BAB II
TINJAUAN PUSTAKA

Bagian ini memuat kajian dan uraian sistematis tentang informasi hasil penelitian yang pernah dilakukan peneliti lain dalam pustaka dan menghubungkannya dengan masalah penelitian. Tinjauan pustaka membahas tentang metode yang digunakan dalam penelitian aplikasi. Hal tersebut digunakan sebagai dasar membuat struktur Landasan Teori.

2.1. Penelitian Terdahulu

Peneliti telah melakukan studi literatur terhadap 6 jurnal yang berkaitan dengan tema penelitian yang dikerjakan. Tabel 2.1 menunjukkan penelitian terkait metode *DevOps Life Cycle*, SDLC, UML dan *BlackBox Testing*.

Penelitian pertama dengan judul "Implementasi DevOps pada Pengembangan Aplikasi e-Skrining Covid-19" memiliki permasalahan pada fase *build, test* dan *deploy* dengan *Agile Scrum* memerlukan banyak waktu dan menyebabkan rilis tidak sesuai jadwal. Tujuan dari penelitian ini adalah fase-fase tersebut dapat dilakukan secara otomatis. Metode yang digunakan adalah *DevOps Life Cycle* dan mengkombinasikannya dengan *Agile Scrum.* Hasil dari penelitian ini adalah penerapan *DevOps Life Cycle* memudahkan penggabungan kode, dan fase *build, test* dan *deploy* dapat dilakukan secara otomatis dan tidak banyak waktu [4].

Penelitian kedua dengan judul "*Quality-Aware DevOps Research: Where Do We Stand ?*" memiliki permasalahan metodologi pengembangan aplikasi tradisional yang kurang efektif karena cukup memakan waktu dan biaya selain itu adanya kesenjangan antara *developer* dan tim operasional. Tujuan dari penelitian adalah memastikan bahwa metodologi *DevOps Life Cycle* dapat mempercepat dan mengurangi biaya pengembangan aplikasi metodologi tersebut, dapat mengurangi kesenjangan antara *developer* dan tim operasional, meningkatkan desain arsitektur, pemodelan dan *infrastructure-*

as-code, *continuous- integration/continuous-delivery* (CI/CD), pengujian dan verifikasi, dan manajemen *runtime*. Metode yang digunakan adalah melakukan *review* dan survei terhadap jurnal dan makalah yang bertemakan *DevOps Life Cycle* dan dituliskan dalam bahasa inggris dengan rentang tahun 2015-2020 dan melakukan percobaan pada *real project*. Hasil dari penelitian ini [6]:

1. *DevOps Testing*, tahap ini dapat mengurangi waktu dan biaya karena dilakukan CI/CD, dan mengurangi kesenjangan antara *developer* dan tim operasional [6].
2. *Architecture Design*, metode ini sangat cocok untuk aplikasi sekala besar yang memiliki desain arsitektur *microservice* maupun *monolithic* [6].

Penelitian ketiga dengan judul "Penerapan DevOps pada Sistem Tertanam dengan ESP8266 menggunakan Mekanisme *Over The Air*". Permasalahan pada penelitian ini, proses pembaharuan *firmware* pada perangkat *Internet of Things* (IoT) dan sistem tertanam mengalami kesulitan pada proses *deployment*. Tujuan dari penelitian adalah proses *deployment* dapat dilakukan secara otomatis dan sistem mampu mendeteksi kesalahan penulisan kode sumber. Metode yang digunakan adalah *DevOps Life Cycle*. Hasil dari penelitian ini, yaitu proses *deployment* dapat dilakukan secara otomatis dan sistem mampu mendeteksi kesalahan penulisan kode sumber dengan rerata waktu keseluruhan proses adalah 77,21 detik dan waktu *deploy* memiliki rerata 1,41 detik [7].

Penelitian keempat dengan judul "Perancangan Sistem Informasi Pengarsipan Data Penduduk Pada Kantor Camat Bilah Hulu Kabupaten Labuhan Batu Dengan Metode *System Develovment Life Cycle* (SDLC)". Permasalahan pada penelitian ini, pengarsipan data penduduk pada kantor camat kabupaten Labuhan Batu masih manual. Tujuan dari penelitian adalah membangun sistem informasi pengarsipan data penduduk pada kantor camat kabupaten Labuhan Batu. Metode yang digunakan adalah *Research and*

Development (R&D) dengan model *Waterfall*. Hasil dari penelitian ini, penggunaan SDLC dalam pengembangan sistem informasi akan membuat rancangan sistem menjadi lebih terstruktur dan jelas [8].

Penelitian kelima dengan judul "Rekayasa Ulang (*Reengineering*) Sistem Informasi Manajemen Pertanahan Nasional Dengan Pendekatan *Unified Modelling Language* (UML)". Permasalahan pada penelitian ini, Sistem Informasi Manajemen Pertanahan Nasional atau disebut SIMTANAS yang sudah ada tidak relevan lagi dengan kebutuhan karena disamping penyusunan basis data hanya berdasarkan pendaftaran tanah, informasi-informasi strategis bidang tanah, berupa pengaturan apa yang boleh dan tidak boleh dilakukan (*restriction*) serta kewajiban - kewajiban yang melekat pada bidang tanah (*servitude*) atau disebut aspek *responsibility* belum terintegrasi di dalamnya. Tujuan dari penelitian adalah memenuhi kebutuhan merekayasa ulang SIMTANAS untuk mengintegrasikan setiap data dan memenuhi kebutuhan sesuai perkembangan zaman. Metode menggunakan pendekatan *model driven* yaitu *Unified Modelling Language* (UML). Hasil dari penelitian ini, penggunaan UML pada rancangan SIMTANAS dapat mendokumentasikan setiap objek yang ada pada sebuah sistem [9].

Penelitian keenam dengan judul "*Software Testing on The Learning of Islamic Education Media Based on Information Communication Technology Using Blackbox Testing*". Permasalahan belum adanya pengujian perangkat lunak pada media pembelajaran pendidikan agama Islam . Tujuan dari penelitian adalah mengimplementasi *BlackBox Testing* pada media pembelajaran perangkat lunak pendidikan Islam. Metode yang digunakan adalah *Waterfall Model*. Hasil dari penelitian ini menunjukkan bahwa *BlackBox Testing* mampu mengukur kualitas perangkat lunak dan menunjukkan hasil yang cukup baik dengan akurasi sebesar 90% [10].

Tabel 2.1. Penelitian terkait DevOps Life Cycle, SDLC, UML dan BlackBox Testing

Peneliti	Judul	Hasil
Tohirin, Sri Farida Utami, Septian Rheno Widianto, Widhy Al Mauludyansah	Implementasi DevOps pada Pengembangan Aplikasi e-Skrining Covid-19 (2020)	Penerapan *DevOps Life Cycle* memudahkan penggabungan kode, dan fase *build*, *test* dan *deploy* dapat dilakukan secara otomatis dan tidak memakan banyak waktu [4].
Ahmad Alnafessah, Alim Ul Gias, Runan Wang, Lulai Zhu, Giuliano Casale, Antonio Filieri	*Quality-Aware DevOps Research: Where Do We Stand?* (2021)	1. *DevOps Testing*, tahap ini dapat mengurangi waktu dan biaya karena dilakukan CI/CD, dan mengurangi kesenjangan antara *developer* dan tim operasional [6]. 2. *Architecture Design*, metode ini sangat cocok untuk aplikasi sekala besar yang memiliki desain arsitektur *microservice* maupun *monolithic* [6].
Azis Wisnu Widhi Nugraha, Imron Rosyadi, Fahmi Khoerullatif	Penerapan DevOps pada Sistem Tertanam dengan ESP8266 menggunakan Mekanisme *Over The Air* (2021)	Proses *deployment* dapat dilakukan secara otomatis dan sistem mampu mendeteksi kesalahan penulisan kode sumber dengan rerata waktu keseluruhan proses adalah 77,21 detik. Proses *build* dan *test* mendominasi waktu proses dengan rerata sebesar 77,21 detik dan waktu *deploy* memiliki rerata 1,41 detik [7].
Ibnu Rasyid Munthe	Perancangan Sistem Informasi Pengarsipan Data Penduduk Pada Kantor Camat Bilah Hulu Kabupaten Labuhan Batu Dengan Metode *System Develovment Life*	Penggunaan SDLC dalam pengembangan sistem informasi akan membuat rancangan sistem menjadi lebih terstruktur dan jelas [8].

Peneliti	Judul	Hasil
	Cycle (SDLC) (2019)	
Wahyuni	Rekayasa Ulang (*Reengineering*) Sistem Informasi Manajemen Pertanahan Nasional Dengan Pendekatan *Unified Modelling Language* (UML) (2017)	Penggunaan UML pada rancangan SIMTANAS dapat mendokumentasikan setiap objek yang ada pada sebuah sistem [9].
Sutiah, Supriyono	*Software Testing on The Learning of Islamic Education Media Based on Information Communication Technology Using Blackbox Testing* (2020)	Hasil dari penelitian ini menunjukkan bahwa *BlackBox Testing* mampu mengukur kualitas perangkat lunak dan menunjukkan hasil yang cukup baik dengan akurasi sebesar 90% [10].

Dari penelitian terdahulu yang telah dirangkum pada tabel 2.1 diketahui penggunaan SDLC sangat penting dalam pengembangan sebuah sistem dan pemilihan metodologi *DevOps Life Cycle* dapat digunakan untuk melakukan proses *build*, *test* dan *deployment* secara cepat karena menggunakan CI/CD. Penggunaan UML juga akan menstrukturisasi dan mendokumentasikan seluruh objek dari sistem. Perbedaan antara penelitian ini dengan yang lain adalah penelitian ini menggunakan metodologi *DevOps Life Cycle* yang dikombinasikan dengan metodologi *BlackBox Testing* untuk memastikan skenario penggunaan aplikasi yang di lakukan oleh aktor/pengguna telah sesuai dengan proses bisnis yang dirancang.

2.2. Dasar Teori

Berikut ini merupakan dasar teori yang menjadi landasan penelitian ini yang bersumber dari buku dan jurnal terdahulu.

2.2.1 Rancang Bangun

Rancang merupakan prosedur untuk menerjemahkan hasil analisis dari sebuah sistem ke dalam bahasa pemrograman, untuk mendeskrisipkan setiap detail sistem diimplementasikan. Bangun merupakan kegiatan menciptakan sistem baru untuk menggantikan atau merevisi sistem yang telah ada, baik sebagian maupun keseluruhan [3].

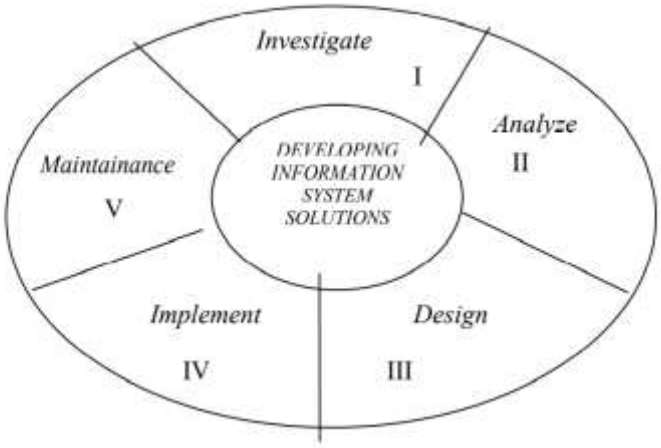
Rancang bangun adalah kegiatan untuk menerjemahkan hasil analisis menjadi sebuah paket perangkat lunak, kemudian perangkat lunak tersebut dapat menjadi sebuah ciptaan baru maupun merevisi sistem yang sudah ada.

2.2.2 *Billing System*

Billing berasal dari bahasa inggris yaitu *bill*, yang artinya adalah bukti pembayaran. Dari arti tersebut dapat diartikan mengirimkan/mengumumkan bukti transaksi.

Billing System merupakan sistem yang digunakan untuk mengontrol dan mencatat seluruh transaksi keuangan, baik pemasukan, pengeluaran hutang, piutang, dan pembukuan yang dapat memudahkan operator keuangan [3].

2.2.3 *Software Development Life Cycle*



Gambar 2.1. Pola Software Development Life Cycle [1]

Software Development Life Cycle (SDLC) merupakan gambaran dari suatu usaha dalam merancang sistem yang akan selalu bergerak seperti roda, yang melewati beberapa langkah atau tahapan yaitu : [11]

1. Investigasi

   Tahap investigasi adalah sebuah proses dasar untuk memahami mengapa sebuah sistem harus dibangun. Pada fase ini diperlukan analisa kelayakan dengan mencari data atau melakukan proses *information gathering* kepada pengguna. Fase ini lebih menekankan pada aspek studi kelayakan pengembangan sistem (*feasibility study*) [11].

2. Analisis

   Tahap analisa adalah sebuah proses memahami sistem yang akan dibangun dengan tujuan untuk mendapatkan jawaban mengenai pengguna sistem, cara kerja sistem dan waktu penggunaan sistem [11].

3. Desain / Perancangan

   Tahap perancangan merupakan proses penentuan cara kerja sistem dalam hal *architechture design*, menganalisa data dan skema *database*, merancang *user interface* dan *program design* [8], [11].

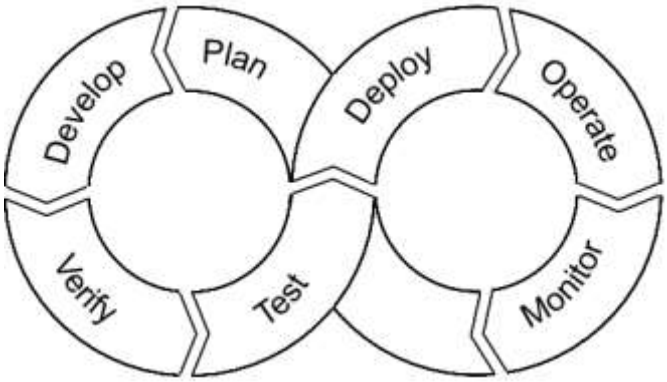
4. Implementasi

Tahap implementasi merupakan proses mengimplementasikan rancangan dari tahap-tahap sebelumnya yang meliputi pembangunan dan pengujian sistem, instalasi sistem, dan rencana dukungan sistem. Dalam pengembangan, dilakukan aktivitas-aktivitas sebagai berikut: pembuatan *database* sesuai skema rancangan, pembuatan aplikasi berdasarkan desain sistem (*Coding*), dan *debugging* yaitu sebuah metode yang dilakukan oleh para pemrogram dan pengembang perangkat lunak untuk mencari dan mengurangi *bug*, atau kerusakan di dalam sebuah program komputer atau perangkat keras sehingga perangkat tersebut bekerja sesuai dengan harapan [11].

5. *Maintenance*

   Tahap ini merupakan dilakukan untuk menjaga sistem tetap mampu beroperasi secara benar seperti pemeliharaan data, pembaruan sistem, serta meningkatkan keamanan data [11].

2.2.4 *DevOps Life Cycle*



Gambar 2.2. Pola DevOps Life Cycle [6]

DevOps Life Cycle diilustrasikan pada Gambar 2.2, *Development and Operations* (*DevOps*) adalah metode studi konseptual pengembangan dan perancangan perangkat lunak terhadap infrastruktur dengan mengkolaborasikan pengembang (*Dev*) dan perangkat lunak (*Ops*) [4]. Ada beberapa tahap yang harus dilakukan untuk menerapkan metode *DevOps Life Cycle*, yaitu :

1. *Plan*

   Tahap ini bertujuan untuk menentukan sasaran dan kebutuhan dari sebuah perangkat lunak, beserta rencana awal untuk pembaruan dan rilis [6].

2. *Develop*

   Setelah pembuatan rencana pengembangan, *Developer* berfokus untuk mengembangkan dan me-*review source code* aplikasi. Biasanya pada tahap ini *developer* sudah sering melakukan *commit* pada *source code repository* serta mengintegrasikan dengan *unit testing* [6].

3. *Verify*

   Verifikasi merupakan proses mengevaluasi *software artifacts* berdasarkan syarat dan ketentuan yang telah dibuat [6].

4. *Test*

   Tahap ini merupakan proses dimana *automation testing* berjalan secara *countinously* untuk memastikan kualitas dan kelayakan *software artifact.* Tahap ini biasanya juga bisa disebut sebagai Beta *testing* [6].

5. *Deploy*

   Tahap ini fokus pada *continously re-deployment* dari sebuah perangkat lunak pada *environment production*. Tahap ini memerlukan konfigurasi sumber daya dan target *platform* [6].

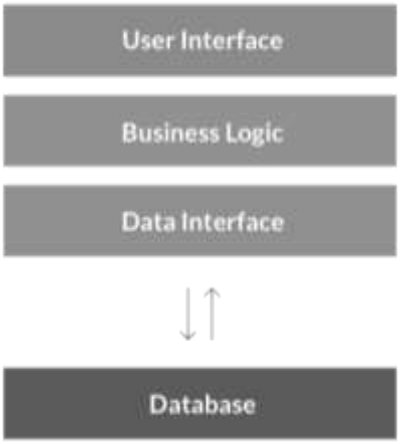
6. *Operate*

   Tahap ini berkaitan dengan konfigurasi dan memanajemen aplikasi setelah melakukan *deployment* seperti *resource provisioning, auto scaling* dan *load balancing* [6].

7. *Monitor*

   Tahap ini ada proses memantau kinerja aplikasi yang digunakan dengan pengumpulan dan menganalisis data penggunaan, data ini dapat digunakan untuk mendeteksi *bugs, logic error* kemudian memberikan *feedback* secara langsung untuk meningkatkan performa aplikasi (*Continous Tracing and Diagnostics)* [6].

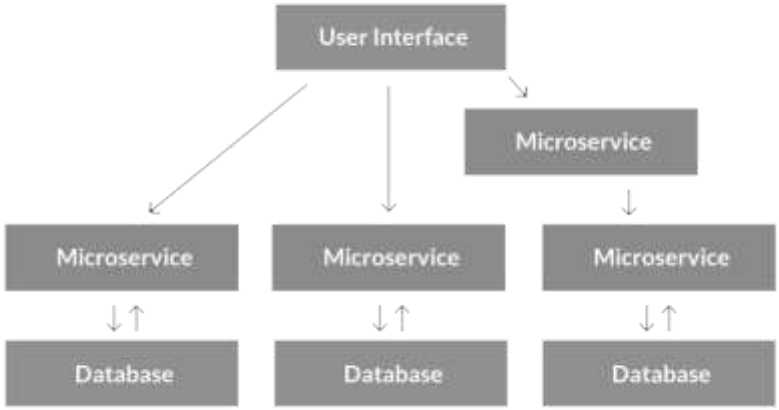
2.2.5 *Monolithic Architecture*



Gambar 2.3. Arsitektur Monolithic [12]

Arsitektur *monolithic* adalah suatu arsitektur yang menggambarkan sebuah aplikasi yang menjalankan semua logika dalam satu *server* aplikasi. Aplikasi yang dibangun dengan arsitektur *monolithic* hanya dijalankan dengan menggunakan satu *server* aplikasi sehingga hanya memerlukan proses pemeliharaan aplikasi pada satu *server* aplikasi. Arsitektur *monolithic* merupakan aplikasi perangkat lunak yang modulnya tidak dapat dijalankan secara independen. Dengan menggunakan arsitektur *monolithic*, setiap aplikasi dijalankan secara bersamaan dikarenakan seluruh modul-modul aplikasi terdapat di dalam sebuah aplikasi yang sangat besar [13].

2.2.6 *Microservice Architecture*



Gambar 2.4. Arsitektur Microservice [12]

Arsitektur *microservice* adalah gaya arsitektur perangkat lunak yang memerlukan pemecahan aplikasi secara bisnis. *Microservice* adalah sebuah arsitektur baru dimana pengembangan aplikasi dilakukan dalam bentuk *web service* kecil yang saling berkomunikasi satu sama lain.

2.2.7 *Unified Modelling Language*

Unified Modelling Language atau yang biasa disebut UML diciptakan oleh Grady Booch, Ivan Jacobson, dan James Rumbaugh pada tahun 1999 sebagai upaya untuk memberikan standar desain dan analisis pemodelan data berorientasi objek [9]. UML adalah pemodelan visual yang digunakan untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan setiap objek dari sistem perangkat lunak [14]. UML merupakan bahasa pemodelan yang menggabungkan metode orientasi obyek yang telah ada sebelumnya, yaitu, *Object Modeling Technique* (OMT), dan *Object Oriented Software Engineering* (OOSE) [9]. Pada dasarnya UML sendiri dibagi menjadi 2 jenis diagram yaitu [15] :

1. *Structural Diagram*

   Structural Diagram merupakan diagram yang digunakan untuk menggambarkan struktur statis yang akan dimodelkan [14], diantaranya adalah :

   a. *Class Diagram*

      Class Diagram digunakan untuk memodelkan objek dan kelas yang terlibat dalam penyediaan informasi sesuai yang akan dibangun [9]. Selain itu juga *class diagram* berguna untuk mendesain penyimpanan data [15].

   b. *Component Diagram*

      Component Diagram menggambarkan struktur dan hubungan antar *package*, termasuk saling ketergantungannya (*dependency*) [9].

c. *Deployment Diagram*

Deployment Diagram digunakan untuk memodelkan desain komponen *hardware* yang digunakan dalam implementasi aplikasi yang telah dibuat [15].

2. *Behavioral Diagram*

a. *Use Case Diagram*

Use Case Diagram digunakan untuk memodelkan fungsionalitas antara aktor dan sistem. Tujuan dari *use case* adalah untuk membuat daftar aktor dan *use case* dan menunjukkan aktor mana yang berpartisipasi dalam setiap *use case* [14].

b. *Activity Diagram*

Activity Diagram digunakan untuk memodelkan alur sebuah prosedur dari sebuah sistem dan digambarkan berdasarkan kegiatan per aktor yang terlibat di dalam sistem [9], [15].

c. *Sequence Diagram*

Sequence Diagram digunakan untuk memodelkan interaksi antara aktor dengan sistem [15]. Pembuatan *sequence diagram* harus berdasarkan *use case diagram* yang telah dibuat.

d. *Collaboration Diagram*

Collaboration Diagram dalah diagram yang mengelompokkan pesan pada kumpulan diagram sekuen menjadi sebuah diagram. Dalam diagram tersebut terdapat *method* yang dijalankan antara objek yang satu dan objek lainnya [14].

2.2.8 *User Acceptance Testing (UAT)*

User Acceptance Testing (UAT) adalah sebuah bentuk pengujian yang dilakukan oleh pihak akhir pengguna (*end user*). pihak akhir tersebut adalah pengguna yang akan langsung berinteraksi menggunakan sistem dan melakukan verifikasi terkait dengan fungsi, apakah sudah berjalan dengan baik dan sesuai dengan fungsi beserta kebutuhannya [16].

2.2.9 *BlackBox Testing*

Black Box Testing atau yang biasa disebut pengujian fungsional adalah metode pengujian perangkat lunak yang digunakan untuk menguji perangkat lunak tanpa mengetahui struktur internal dari kode atau program [10].

Saat ini tersedia beberapa teknik untuk melakukan pengujian menggunakan *BlackBox* :

1. *Boundary Value Analysis (BVA)*

   Boundary Value Analysis merupakan pengujian yang dilakukan dari masukan yang terkecil hingga terbesar dan secara efektif dapat menangkap *error* pada aplikasi sebanyak mungkin [17].
2. *Decision Tables (DT)*

   Decision Tables merupakan pengujian yang dilakukan untuk mengetahui pengaruh sejumlah kombinasi data masukkan pengguna terhadap hasil keluaran yang ditampilkan sistem [18].
3. *Equivalence Partitioning (EP)*

   Equivalence Partitions merupakan sebuah pengujian berdasarkan masukkan data pada setiap *form* yang ada, setiap menu masukan akan dilakukan pengujian dan dikelompokkan berdasarkan fungsinya baik itu bernilai *valid* ataupun tidak *valid* [19].

Metode pengujian menggunakan *Black Box* terdiri dari 3 langkah utama [17], yaitu :

1. Mendifinisikan kasus pengujian, kumpulan kasus pengujian disimpan dalam sebuah dokumen agar dapat digunakan kembali [17].
2. Eksekusi kasus pengujian, mengeksekusi setiap kasus pengujian dan meyimpan kedalam dokumen [17].
3. Hasil pengujian, setiap pengujian diberikan keterangan *valid* atau *invalid* dan membuat laporan pengujian [17].

2.2.10 *Automated Testing*

Automated Testing adalah pengujian terhadap suatu sistem aplikasi secara otomatis menggunakan alat bantu untuk mengeksekusi skenario uji

coba yang telah dibuat. Piranti yang dikembangkan untuk melakukan *automated testing* juga bisa memproses data dalam sistem yang diuji coba, membandingkan hasil yang didapat dan menghasilkan laporan detail dari pengujian yang dilakukan [20].

Terdapat 2 jenis pengujian yang paling utama yaitu :

1. *Unit Testing*

   Unit testing merupakan jenis pengujian otomatis yang dilakukan pada *codebase internal* [20].

2. *Integration Testing*

   Integration Testing merupakan pengujian yang dilakukan secara otomatis untuk menguji service luar yang terintegrasi dengan kode program yang terdapat pada internal [20].