dari *physical view* adalah *topologi* sistem perangkat lunak. *View* ini dikenal juga sebagai deployment view. Yang termasuk dalam *physical* view ini adalah *deployment diagram* dan *timing diagram*.
5. *Process View*, berhubungan erat dengan aspek dinamis dari sistem, proses yang terjadi di sistem dan bagaimana komunikasi yang terjadi di sistem serta tingkah laku sistem saat dijalankan. *Process view* menjelaskan apa itu *concurrency, distribusi integrasi, kinerja* dan lain-lain. Yang termasuk dalam process *view* adalah *activiy diagram, communication diagram, sequence diagram* dan *interaction overview diagram.*

Mesikpun UML sudah cukup banyak menyediakan diagram yang bisa membantu mendefinisikan sebuah aplikasi, tidak berarti bahwa semua diagram tersebut akan bisa menjawab persoalan yang ada. Dalam banyak kasus, diagram lain selain UML sangat banyak membantu. Pada penelitian ini konsep UML

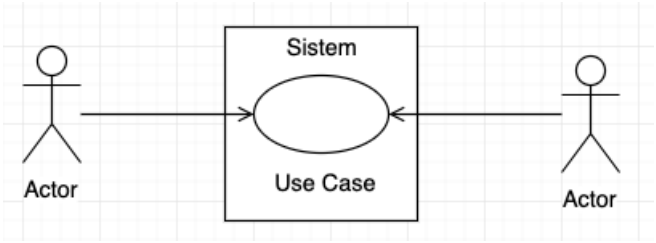
digunakan untuk mengGambarkan bagaimana sistem bekerja, hubungan antar class, memberi Gambaran kepada user sistem yang akan di gunakan dan memberikan penjelasan detail pemanggilan fungsi-fungsi atau method dalam class [10].

1.2.5.1.Diagram UML

UML menyediakan 4 macam diagram untuk memodelkan aplikasi perangkat lunak berorientasi objek. Yaitu:

1. ***Use Case Diagram***

   Use case diagram adalah deskripsi fungsi dari sebuah sistem dari perspektif pengguna. *Use case* bekerja dengan cara mendeskripsikan tipikal interaksi antara user (pengguna) sebuah sistem dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah sistem dipakai. Urutan langkah-langkah yang menerangkan antara pengguna dan sistem disebut *scenario*. Setiap *scenario* mendeskripsikan urutan kejadian. Setiap urutan diinisialisasi oleh orang, system yang lain, perangkat keras atau urutan waktu. Dengan demikian secara singkat bisa dikatan *use case* adalah serangkaian *scenario* yang digabungkan Bersama-sama oleh tujuan umum pengguna. Diagram use case menunjukan 3 aspek dari sistem yaitu : *actor*, *use case* dan *sistem* atau *sub sistem boundary*. *Actor* mewakili peran orang, sistem yang lain atau alat ketika berkomunikasi dengan use case. Gambar 2.2 menunjukan *Use Case Diagram* dalam UML [10].

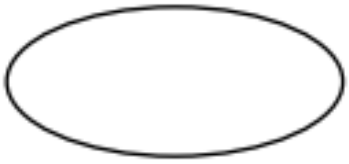


Gambar 2.2 Use Case Diagram

Berikut ini adalah bagian dari sebuah use case diagram :

a. Use Case

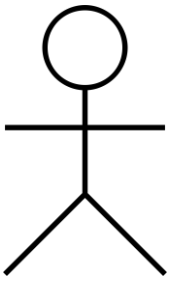
Use case adalah abstraksi dari interaksi antarasistem dengan *actor*. Oleh karena itu sangat penting untuk memilih abstraksi yang cocok. *Use case* dibuat berdasarkan keperluan *actor*. *Use case* harus merupakan 'apa' yang dikerjakan software aplikai, bukan 'bagaimana' software aplikasi mengerjakannya. Setiap *user case* harus diberi nama yang menyatakan apa hal yang dicapai dari hasil interaksinya dengan *actor*. Nama *use case* boleh terdiri dari beberapa kata dan tidak boleh ada dua *use case* yang memiliki nama yang sama. Gambar 2.3 menunjukan bentuk *Use Case* dalam UML.



Gambar 2.3 Use Case [10]

b. *Actors*

Actors adalah *abstraction* dari orang dan sistem yang lain yang mengaktifkan fungsi dari target sistem. Orang atau sistem bisa muncukl dalam beberapa peran. Bahwa actor berinteraksi dengan *use case*, tetapi tidak memiliki control atas *use case*. Gambar 2.4 menunjukan bentuk *actor* dalam UML.



Gambar 2.4 Actors [10]

c. *Relationship*

Relationship adalah hubungan antara *use cases* dengan *actors[10]*. *Relationship* dalam *use case* diagram meliputi:

a. Asosiasi antara *actor* dan *use case*.

   Hubungan antara *actor* dan *use case* yang terjadi karena adanya interaksi antara kedua belah pihak. Asosiasi tipe ini menggunakan garis lurus dari actor menuju *use case* baik dengan menggunakan mata panah terbuka ataupun tidak.

b. Asosiasi antara 2 *use case*

   Hubungan antara *use case* yang satu dan *use case* lainnya yang terjadi karena adanya interaksi antara kedua belah pihak. Asosiasi tipe ini menggunakan garis putus-putus/garis lurus dengan mata panah terbuka di ujungnya.

c. Generalisasi antara 2 *actor*

   Hubungan *inheritance* (pewarisan) yang melibatkan *actor* yang satu (*the child*) dengan *actor* lainnya (*the parent*). Generalisasi tipe ini menggunakan garis lurus dengan mata panah tertutup di ujungnya.

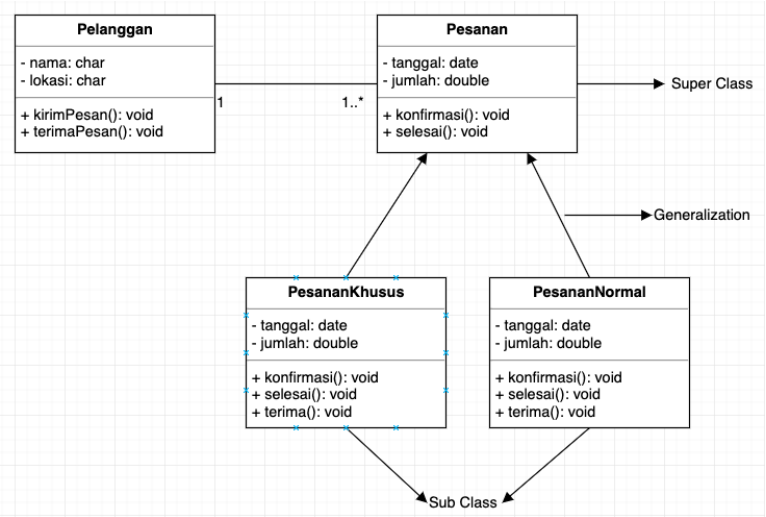
d. Generalisasi antara 2 *use case*.

   Hubungan *inheritance* (pewarisan) yang melibatkan *use case* yang satu (*the child*) dengan use case lainnya(*the parent*). Generalisasi tipe ini menggunakan garis lurus dengan mata panah tertutup di ujungnya.

2. *Class Diagram*

Class diagram adalah diagram statis. Ini mewakili pandangan statis dari suatu aplikasi. Class diagram tidak hanya digunakan untuk memvisualisasikan, mengGambarkan,o dan mendokumentasikan berbagai aspek sistem tetapi juga membangun kode eksekusi dari aplikasi perangkat lunak. *Class diagram* mengGambarkan *atribut*, *operation* dan juga *constraint* yang terjadi pada sistem. *Class diagram* banyak digunakan dalam pemodelan sistem OO karena mereka adalah satu-satunya diagram UML, yang dapat dipetakan langsung dengan bahasa berorientasi objek. Class diagram menunjukan koleksi Class, antarmuka, asosiasi, kolaborasi, dan *constraint*. Dikenal juga sebagai

diagram structural [10]. Gambar 2.5 menunjukan *Class Diagram* dalam UML.



Gambar 2.4 Class Diagram

Class diagram mempunyai 3 relasi dalam penggunaanya, yaitu :

a. *Assosiation*

   Assosiation adalah sebuah hubungan yang menunjukkan adanya interaksi antar *class*. Hubungan ini dapat ditunjukkan dengan garis dengan mata panah terbuka di ujungnya yang mengindikasikan adanya aliran pesan dalam satu arah.

b. *Generalization*

   Generalization adalah sebuah hubungan antar *class* yang bersifat dari khusus ke umum

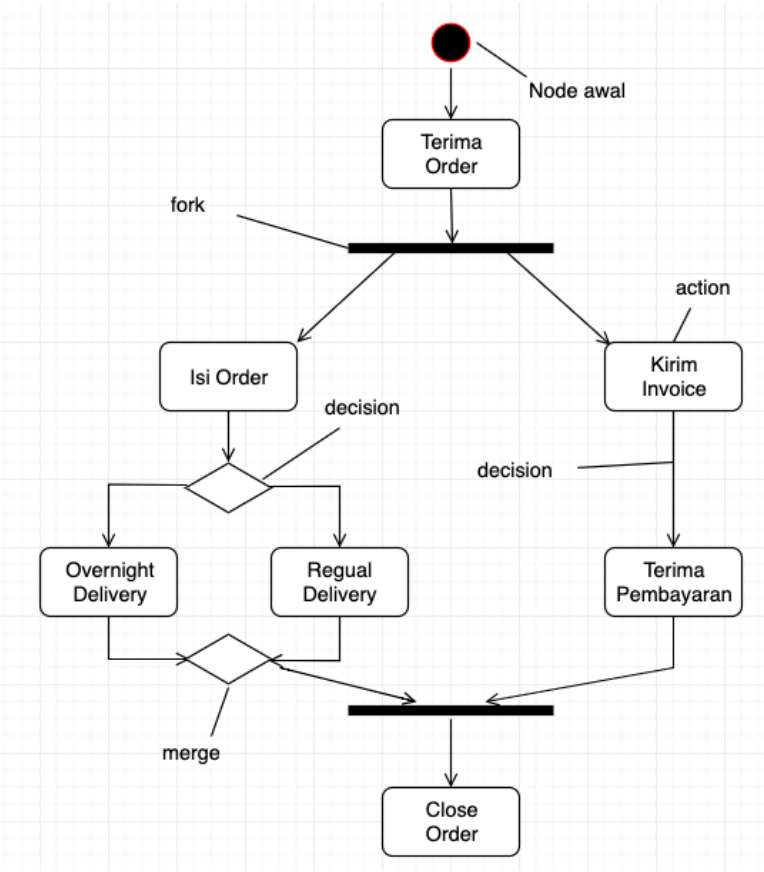
c. *Constraint*

   Constraint adalah sebuah hubungan yang digunakan dalam sistem untuk memberi batasan pada sistem sehingga didapat aspek yang tidak fungsional.

3. *Activity Diagram*

Activity diagram adalah bagian penting dari UML yang mengGambarkan aspek dinamis dari sistem. Logika procedural, proses bisnis dan aliran kerja suatu bisnis bisa dengan mudah dideskripsikan dalam *activity diagram*. *Activity diagram* mempunyai peran seperti

halnya *flowchart*, akan tetapi perbedaanya dengan *flowchart* adalah *activity diagram* bisa mendukug perilaku pararel sedangkan *flowchart* tidak bisa. Tujuan dari activity diagram adalah untuk menangkap tingkah laku dinamis dari sistem dengan cara menunjukan aliran pesan dari satu aktifitas ke aktifitas lainya. Secara umum activity diagram digunakan untuk mengGambarkan diagram alir yang terdiri dari banyak aktifitas dalam sistem dengan beberapa fungsi tambahan seperti percabangan, aliran pararel, swim lane, dan sebagainya. Sebelum mengGambarkan sebuah sctivity diagram, perlu adanya pemahaman yang jelas tentang elemen yang akan digunakan di activity diagram. Elemen tuama dalam activity diagram adalah aktifitas itu sendiri. Aktifitas adalah fungsi yang dilakukan oleh sistem Setelah akitifitas terindentifikasi, selanjutnya yang perlu diketahui adalah bagaimana semua elemen tersebut berasosiasi dengan constraint dan kondisi. Langa selanjutnya perlu penjabaran tata letak dari keseluruhan aliran agara bisa ditransofrmasikan ke activity diagram[10]. Gambar 2.6 menunjukan *Activity Diagram* dalam UML.



Gambar 2.5 Activity Diagram

Berikut ini merupakan komponen dalam activity diagram, yaitu :

a. *Activity node*

   Activity node mengGambarkan bentuk notasi dari beberapa proses yang beroperasi dalam kontrol dan nilai data

b. *Activity edge*

   Activity edge mengGambarkan bentuk *edge* yang menghubungkan aliran aksi secara langsung ,dimana menghubungkan *input* dan *output* dari aksi tersebut

c. *Initial state*

   Bentuk lingkaran berisi penuh melambangkan awal dari suatu proses.

d. *Decision*

Bentuk wajib dengan suatu *flow* yang masuk beserta dua atau lebih *activity node* yang keluar. *Activity node* yang keluar ditandai untuk mengindikasikan beberapa kondisi.

e. *Fork*

Satu bar hitam dengan satu *activity node* yang masuk beserta dua atau lebih *activity node* yang keluar.

f. *Join*

Satu bar hitam dengan dua atau lebih *activity node* yang masuk beserta satu *activity node* yang keluar, tercatat pada akhir dari proses secara bersamaan. Semua *actions* yang menuju *join* harus lengkap sebelum proses dapat berlanjut.

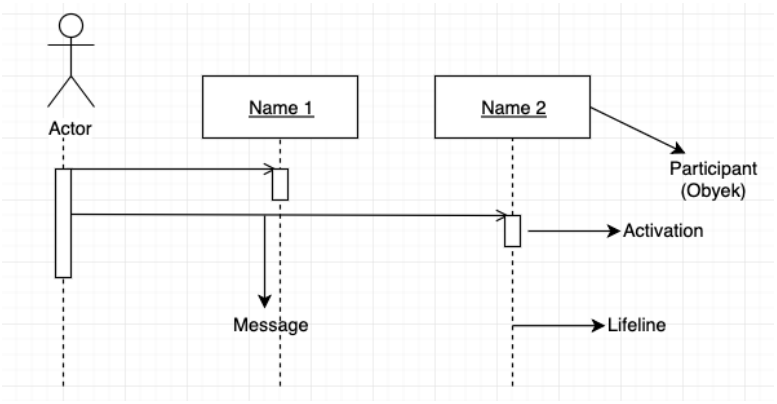
g. *Final state*

Bentuk lingkaran berisi penuh yang berada di dalam lingkaran kosong, menunjukkan akhir dari suatu proses.

4. *Sequence Diagram*

Sequence diagram digunakan untuk mengGambarkan perilaku pada sebuah *scenario*. Diagram ini menunjukan sejumlah contoh obyek dan *message* (pesan) yang diletakan diantara obyek-obyek ini di dalam use case. Komponen utama *sequence diagram* terdiri atas objek yang ditulisan dengan kotak segiempat bernama. *Message* diwakili oleh garis dengan tanda panah dan waktu yang ditunjukan dengan *progress vertical*. Objek diletakan di detak bagian atas diagram dengan urutan dari kiri ke kanan. Mereka diatur dalam urutan guna menyederhanakan diagram, istilah objek dikenal juga dengan *participant*, setiap *participant* terhubung dengan garis titik-titik yang disebut *lifeline*. Sepanjang *lifeline* ada kotak yang disebut *activation*, *activation* mewakili sebuah eksekusi operasi dari *participant*. Panjang kotak ini berbanding lurus dengan durasi *activation*. Sebuah *message* bisa jadi simple, *synchronous* atau *asynchronous*. *Message* yang simple adalah sebuah perpindahan (*transfer*) *control* dari satu *participant* ke *participant* yang lainnya. Jika sebuah *participant* mengirimkan sebuah message *synchronous*, maka jawaban atas *message* tersebut akan

ditunggu sebelum diproses dengan ururssannya. Namun jika message message *asynchronous* yang dikirimkan, maka jawaban atas *message* tersebut tidak perlu ditunggu. *Time* adalah diagram yang mewakili waktu pada arah *vertical*. Waktu dimulai dari atas ke bawah. *Message* yang lebih dekat dari atas akan dijadikan terlebih dahulu dibanding *message* yang lebih dekat ke bawah [10]. Gambar 2.7 menunjukan *Sequence Diagram* dalam UML.



Gambar 2.6 Sequence Diagram

Berikut ini merupakan komponen dalam *sequence diagram* :

a. *Activations*

   Activations menjelaskan tentang eksekusi dari fungsi yang dimiliki oleh suatu objek.

b. *Actor*

   Actor menjelaskan tentang peran yang melakukan serangkaian aksi dalam suatu proses.

c. *Collaboration boundary*

   Collaboration boundary menjelaskan tentang tempat untuk lingkungan percobaan dan digunakan untuk memonitor objek.

d. *Parallel vertical lines*

   Parallel vertical lines menjelaskan tentang suatu garis proses yang menunjuk pada suatu *state*.

e. *Processes*

Processes menjelaskan tentang tindakan/aksi yang dilakukan oleh aktor dalam suatu waktu.

f. *Window*

Window menjelaskan tentang halaman yang sedang ditampilkan dalam suatu proses.

g. *Loop*

Loop menjelaskan tentang model logika yang berpotensi untuk diulang beberapa kali.

2.2.7. *Entity Relationship Diagram* (ERD)

Entity Relationship Diagram (ERD) adalah model teknik pendekatan yang menyatakan atau mengGambarkan hubungan suatu model. Didalam hubungan ini tersebut dinyatakan yang utama dari ERD adalah menunjukan objek data (*Entity*) dan hubungan (*Relationship*), yang ada pada Entity berikutnya. Proses memungkinkan analis menghasilkan struktur basis data dapat disimpan dan diambil secara efisien[11]. Pada penelitian ini konsep ERD digunakan untuk merancang database yang akan di gunakan dalam sistem. Simbol-simbol dalam *Entity Relationship Diagram* (ERD) adalah sebagai berikut:

1. Entitas, suatu yang nyata atau bastrak yang mempunyai karakteristik dimana kita akan menyimpan data.
2. Atribut, ciri umum semua atau sebagian besar instansi pada entitas tertentu.
3. Relasi, hubungan alamiah yang terjadi antara satu atau lebih entitas.
4. Link, garis penghubung atribut dengan kumpulan entitas dan kumpulan entitas dengan relasi.

Hubungan kardinalitas relasi dalam *Entity Relationship Diagram* (ERD) adalah sebagai berikut:

1. Satu ke satu (One to One)

   Setiap elemen dari Entitas A berhubungan paling banyak dengan elemen pada Entitas B. Demikian juga sebaliknya setiap elemen B berhubungan paling banyak satu elemen pada Entitas A.
2. Satu ke banyak (One to Many)

Setiap elemen dari Entitas A berhubungan dengan maksimal banyak elemen pada Entitas B. Dan sebaliknya setiap elemen dari Entitas B berhubungan dengan paling banyak satu elemen di Entitas A.

3. Banyak ke satu (Many to One)

   Setiap elemen dari Entitas A berhubungan paling banyak dengan satu elemen pada Entitas B. Dan sebaliknya setiap elemen dari Entitas B behubungan dengan maksimal banyak elemen di entitas A.

4. Banyak ke banyak (Many to Many)

   Setiap elemen dari Entitas A berhubungan maksimal banyak elemen pada Entitas B demikian sebaliknya.

2.2.8. *Fritzing*

Fritzing merupakan salah satu dari sekian banyak aplikasi untuk mendesain rangkaian elektronika. Sama halnya seperti aplikasi lain (*Eagle*, *Protel*, *Pad2pad*, dll) *Fritzing* juga memiliki *Board Designer* (untuk membuat jalur PCB). Pada penelitian ini fritzing digunakan untuk mendesain komponen *raspberry pi*, sensor-sensor dan modul yang digunakan Gambar 2.11 menunjukan logo *fritzing*.

Gambar 2.7 Logo Fritzing

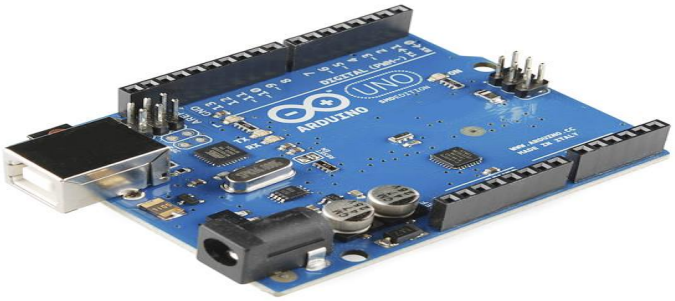
Fritzing sedikit berbeda dengan aplikasi-aplikasi lainnya, kelebihan yang ada pada *fritzing* di antaranya :

1. Open Source, Sudah menyediakan banyak library, bahkan untuk mikrokontroller buatan seperti *Arduino*, *Raspberry Pi*, *basic stamp*, *pic.exe*, dan lain-lain. Serta librarynya juga banyak tersedia di internet dan bisa di download secara gratis.
2. Gambar rangkaian bisa di-*Share* di internet melalui aplikasi ini, jadi Gambar rangkaiannya bersifat *Open Hardware*.

3. Dapat mengGambar rangkaian di *BreadBoard*. fitur seperti ini sangat membantu bagi pemula yang ingin belajar rangkaian elektronika, karena Gambar rangkaiannya sangat mudah dimengerti, bahkan bisa ditiru langsung ke BreadBoard.

2.2.9. Arduino Uno

Arduino adalah pengendali mikro single-board yang bersifat open-source, diturunkan dari wiring platform, dirancang untuk memudahkan penggunaan elektronik dalam berbagai bidang. Perangkat kerasnya memiliki prosesor Atmel AVR dan perangkat lunaknya memiliki bahasa pemrograman sendiri.



Gambar 2.8 Arduino Uno

2.2.10. Sensor

Sensor adalah suatu perangkat yang mendeteksi perubahan energi yang berada di alam seperti energi listrik, energi fisika, energi kimia, energi biologi, energi mekanik dan sebagainya. Di dalam sebuah sensor terdapat transduser yang berfungsi untuk mengubah besaran mekanis, magnetis, panas, sinar, dan kimia menjadi besaran listrik berupa tegangan, resistansi dan arus listrik[12].

2.2.10.2. Relay

Relay merupakan komponen elektronika yang memiliki fungsi bekerja sebagai saklar mekanik yang digerakkan oleh energi listrik. *Relay* menggunakan gaya elektromagnetik untuk memutuskan atau menghubungkan suatu rangkaian elektronika yang satu dengan rangkaian elektronika yang lainnya. *Relay* terdiri dari *coil* dan *contact*. *Coil* adalah gulungan kawat yang mendapat arus listrik, sedangkan

contact adalah sejenis saklar yang pergerakannya tergantung dari ada tidaknya arus listrik di *coil*. *Contact* ada 2 jenis yaitu *normally open* (kondisi awal sebelum diaktifkan *open*), dan *normally closed* (kondisi awal sebelum diaktifkan *closed*)[3]. Pada penelitian ini menggunakan relay 4 CH sebagai penghubung dan pemutus arus listrik pada motor..

2.2.10.3. Sensor Sidik Jari

Serial Modul Fingerprint AS608 merupakan sensor sidik jari optikal, yang dapat mendeteksi sidik jadi dengan verifikasi yang sangat sederhana. Module sensor ini bekerja dengan otak utama berupa chip DSP yang melakukan image rendering, kemudian mengkalkulasi, feature-finding dan terakhir searching pada data yang sudah ada.

Gambar 2.9 Fingerprint Sensor

2.2.10.4. Sensor RFID RC522

RFID RC522 (Radio Frequency Identification) merupakan suatu teknologi yang memanfaatkan frekuensi radio sebagai pengidentifikasian terhadap suatu objek. Merupakan sebuah perangkat yang akan diidentifikasi oleh **RFID** reader yang dapat berupa perangkat pasif maupun aktif yang berisi suatu data atau informasi.

Gambar 2.10 RFID Sensor

2.2.10.5. DC Stepdown Converter

DC Stepdown converter merupakan peralatan yang dapat mengkonversikan energi listrik dari tegangan searah tertentu ke tegangan searah yang diinginkan atau teregulasi dengan nilai voltase yang lebih kecil dari catu daya masukan[3]. Pada penelitian ini *DC Converter* yang digunakan pada penelitian ini yang bertipe *stepdown* atau penurun tegangan sumber dari accu.

2.2.11. Metode Pengujian

Pengujian perangkat lunak merupakan proses eksekusi program atau perangkat lunak dengan tujuan mencari kesalahan atau kelemahan dari program tersebut. Proses tersebut dilakukan dengan mengevaluasi atribut dan kemampuan program. Suatu program yang diuji akan dievaluasi apakah keluaran atau output yang dihasilkan telah sesuai dengan yang diinginkan atau tidak. Ada berbagai macam metode pengujian, teknik black box dan teknik white box merupakan metode pengujian yang telah dikenal dan banyak digunakan oleh pengembang perangkat lunak.

1.2.10.1.Black Box Testing

Black box testing merupakan metode pengujian dengan pendekatan yang mengasumsikan sebuah sistem perangkat lunak atau program sebagai sebuah kotak hitam (*black box*). Pendekatan ini hanya mengevaluasi program dari output atau hasil akhir yang dikeluarkan oleh program tersebut. Struktur program dan kode-kode yang ada di dalamnya tidak termasuk dalam pengujian ini. Keuntungan dari

metode pengujian ini adalah mudah dan sederhana. Namun, pengujian dengan metode ini tidak dapat mendeteksi kekurangefektifan pengkodean dalam suatu program. Metode pengujian *black box* merupakan metode pengujian dengan pendekatan yang mengasumsikan sebuah sistem perangkat lunak atau program sebagai sebuah kotak hitam (*black box*). Pendekatan ini hanya mengevaluasi program dari output atau hasil akhir yang dikeluarkan oleh program tersebut. Struktur program dan kode-kode yang ada di dalamnya tidak termasuk dalam pengujian ini. Keuntungan dari metode pengujian ini adalah mudah dan sederhana. Namun, pengujian dengan metode ini tidak dapat mendeteksi kekurangefektifan pengkodean dalam suatu program. Ciri-ciri *black box testing* adalah sebagai berikut:

1. *Black box testing* berfokus pada kebutuhan fungsional pada *software*, berdasarkan pada spesifikasi kebutuhan dari *software*.
2. Merupakan pendekatan pelengkap dalam mencangkup error dengan kelas yang berbeda dari metode white box testing.
3. Melakukan pengujian tanpa pengetahuan detil struktur internal dari sistem atau komonen yang dites. Juga disebut sebagai *behavioural testing*, *specification-based testing*, *input*/*ouput testing* atau *functional testing*.
4. Terdapat jenis test yang dapat dipilih berdasarkan pada tipe testing yang digunakan.
5. Kategori error yang akan diketahuai melalui *black box testing* seperti fungsi yang hilang atau tidak benar, *error* dari antar-muka, *error* dari struktur data atau akses *eksternal database*, *error* dari kinerja dan *error* dari inisialisasi.

Equivalence class partitioning adalah sebuah metode *black box* terarah yang meningkatkan efisiensi dari pengujian dan meningkatkan *coverage* dari *error* yang potensial. Sebuah *equivalence class* adalah sebuah kumpulan dari nilai variable *input* yang memprduksi *output* yang sama. Selanjutnya *output correctness test* merupakan pengujian yang memakan sumber daya paling besar dari pengujian. Pada kasus yang sering terjadi dimana hanya *output correctness test* yang dilakukan, maka sumber daya pengujian akan digunakan semua. Implementasi dari

kelas-kelas pengujian lain tergantung dari sifat produk *software* dan pengguna selanjutnya dan juga prosedur dan keputusan pengembang. *Output correctness test* mengaplikasikan konsep dari *test case* [24]. Pemilihan *test case* yang baik dapat dicapai dengan efesiensi dari penggunaan *equivalence class partitioning*. Jenis-jenis *test case* sebagai berikut:

1. *Availability test*

   Availability didefinisikan sebagai waktu reaksi yaitu waktu yang dibutuhkan untuk mendapatkan informasi yang diminta atau waktu yang dibutuhkan oleh firmware yang diinstal pada perlengkapan komputer untuk bereaksi. *Availability* adalah yang paling penting dalam aplikasi online sistem informasi yang sering digunakan. Kegagalan *firmware software* untuk memenuhi persyaratan ketersediaan dapat membuat perlengkapan tersebut tidak berguna.
2. *Realiability test*

   Realiability berkaitan dengan fitur yang dapat diterjemahkan sebagai kegiatan yang terjadi sepanjang waktu seperti waktu rata-rata antara kegagalan (misalnya 500 jam), waktu rata-rata untuk *recovery* setelah kegagalan sistem (misalnya 15 menit) atau *average downtime* per bulan (misalnya 30 menit per bulan). Persyaratan realiabilitas memiliki efek selama *regular full-capacity* operasi sistem. Harus diperhatikan bahwa penambahan faktor *software realiability* juga berkaitan dengan perangkat, sistem operasi, dan efek dari sistem komunikasi data.
3. *Stress test*

   Stress test terdiri dari 2 tipe pengujian yaitu *load test* dan *durability test*. Suatu hal yang mungkin untuk melakuakn pengujian-pengujian tersebut setelah penyelesaian sistem software. Durability test dapat dilakukan hanya setelah firmware atau sistem informasi *software* diinstall dan siap untuk diuji. Pada *load test* berkaitan dengan *functional performance system* dibawah beban maksimal operasional, yaitu maksimal transaksi per menit, hits per menit ke tempat internet dan sebagainya. *Load test*, yang biasanya dilakukan untuk beban yang lebih tinggi dari yang diindikasikan spesifikasi persyaratan merupakan hal yang penting untuk

sistem *software* yang rencananya akan dilayani secara simultan oleh sejumlah pengguna. Pada sebagian besar kerja sistem software, beban maksimal mengGambarkan gabungan beberapa tipe transaksi. Selanjutnya *durability test* dilakukan pada kondisi operasi fisik yang ekstrem seperti temperatur yang tinggi, kelembaban, mengendara dengan kecepatan tinggi, sebagai detail persyaratan spesifikasi durabilitas. Jadi, dibutuhkan untuk *real-time firmware* yang diintegrasikan ke dalam sistem seperti sistem senjata, kendaraan transport jarak jauh, *Internet Of Things* (IOT) dan keperluan meterologi. Isu ketahan pada *firmware* terdiri dari respon *firmware* terhadap efek cuara seperti temperatur panas atau dingin yang ekstrem, debu, kegagalan operasi ekstrem karena kegagalan listrik secara tiba-tiba, loncatan arus listrik dan putusnya komunikasi secara tiba-tiba.

4. *Training usuability test*

   Ketika sejumlah besar pengguna terlibat dalam sistem operasi, *training usability requirement* ditambahkan dalam agenda pengujian. Lingkup dari *training usability test* ditentukan oleh sumber yang dibutuhkan untuk melatih pekerja baru untuk memperoleh level perkenalan dengan sistem yang ditentukan atau untuk mencapai tingkat produksi tertentu. Detail dari pengujian ini, sama halnya dengan yang lain, didasarkan pada karakteristik pekerja. Hasil dari pengujian ini harus menginspirasi rencana dari kursus pelatihan dan follow-up serta memperbaiki sistem operasi *software*.

5. *Operational usability test*

   Fokus dari pengujian ini adalah produktifitas operator, yang aspeknya terhadap sistem yang mempengaruhi performance dicapai oleh operator sistem. *Operational usability test* dapat dijalankan secara manual.

Revision class partitioning adalah sebuah metode *black box* lainnya yang merupakan faktor dasar yang menentukan keberhasilan paket suatu *software*, pelayanan jangka panjang, dan keberhasilan penjualan ke sejumlah besar populasi pengguna. Berkaitan dengan hal tersebut terdapat tiga kelas pengujian revisi sebagai berikut:

1. *Reusability test*

   Reusability menentukan bagian mana dari suatu program (modul, integrasi, dbs) yang akan dikembangkan untuk digunakan kembali pada projet pengembangan *software* lainnya, baik yang telah direncanakan maupun yang belum. Bagian ini harus dikembangkan, disusun, dan didokumentasikan menurut prosedur perpustakaan *software* yang digunakan ulang.
2. *software interoperability test*

   software interoperability berkaitan dengan kemamputan *software* dalam memenuhi perlengkapan dan paket *software* lainnya agar memungkinkan untuk mengoperasikannya bersama dalam satu sistem komputer kompleks.
3. *Equipment interoperability test*

   Equipment interoperability berkaitan dengan perlengkapan *firmware* dalam menghadapi untuk perlengkapan lain dan atau paket *software*, dimana persyaratan mencantumkan *specified interfaces*, termasuk dengan *interfacing standard*. Pengujian yang relevan harus menguji implementasi dari *interoperability reqirements* dalam sistem.

Keuntungan dan kekurangan dari *black box testing*, beberapa keuntungan dari *black box testing*, diantaranya sebagai berikut:

1. *Black box* testing memungkinkan kita untuk memiliki sebagian besar tingkat pengujian, yang sebagian besarnya dapat diimplementasikan.
2. Untuk tingkat pengujian yang dapat dilakukan baik dengan *white box testing* maupun *black box testing*, *black box testing* memerlukan lebih sedikit sumber dibandingkan dengan yang dibutuhkan oleh *white box testing* pada pake *software* yang sama.

Sedangkan kekurangan dari *black box testing* adalah sebagai berikut:

1. Adanya kemungkinan untuk terjadinya beberapa kesalahan yang tidak disengaja secara bersama-sama akan menimbulkan respon pada pengujian ini dan mencegah deteksi kesalahan (*error*). Dengan kata lain, *black box test* tidak siap untuk mengidentifikasi kesalahan-kesalahan

yang berlawanan satu sama lain sehingga menghasilkan output yang benar.

2. Tidak adanya kontrol terhadap *line coverage*. Pada kasus dimana *black box test* diharapkan dapat meningkatkan *line converage*, tidak ada cara yang mudah untuk menspesifikasikan parameter-parameter pengujian yang dibutuhkan untuk meningkatkan *coverage*. Akibatnya, *black box test* dapat melakukan bagian penting dari baris kode, yang tidak ditangani oleh set pengujian. Ketidakmungkinan untuk menguji kualitas pembuatan kode dan pendekatannya dengan standar pembuatan kode.