# BAB 2
# PENDAHULUAN

## 2.1 Tinjauan Perusahaan

Berikut adalah tinjauan perusahaan mulai dari profil, riwayat, visi-misi, dan struktur Organisasi perusahaan.

### 2.1.1 Profil Perusahaan


Gambar 2.1 Perusahaan

PT Svara Inovasi Indonesia merupakan sebuah Start-Up yang bergerak di bidang media, broadcasting, & community Platform. Svara berdiri pada tanggal 9 September 2017. Meski baru 5 tahun berdiri, Svara telah memiliki pengalaman yang banyak dalam bidang Teknologi, Informasi, dan Komunikasi (TIK), karena Svara dibentuk oleh PT Zamrud Khatulistiwa Technology yang memiliki banyak pengalaman dalam bidang transformasi digital sejak tahun 2005.

Platform Svara meliputi Svara On-Air dan Svara Online. Svara On-Air merupakan aplikasi Broadcasting Automation untuk mendukung modernisasi siaran dan memudahkan dalam pengelolaan kontennya. Sedangkan Svara Online merupakan aplikasi bagi user untuk menikmati konten-konten yang dibuat oleh Broadcaster. Konten dapat dinikmati melalui Mobile Apps, Web Apps, dan IoT seperti Smart Speaker, SmartTV, dan Connected Car sehingga user memiliki

*experience* baru dalam menikmati berbagai konten seperti Streaming Radio, Playlist Music, Podcast, dan juga Video.

Tabel 2.1 Deskripsi Perusahaan

| | |
|---|---|
| **Nama Perusahaan** | PT Svara Inovasi Indonesia |
| **Bidang Perusahaan** | Penyiaran Radio Oleh Swasta, Perdagangan Besat Atas Dasar Balas Jasa (fee) Atau Kontrak, Aktivitas Teknologi Dan Jasa Komputer Lainnya, Aktivitas Jasa Information Lainnya Ytdl. |
| **Alamat Perusahaan** | Jl. Bungur No. 9, Cipedes, Kota Bandung, Jawa Barat 40162 |
| **Akte Pendirian Perusahaan** | No. 17 tanggal 9 September 2017 |
| **Nomor Induk Berusaha** | 9120103981305 |
| **Nomor Telepon** | +62 (22) 82045711 |
| **Surat Izin Usaha** | 0093/IUP/V/2018/DPMPTSP |
| **Website Perusahaan** | www.svarainnovation.com |
| **E-mail Perusahaan** | info@svarainnovation.co.id |

### 2.1.2 Riwayat Perusahaan

Dimulai dengan melihat bahwa bisnis dari industri radio dan musik dalam 15 tahun terakhir terus menurun, salah satu penyebabnya adalah DISRUPTION yang dilakukan Radio Internet Pureplay seperti Spotify, Joox, TuneIn, SoundCloud, dll. SVARA hadir dengan konsep *Beyond Disruption* (bukan mengganggu industri yang ada tapi justru menyelamatkan industri radio dan musik) yang diawali dengan melakukan Radio Digital Transformation.


Gambar 2.2 Riwayat Perusahaan

### 2.1.3 Visi dan Misi Perusahaan

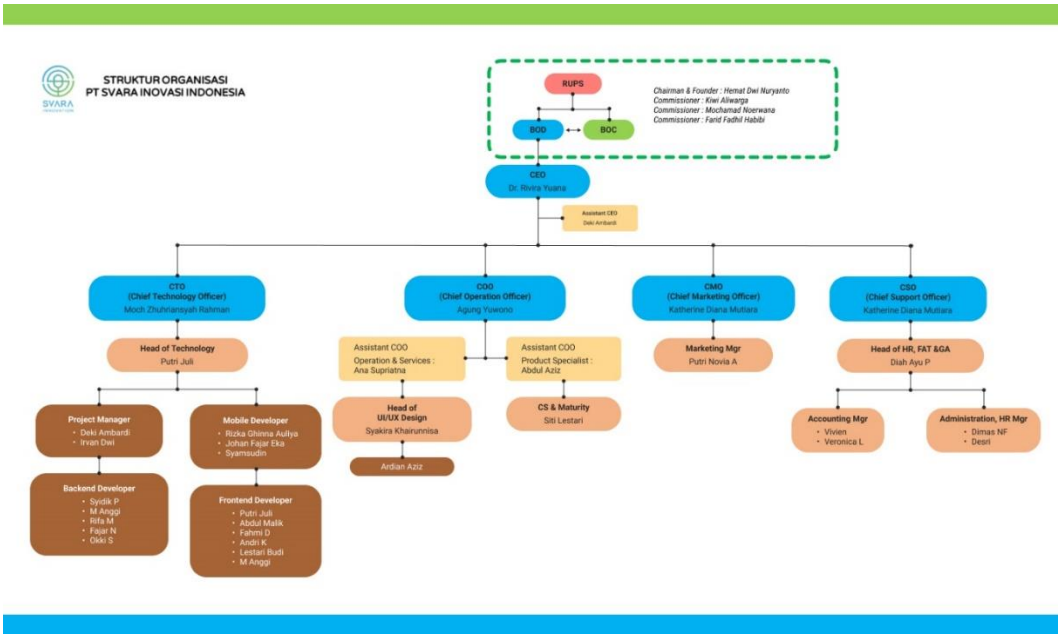
Pada umumnya instansi pemerintahan memilih visi dan misi, PT Svara Inovasi Indonesia pun memiliki visi dan misi yang ingin dicapai, karena visi dan misi merupakan bayangan dari kegiatan yang akan dilakukan dan tujuan perusahaan. Berikut dibawah adalah visi dan misi dari PT Svara Inovasi Indonesia.

Visi : *Creating a Smarter World*

Misi : *Svara's Mission is to Bettering the World's Information, Entertainment and Education through Digital Transformation & Innovation of the Media and Broadcasting Industry.*

### 2.1.4 Struktur Organisasi Perusahaan

Struktur organisasi dalam perusahaan bermaksud untuk merumuskan tugas pokok dan fungsi berbagai bidang. Berikut adalah struktur organisasi di PT Svara Inovasi Indonesia.




Gambar 2.3 Struktur Organisasi

## 2.2 Landasan Teori

Berikut beberapa landasan teori terkait aplikasi dan dokumen yang akan dibuat dalam penelitian.

### 2.2.1 Software Engineering

*Software engineering* merupakan disiplini ilmu yang berkaitan dengan semua aspek produksi perangkat lunak dari tahap awal spesifikasi sistem sampai pemeliharaan sistem setelah digunakan. Penerapan *software engineering* dalam teknologi informasi tidak dapat berdiri sendiri, banyak disiplin ilmu yang mendukung keberhasilan dari *software engineering* tersebut [8].

*Software engineering* adalah ilmu yang memebahas seluruh aspek produksi perangkat lunak, mulai dari tahap awal spesifikasi sistem sampai pemeliharaan sistem setelah digunakan. *Software engineering* penting untuk diterapkan karena semakin banyak individu dan Masyarakat mengandalkan kemajuan sistem perangkat lunak. Seorang *software engineer* harus bisa memproduksi sistem terpercaya dan dapat diandalkan secara ekonomis dan cepat.

### 2.2.2 *Software Reengineering*

*Reengineering* atau rekayasa ulang konsep yang pertama kali dikemukakan oleh Michael Hammer dan James Champy di awal tahun 1990 bernama *Business Process Reengineering* [2]. Proses *software reengineering* adalah proses memodifikasi dan merombak *software* yang sudah ada agar dapat lebih terpelihara. Proses ini adalah menulis ulang atau restrukturisasi keseluruhan sistem tanpa merubah fungsionalitas dari suatu sistem [9].

Pendekatan *reengineering* yang akan dilakukan pada penelitian ini adalah *Enhanced Re-engineering Mechanism*. Mekanisme rekayasa ulang yang ditingkatkan adalah proses rekayasa ulang perangkat lunak yang memanfaatkan banyak hal metode dan level abstraksi, untuk mengubah yang sudah ada sistem perangkat lunak ke sistem perangkat lunak target baru [9]. Metodologi kerjanya adalah sebagai berikut dan dijelaskan dalam sub bagian berikutnya. Mekanisme

rekayasa ulang yang ditingkatkan memiliki lima fase yaitu studi kelayakan dan persyaratan, SRS yang direstrukturisasi, desain ke kode, Perbandingan Fungsi yang Ada dan yang diusulkan dan Implementasi. Berikut adalah lima fase dalam *Enhance Re-engineering* :

1. Studi kelayakan dan persyaratan

   Dalam pekerjaan ini, tahap awal adalah studi kelayakan dan persyaratan. Pada tahap ini dilakukan studi kelayakan periksa konfigurasi dan kompatibilitas komputer sistem. Setelah selesai studi kelayakan, kebutuhan sistem ditentukan ulang berdasarkan keinginan pengguna. Itu SRS memiliki semua persyaratan dalam struktur tertulis sebagai dokumen resmi. Untuk menentukan ulang sistem persyaratan, sistem perlu memetakannya dengan SRS.

2. Spesifikasi persyaratan sistem yang direstrukturisasi

   Tahap ini menggambarkan secara rinci SRS yang direstrukturisasi Proses. Dokumentasi merupakan atribut penting dalam proses pengembangan perangkat lunak karena mereproduksi komponen dari proses rekayasa ulang total dan bertindak sebagai perencana untuk produk akhir. Di sini, itu para ahli membuat perbandingan antara persyaratan sistem yang ada dan dengan mekanisme yang baru. SRS adalah digunakan untuk mengintegrasikan kedua SRS baru dengan SRS yang sudah ada.

3. *Design to Code*

   Tahap ini menawarkan detail tentang desain hingga kode proses. Pada tahap ini sesuai dengan desain ulang dokumen kode dilakukan oleh programmer. Biasanya, tua algoritma diimplementasikan dalam pengembangan tradisional bahasa dan dengan fungsi yang ada perlu ditulis ulang. Misalnya saja jika suatu sistem teknologi memerlukan waktu dan ketelitian dan sudah menjadi tua untuk menjadi akurat dan membutuhkan yang baru algoritma jadi sistem, maka perlu direncanakan rekayasa ulang dengan teknik baru.

4. Perbandingan fungsi yang ada dan yang diusulkan

Tahap ini memberikan rincian tentang proses pengujian ulang. Untuk pengujian ulang, terlebih dahulu baik perangkat lunak yang sudah ada maupun yang baru lamaran diambil. Kemudian membandingkan kinerjanya fungsionalitas aplikasi perangkat lunak yang ada dengan fungsionalitas aplikasi perangkat lunak baru. Untuk evaluasi, jika suatu sistem menggunakan standar seperti memori penggunaan, waktu pengoperasian, dan konfigurasi sistem. Lalu, itu kinerja fungsi lama dibandingkan dengan yang diusulkan algoritma. Berdasarkan hasilnya, proses pembangunan kembali harus dilakukan. Sebagai perbandingan hasil, jika suatu algoritma mendapat kinerja lebih dari yang lain, maka semakin baik algoritma kinerja diganti untuk proses pembangunan kembali.

5. Implementasi

Tahap ini merupakan tahap dari rekayasa ulang yang ditingkatkan mekanisme. Berdasarkan hasil tahapan rekayasa ulang sebelumnya, implementasi suatu perangkat lunak lamaran bisa lengkap. Dalam pelaksanaannya, part tertentu bisa diganti dengan part yang bagus sepenuhnya bergantung pada empat tahap sebelumnya yang ditingkatkan mekanisme rekayasa ulang perangkat lunak.

6. Pengujian

Tahap ini merupakan tahap terakhir dari rekayasa ulang. Pengujian dilakukan setelah dilakukannya tahap implementasi untuk di uji kembali hasil dari *enchanced reengineering*.

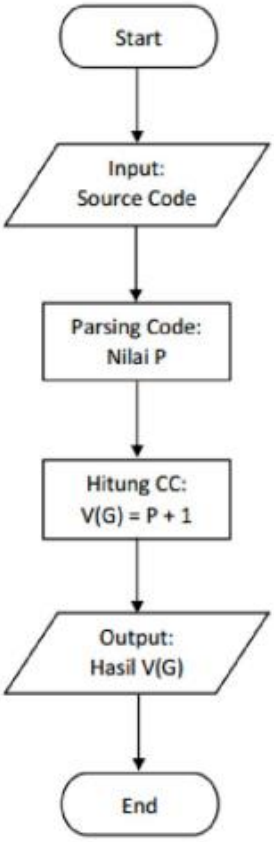
### 2.2.3 Kompleksitas

Kompleksitas perangkat lunak menunjukan kerumitan dalam memahami, memelihara, memodifikasi dan menggunakan kembali perangkat lunak [10]. Jika nilai kompleksitas tinggi maka *developer* akan kesulitan dalam memahami perangkat lunak pada tahap pemeliharaan maupun jika akan melakukan perubahan. Oleh karen itu kompleksitas perangkat lunak perlu dipantau untuk menetapkan kualitas dan kehandalan perangkat lunak tersebut menggunakan *software metric*.

Pengukuran merupakan dasar dari setiap disiplin rekayasa dan berlaku juga dalam perekayasaan perangkat lunak. Menurut *IEEE Standard Glossary of Software Engineering terminology* (1990), pengukuran adalah ukuran kuantitatif dari Tingkat dimana sebuah sistem, komponen atau proses memiliki atribut tertentu. Sedangkan, mengukur adalah mengindikasi kuantitaif dari luasan, jumlah, dimensi dan kapasitas [11].

Terdapat beberapa metode pengukuran kompleksitas pada perangkat lunak, yaitu LOC (line of code), McCabe's Cyclomatic Complexity dan halstead's Volume. LOC adalah suatu Teknik pengukuran dengan cara menghitung jumlah keseluruhan baris kode pada seuluruh file kode sumber, Sedangkan McCabe's Cyclomatic Complexity dan Halstead's Volume adalah atribut yang secara umum digunakan untuk menghitung Tingkat kompleksitas kode sumber.

Metrik *Halstead* digunakan untuk mengevaluasi dan melakukan pengukuran kode sumber berdasarkan pada operator dan operand. Dan *Cyclomatic Complexity* merupakan pendekatan ukurna kompleksitas yang dilakukan untuk mengukur dan mengontrol jumlah alur melakukan program. Semakin tinggi angka *Cyclomatic Complexity* akan meningkatkan *error* serta pemeliharaannya. Sehingga, semakin rendah kompleksitas maka semakin tinggi kuelitasnya. Perhitungan hasil kompleksitas berdasarkan metrik *Halstead's Volume* dan *Cyclomatic Complexity* tersebut berperan sebagai pendukung proses pendeteksian cacat secara dini pada perangkat lunak. Dalam penelitian ini, pengembangan sistem perhitungan kompleksitas kode sumber menggunakan *library java parser* dan *AST*.




Gambar 2.4 *Flowchart* Metode *Cyclomatic Complexity*

*Cyclomatic Complexity* adalah suatu metrik perangkat lunak yang digunakan untuk mengukur kompleksitas sebuah potongan kode atau algoritma dari sebuah program. *Cyclomatic Complexity* dapat diukur dengan memanfaatkan graf aliran control dari suatu fungsi, modul, metode maupun kelas yang terdapat pada suatu program [12].

Cyclomatic Complexity diukur berdasarkan jumlah jalur independent linear pada kode. Jika pada kode tidak terdapat jalur Keputusan seperti percabangan atau perulangan, maka kompleksitasnya adalah 1, Sedangkan jika terdapat dua jalur Keputusan yang tergantung pada satu blok simpul, maka kompleksitasnya 2. Untuk memudahkan perhitungan kompleksitas pada graf yang cukup kompleks, dapat digunakan beberapa persamaan yaitu

$$V(G) = E - N + 2$$

Dengan:

E = Jumlah sisi

N = Jumlah simpul

$$V(G) = P + 1$$

Dengan:

P = Jumlah node yang mengandung jalur Keputusan

Arti dari nilai V(G) yang didapat memiliki makna yang disajikan pada tabel 2.4 berikut

Tabel 2.2 Klasifikasi *Cyclomatic Complexity*

| V(G)/Nilai Kompleksitas | Makna |
|---|---|
| 1-10 | Kode terstruktur dan ditulis dengan baik, mudah diuji, biaya pengembangan lebih sedikit |
| 11-20 | Kode cukup kompleks, relative lebih sulit untuk diuji, biaya pengembangan lebih tinggi |
| 21-50 | Kode sangat kompleks, sulit untukdiuji, biaya pengembangan tinggi |
| >50 | Kode terlalu kompleks, sangat sulit bahkan tidak dapat diuji, biaya pengembangan sangat tinggi. |

### 2.2.4 *Maintainability*

*Maintainability* adalah ukuran dari seberapa mudah suatu sistem perangkat lunak dapat dipertahankan atau dilakukan pemeliharaan (*maintenance*) [8]. *Maintenance* dari perangkat lunak terdapat tiga jenis diantara lain sebagai berikut:

1. *Maintenance* untuk Memperbaiki kesalahan yang terjadi atau biasa disebut dengan *fault repair.*
2. *Maintenance* apabila terdapat perubahan seperti sistem operasi, perangkat keras, dan juga perangkat lunak pendukung lain pada lingkungan perangkat lunak yang disebut dengan *platform adaption.*

3. *Maintenance* apabila terjadi penambahan fitur atau perubahan terhadap kebutuhan yang merupakan perubahan dari bisnis atau Organisasi, hal tersebut biasa disebut dengan *functionality addition/system enhancement.*

Maintainability didefinisikan sebagai kemudahan untuk memodifikasi sistem atau komponen dari perangkat lunak [13]. Kemampuan *maintainability* sendiri merupakan atribut kualitas kunci yang menentukan keberhasilan suatu produk perangkat lunak. Dalam hal pembiayaan sendiri, perawatan perangkat lunak adalah suatu bagian terbbesar dari total biaya pembuatan aplikasi perangkat lunak. Beberapa studi memperkirakan bahwa maintenance membutuhkan hingga 80% dari total biaya yang digunakan. Sehingga perlu untuk mengetahui nilai maintainability sejak awal pengembangan perangkat lunak.

### 2.2.5 *Reusability*

*Reusability* adalah kemampuan dapat digunakan kembali komponen perangkat lunak untuk mengurangi waktu, biaya, dan sumber daya manusia. Memperkirakan faktor reusability perlu dari awal siklus hidup pengembangan perangkat lunak karena dapat sangat mengurangi biaya pengembangan produk dan meningkatkan kepuasan pelanggan [7]. Menurut Chidamber dan Kemerer, Kemampuan produk software yang akan digunakan di tempat produk software lain yang ditentukan untuk tujuan yang sama dalam lingkungan yang sama [14]. Kriteria reusability senantiasa mendukung perancang untuk meningkatkan rancangan perangkat lunak pada tahap awal proses pengembangan perangkat lunak. Mengukur nilai reusability membutuhkan alat ukur atau metrik perangkat lunak. Telah dilakukan beberapa penelitian tentang pengukuran reusability baik dari kode program maupun dari rancangan perangkat lunak.

Pengukuran reusability pada kode program telah dilakukan oleh Washizaki yang berjudul “Reusability Metrics for Program Source Code Written in C Language and Their Evaluation” menggunakan metode Goal-Question-Metric (GQM) berhasil mengembangkan prosedur yang akurat untuk mengukur reusability (Washizaki et al., 2012). Kemudian pengukuran nilai reusability pada rancangan

perangkat lunak telah dilakukan oleh Huda yang berjudul "Quantifying Reusability of Object Oriented Design : A Testability Perspective" menggunakan metrik properti desain berorientasi objek yaitu coupling, encapsulation, dan inheritance telah divalidasi dan memiliki nilai acceptance yang tinggi yaitu 95%.

Object-oriented metrics (OOM) berperan penting dalam industri perangkat lunak karena hampir semua proyek mengembangkan perangkat lunak dengan konsep berorientasi objek. Mengukur kualitas dan kuantitas adalah tugas yang menantang bagi pengembang. Arti dari metrik adalah kuantitas yang terukur. Banyak metrik perangkat lunak telah dikembangkan dan digunakan untuk prediksi kesalahan, yang dapat memperkirakan ukuran, kompleksitas, kohesi, dan kopling. Beberapa metrik yang dapat dihitung dari informasi desain adalah Quality Model ISO/IEC 9126-1 dengan metrik Chidamber dan Kemerer seperti yang ada pada Tabel 2.3

Tabel 2.3 *Reusability Metric*

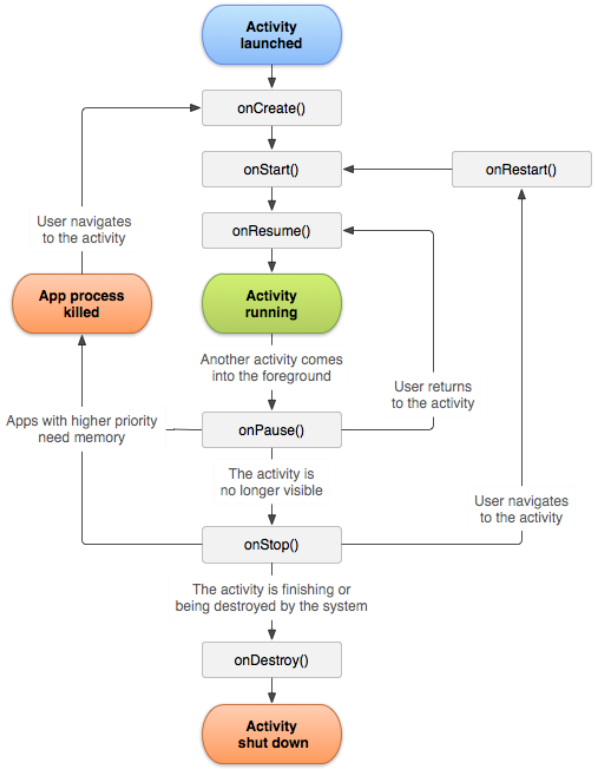
| Metric | Nama | Deskripsi |
|---|---|---|
| CBO | Direct Class Coupling (Coupling Metric) | Metrik ini menghitung jumlah kelas yang berbeda yang berelasi langsung dengan kelas. Metrik ini mencakup kelas-kelas yang secara langsung berelasi dengan deklarasi atribut dan pengiriman pesan (parameter) dalam method. |
| LCOM | Measure of Functional Abstraction (Inheritance Metric) | Metrik ini adalah rasio jumlah method yang diwarisi oleh kelas dengan jumlah method yang dapat diakses oleh method anggota kelas. (Kisaran 0 hingga 1) |

### 2.2.6 Android

Android adalah sebuah sistem operasi untuk perangkat mobile berbasis linux yang mencakup sistem operasi, middleware, dan aplikasi. Android adalah sistem operasi untuk telepon seluler yang berbasis Linux. Android menyediakan platform terbuka bagi para pengembang untuk membuat aplikasi mereka sendiri. Pada awalnya dikembangkan oleh Android Inc, sebuah perusahaan pendatang baru yang membuat perangkat lunak untuk ponsel yang kemudian dibeli oleh Google

Inc. Untuk pengembangannya, dibentuklah Open Handset Alliance (OHA), konsorsium dari 34 perusahaan perangkat keras, perangkat lunak, dan telekomunikasi termasuk Google, HTC, Intel, Motorola, Qualcomm, T-Mobile, dan Nvidia7 [15].

Saat pengguna menelusuri, keluar, dan kembali ke aplikasi Anda, instance Activity dalam aplikasi Anda melakukan transisi ke berbagai status dalam siklus prosesnya. Class Activity menyediakan sejumlah callback yang memungkinkan aktivitas mengetahui bahwa status telah berubah: bahwa sistem membuat, menghentikan, atau melanjutkan suatu aktivitas, atau menutup proses tempat beradanya aktivitas. Untuk menavigasi transisi di antara tahap siklus porses aktivitas, class Activity menyediakan set inti sebanyak enam callback: onCreate(), onStart(), onResume(), onPause(), onStop(), dan onDestroy(). Sistem memanggil masing-masing callback ini saat aktivitas memasuki status baru [16].




Gambar 2.5 Android Lifecycle

Setelah pengguna mulai meninggalkan aktivitas, sistem memanggil metode untuk membongkar aktivitas. Biasanya, pembongkaran ini hanya sebagian; aktivitas masih berada dalam memori (seperti ketika pengguna beralih ke aplikasi lain), dan masih dapat kembali ke latar depan. Jika pengguna kembali ke aktivitas itu, aktivitas akan dilanjutkan dari tempat pengguna keluar. Dengan beberapa pengecualian, aplikasi dibatasi dari memulai aktivitas saat berjalan di latar belakang. Kemungkinan sistem untuk menutup proses yang diberikan bersama dengan aktivitas di dalamnya tergantung pada status aktivitas pada saat itu. Status aktivitas dan pengeluaran dari memori memberikan informasi selengkapnya tentang hubungan antara status dan kerentanan terhadap pengeluaran.

### 2.2.7 Android Studio

Android Studio merupakan Integrated Development Environment (IDE) atau dalam artian lain adalah sebuah lingkungan pengembangan terintegrasi resmi yang memang di rancang khusus untuk pengembangan sistem operasi google Android [17]. Android studio ini adalah lingkungan pengembangan baru dan terintegrasi dengan penuh,yang telah di rilis oleh google untuk sistem operasi Android dan di rancang untuk menjadi peralatan baru dalam pengembangan aplikasi dan memberi alternaif selain Eclips yang saat ini menjadi IDE yang banyak di pakai. Android Studio memiliki fitur yang banyak dan dapat dipercaya baik oleh pemula maupun programmer professional untuk mengembangkan aplikasi android. Fungsi-fungsinya adalah sebagai berikut:

1. Sistem control versi Gradle yang fleksibel.
2. Emulator yang cepat dan kaya fitur.
3. Lingkungan terintegrasi untuk mengembangkan aplikasi android untuk semua perangkat Android.
4. Kode dan Integrasi GitHub menciptakan fungsionalitas untuk aplikasi yang sama.
5. Dukungan Google Cloud Platform untuk integrasi yang mudah dari Google Cloud Messaging dan App Engine.

### 2.2.8 *Software Development Kit* (SDK)

Pada pemakaian Android Studio, tentunya diperlukan SDK pada saat peng-*instalan*-nya. Android *Software development kit* (SDK) merupakan *kit* yang bisa digunakan oleh para *developer* untuk mengembangkan aplikasi berbasis Android. Di dalamnya terdapat beberapa *tools* seperti *debugger*, *software libraries*, *emulator*, dokumentasi *sample code* dan tutorial [18].

Java SE Development kit adalah salah satu contoh Android SDK dan menjadi bahasa pemrograman yang paling sering digunakan untuk mengembangkan aplikasi Android. Di samping itu ada beberapa bahasa lainnya seperti C++, Go, dan Kotlin sebagai bahasa yang ditetapkan Google pada tahun 2017 lalu.

### 2.2.9 SonarQube

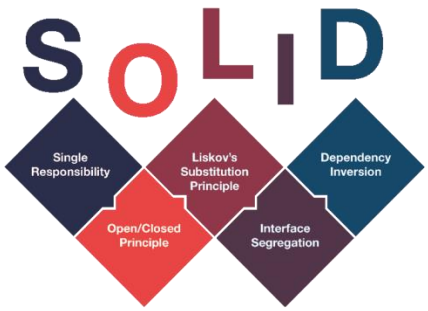
SonarQube adalah platform analisis kode statis yang populer dalam dunia pengembangan perangkat lunak. Alat ini dirancang untuk meningkatkan kualitas dan keamanan kode sumber dengan cara menganalisis dan mengidentifikasi berbagai masalah seperti bug, kerentanan keamanan, dan code smells. SonarQube mendukung berbagai bahasa pemrograman, termasuk Java, C#, JavaScript, Python, dan banyak lagi, membuatnya sangat serbaguna untuk berbagai jenis proyek. Salah satu keunggulan utamanya adalah kemampuannya untuk mengintegrasikan dengan alat pengembangan lain dan pipeline CI/CD, memungkinkan analisis kode otomatis sebagai bagian dari proses pengembangan [19].

SonarQube menyajikan hasil analisisnya melalui dashboard yang intuitif, menyediakan berbagai metrik kode dan memungkinkan tim pengembang untuk dengan cepat mengidentifikasi area yang memerlukan perbaikan. Selain itu, platform ini memungkinkan pengguna untuk menyesuaikan aturan analisis sesuai dengan kebutuhan spesifik tim atau organisasi. Dengan kemampuan untuk melacak masalah kode dari waktu ke waktu, SonarQube menjadi alat yang sangat berharga bagi tim pengembang yang berkomitmen untuk meningkatkan dan mempertahankan kualitas kode secara konsisten dalam jangka panjang [19].

### 2.2.10 *Clean Architecture*

*Clean Architecture* adalah pendekatan dalam perancangan perangkat lunak yang menekankan pemisahan konsep dan tanggung jawab, dengan fokus pada pemeliharaan kejelasan dan fleksibilitas kode [6]. Beberapa prinsip dasar *Clean Architecture* melibatkan pemisahan antara lapisan-lapisan perangkat lunak, dengan prinsip *dependency rule* sebagai salah satu aspek kunci. Prinsip ini menyatakan bahwa arah ketergantungan dalam kode harus mengarah ke arah lapisan yang lebih dalam, sehingga lapisan-lapisan dalam aplikasi tidak tergantung pada detail implementasi lapisan-lapisan yang lebih luar.

Sistem dapat berubah dengan cepat, maka pemanfaatan *clean architecture* juga penting untuk menyeimbangi kecepatan *development*. Kode yang memanfaatkan *clean architecture* diperlukan dengan maksud untuk membuat kode rapi dan baik. Setiap potongan kode memiliki arti dan fungsi sesuai tugasnya masing-masing. Terlebih lagi Ketika melakukan kolaborasi pekerjaan, akan sangat memudahkan apabila program yang dibuat mengimplemnetasikan *clean architecture* di dalamnya sehingga *programmer/developer* lain dapat melanjutkan pekerjaan tersebut. *Clean architecture* juga mendorong praktik-praktik seperti pengujian otomatis, penggunaan pola desain, dan prinsip-prinsip SOLID untuk memastikan kode yang dihasilkan bersih, teruji, dan mudah diubah.




Gambar 2.6 SOLID Principle

Berikut adalah penjelasan dari prinsip-prinsip SOLID [20]:

1. Single Reponsibility Principle

Uncle Bob menyatakan bahwa sebuah class atau module sebaiknya memiliki hanya satu alasan untuk berubah yang berarti hanya memiliki satu tanggung jawab dalam sebuah class / module.

2. Open-Closes Principle

Suatu class harus bisa diekstensi dan tidak boleh dimodifikasi. Dengan menerapkan hal tersebut, kita akan berhenti untuk memodifikasi code yang sudah ada dan terhindar dari potensi error yang akan terjadi.

3. Liskov Substitution Principle

Liskov Substitution Principle ini memperluas Open Closed Principle dengan memfokuskan pada behavior dari superclass dan turunannya. Prinsip ini dapat juga diartikan bahwa objek-objek dari superclass seharusnya dapat digantikan dengan objek-objek yang merupakan turunan dari superclass tanpa merusak aplikasi. Dimana mengharuskan objek-objek dari turunan superclass memiliki behavior yang sama dengan superclassnya.

4. Interface Segregation

Maksud dari Interface Segretion adalah memecah Interface yang besar menjadi Interface yang kecil-kecil sehingga suatu class dapat mengimplementasi Interface-interface yang dibutuhkan saja. Sebagai contoh kita memiliki Interface transportasi darat, udara dan air. Kemudian kita memiliki satu class mobil yang hanya mengimplementasi transportasi darat. Lalu ada class pesawatAir yang mampu mengimplementasi interface udara dan air.

5. Dependency Inversion

Dependency Inversion adalah prinsip yang menyetakan bahwa sebuah entitas itu bergantung pada abstraksi. Sehingga sebuah modul tingkat tinggi tidak boleh bergantung pada modul tingkat rendah, tetapi bergantung kepada abstraksi.

Pada clean architecture, dibutuhkan clean code yang membantu memastikan bahwa kode dalam setiap lapisan tersebut ditulis dengan baik, sehingga membuat aplikasi lebih mudah dipahami, lebih mudah diubah, dan lebih stabil. Dengan demikian, Clean Architecture dan Clean Code saling melengkapi dan membantu mencapai tujuan yang sama [21]. Terdapat beberapa hal yang merupakan komponen untuk memenuhi clean code :

1. Meaningful Names

   Kelas dan objek harus memiliki nama kata benda atau frase kata benda seperti Customer, WikiPage, Account, dan AddressParser. Hindari kata-kata seperti Manager, Processor, Data, atau Info dalam nama kelas. Nama kelas tidak boleh menjadi kata kerja.
2. Clean Functions

   Fungsi-fungsi yang bersih adalah fungsi-fungsi yang ringkas, fokus, dan hanya melakukan satu hal. Mereka harus memiliki tanggung jawab tunggal dan tidak boleh terlalu panjang atau kompleks. Fungsi yang bersih harus mudah dibaca dan dipahami, dengan struktur yang jelas dan konsisten.
3. Clean Comments

   Komentar-komentar yang bersih adalah komentar-komentar yang diperlukan, ringkas, dan jelas. Mereka harus menjelaskan mengapa kode ditulis dengan cara tertentu, tetapi tidak bagaimana kode itu ditulis. Komentar tidak boleh mengulangi kode atau menjelaskan apa yang kode lakukan, tetapi harus memberikan konteks atau penjelasan tambahan.
4. Clean Code Formatting

   Format kode yang bersih mengacu pada organisasi dan tata letak kode. Ini termasuk hal-hal seperti indentation, spasi, dan panjang baris. Format kode yang bersih membuat kode mudah dibaca dan dipahami, dengan struktur yang jelas dan konsisten.
5. Clean Error Handling

   Penanganan kesalahan yang bersih mengacu pada cara kesalahan ditangani dan dilaporkan dalam kode. Ini termasuk hal-hal seperti blok try-catch, pesan kesalahan, dan logging. Penanganan kesalahan yang bersih

membuatnya mudah untuk mengidentifikasi dan memperbaiki kesalahan, dan memberikan pengalaman pengguna yang baik bahkan ketika kesalahan terjadi.

### 2.2.11 *Design Pattern*

*Design pattern* atau yang dapat diartikan sebagai pola desain adalah metode yang dibuat untuk membantu tim pengembang dalam menemukan solusi dari masalah-masalah umum yang muncul saat pengembangan perangkat lunak sedang berlangsung. Pola desain ini dapat digunakan kembali dalam pengembangan perangkat lunak selanjutnya. Ia bukanlah suatu metode yang dapat diimplementasikan langsung menjadi sebuah kode program, tetapi ia merupakan sebuah pola untuk menyelesaikan masalah dalam situasi yang bermacam-macam [22].

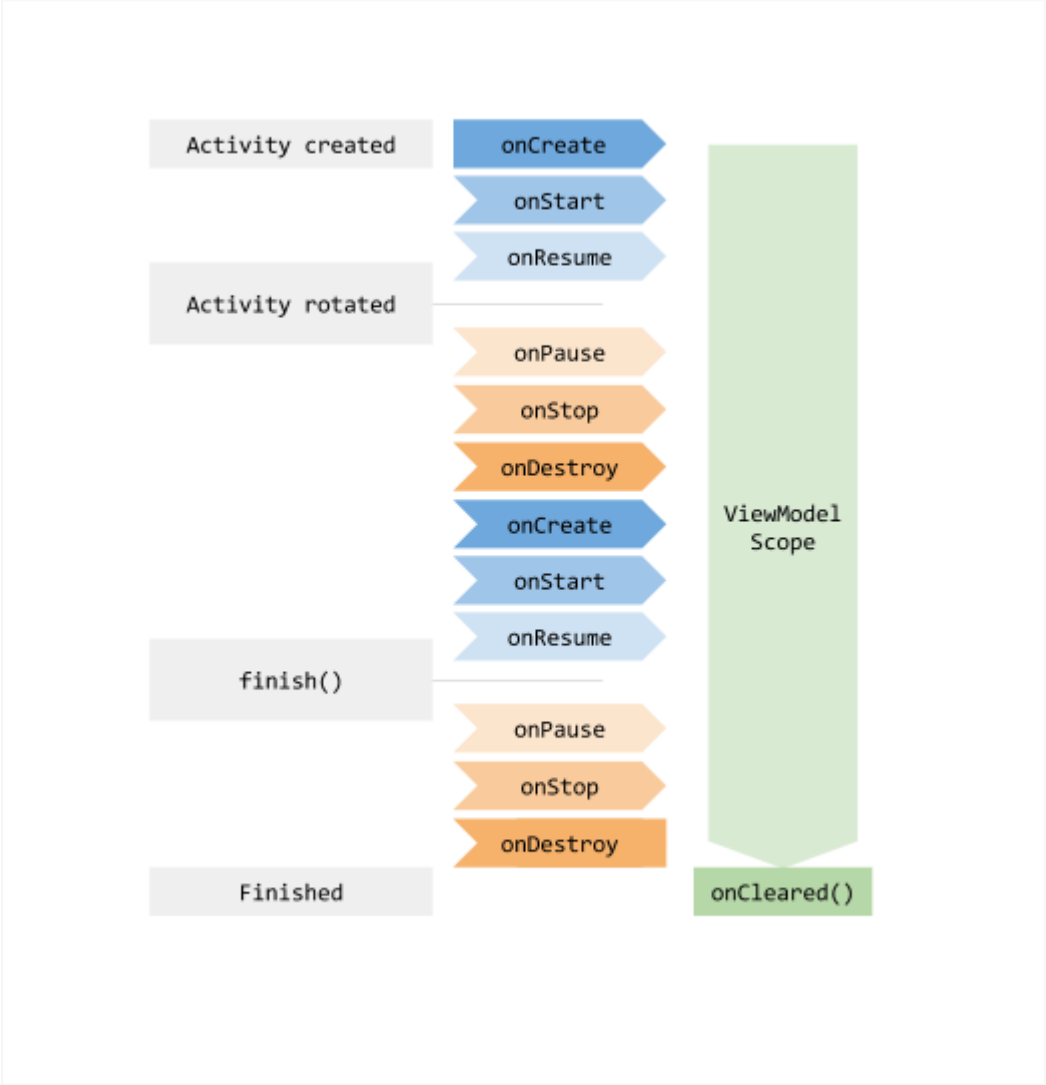
Dengan adanya Design Pattern, developer dapat terbantu dalam merancang perangkat lunak dengan cara yang terorganisr, mudah dipahami dan dapat digunakan kembali untuk masalah-masalah yang serupa. Adanya solusi yang teruji kegunaannya dalam berbaga situasi yang sering terjadi dalam pengembangan perangkat lunak. Pola-pola desain tersebut dapat mengurangi kebingungan, menghemat waktu, dan membuat pengembangan perangkat lunak menjadi lebih efisien. Selain itu, Design Pattern tidak mengharuskan developer untuk mengikuti aturan yang kaku, tetapi memberikan panduan dan prinsip yang dapat diadaptasi sesuai kebutuhan pengembangan perangkat lunak [22]. Berikut ini adalah beberapa manfaat dari *design pattern*:

1. Pola desain ini memberikan solusi atas masalah yang timbul saat pengembangan perangkat lunak.
2. Penulisan kode menjadi lebih rapi, terstruktur, dan lebih mudah dibaca.
3. Membuat komunikasi antara tim pengembang menjadi lebih efisien.

### 2.2.12 MVVM (Model View View-Model)

Model View ViewModel (MVVM) merupakan sebuah design pattern yang memiliki tiga komponen, yaitu Model, View, dan ViewModel [23]. MVVM adalah salah satu arsitektur pembuatan aplikasi berbasis GUI yang berfokus pada pemisahan antara kode untuk logika bisnis dan tampilan aplikasi. MVVM juga merupakan sebuah design pattern yang direkomendasikan oleh google untuk mengembangkan aplikasi android. [24]. Google sebagai kiblat untuk menerapkan design pattern tersebut sudah banyak membuat API dan dokumentasi khusus untuk membantu developer ketika ingin menerapkannya. Berikut adalah penjelasan ketiga komponen tersebut.

1. Model merupakan representasi dari data yang digunakan pada business logic. Model dapat berupa Plain Old Java Object (POJO) atau Kotlin Data Classes
2. View adalah komponen yang terdiri dari layout resource file dan Activity/Fragment. Tampilan pada layout resource file akan dikontrol melalui Activity/Fragment secara dinamis
3. ViewModel akan berinteraksi dengan Model dan menyiapkan observables variable yang akan diobservasi oleh View. ViewModel bersifat lifecycle-aware yang dapat dilihat pada Gambar 2.7 yang mana kelas ini akan hidup ketika sebuah kelas View telah melalui tahapan create dan belum melalui tahapan destroy. Pada kelas View, pemanggilan data hanya dilakukan satu kali dan data akan dipertahankan di ViewModel Scope selama kelas View belum melalui tahapan destroy. Jika kelas View telah melewati tahapan destroy, maka data yang ada di ViewModel akan dibersihkan




Gambar 2.7 Siklus Hidup ViewModel

### 2.2.13 Kotlin

Kotlin adalah bahasa pemrograman open source berjenis statis yang mendukung pemrograman berorientasi objek dan fungsional. Kotlin memberikan sintaksis dan konsep serupa dari bahasa lain, di antaranya termasuk C#, Java, dan Scala. Kotlin tidak dimaksudkan agar unik, melainkan mengambil inspirasi dari perkembangan bahasa selama puluhan tahun. Kotlin ada dalam varian yang menargetkan JVM (Kotlin/JVM), JavaScript (Kotlin/JS), dan kode native (Kotlin/Native). Kotlin dikelola oleh Kotlin Foundation, sebuah grup yang didirikan oleh JetBrains dan Google, yang ditugaskan untuk mengembangkan dan

melanjutkan pengembangan bahasa. Kotlin secara resmi didukung oleh Google untuk pengembangan Android. Artinya, dokumentasi dan alat Android didesain dengan mempertimbangkan Kotlin [25].

Kemunculan bahasa Kotlin dimulai ketika salah satu *Lead Developer* dari JetBrains yaitu Dmitry Jemerov tidak menemukan beberapa fitur pada bahasa Java. Dari situlah ia ingin sebuah bahasa yang di dalamnya terdapat fitur-fitur dari bahasa modern, dan bahasa tersebut dapat berjalan di JVM. Tidak hanya itu, ia juga ingin bahasa tersebut memiliki kecepatan kompilasi yang sama seperti Java. Sehingga muncullah bahasa Kotlin yang ia ciptakan sendiri. Kemudian nama Kotlin sendiri diambil dari nama sebuah pulau dari markas JetBrains yang lokasinya terdapat di Saint. Petersburg, Rusia. Bahasa Kotlin menjadi lebih populer setelah Google mengumumkan bahwa Kotlin adalah bahasa yang secara resmi dapat menjadi alternatif dalam pembuatan aplikasi Android. Sejak saat itulah para *developer*, khususnya *Android Developer* semakin tertarik menggunakan bahasa ini [26].

### 2.2.14 Maintainability Index

Maintainability Index merupakan sebuah metrik yang mengukur perangkat lunak untuk mengembangkan perangkat lunak tersebut. Maintainability Indeks (MI) memudahkan pengembangan jika kode sumbernya hendak dirubah. Maintainability Index mengalkulasi formula berdasarkan Lines of Code (LOC), Cyclomatic Complexity (CC) dan Halstead Volume (HV). Berikut adalah rumus untuk menentukan *Maintainability Index* [27].

$$MI = 171 - 5.2 \times \ln(HV) - 0.23 \times CC - 16.2 \times \ln(LOC) + ( 50 \times \sin(\sqrt{2.46 \times perCM}))$$

Keterangan:

HV = Halstead Metrics Volumes

CC = Cyclomatic Complexity

LOC = Lines of Code

PerCm = Percent Line of Comment

Pada rumus tersebut, diketahui bahwa maksimum index dari maintainability index adalah 171 yang merupakan nilai awal pada perhitungan maintainability index. Adapun klasifikasi metrik untuk analisis *maintainability* dapat dilihat pada table 2.4.

Tabel 2.4 Klasifikasi Maintainability Index

| Nilai Maintainability Index | Klasifikasi |
|---|---|
| *MI > 85* | *Highly Maintainable* |
| *MI 65 ≤ 85* | *Moderately Maintainable* |
| *MI < 65* | *Difficult to Maintain* |

### 2.2.15 *Reusability Metric*

Terdapat beberapa metrik berorientasi objek yang populer dan membahas bagaimana memprediksi reusability, Paremeter metrik yang digunakan adalah dengan menggunakan metrik dari Chidamber dan Kemerer. Metrik *reusability* pada metrik Chidamber dan Kemerer adalah WMC, LOC, CBO, DIT, dan NOC [7].

Dalam masing-masing dari tiga sistem, analisis regresi stepwise menunjukkan CBO dan LCOM sebagai variabel penjelas yang signifikan. Penting untuk ditekankan kekuatan hasil ini mengingat masing-masing model ini memiliki variabel dependen yang berbeda, yaitu produktivitas untuk TPM, upaya pengerjaan ulang untuk FIS, dan upaya desain untuk SLB. Menarik untuk dicatat bahwa hasil ini konsisten dengan hasil awal yang ditemukan di tempat lain, dan mungkin mencerminkan kekuatan konsep dasar coupling dan cohesion [7].

*Coupling Between Object* (CBO) adalah adalah metric C&K yang mengukur seberapa kuat ketergantungan antara objek-objek dalam sebuah sistem. CBO menghitung jumlah kelas lain yang digunakan oleh sebuah kelas. Semakin tinggi nilai CBO, maka semakin kuat ketergantungan antara objek-objek dan semakin sulit untuk mengubah atau mengganti salah satu kelas tanpa mempengaruhi kelas lain [28].

LCOM adalah metric C&K yang paling kontroversial dan diperdebatkan. Dalam inkarnasi aslinya, C&K mendefinisikan LCOM berdasarkan jumlah pasangan metode yang berbagi referensi ke variabel instance. Setiap kombinasi pasangan metode dalam kelas dievaluasi. Jika pasangan tidak berbagi referensi ke variabel instance apa pun, maka hitungan meningkat 1 dan jika mereka berbagi variabel instance apa pun, maka hitungan berkurang 1. LCOM dianggap sebagai ukuran seberapa baik metode kelas bekerja sama untuk mencapai tujuan kelas. Nilai LCOM yang rendah menunjukkan bahwa kelas lebih koheren dan dianggap lebih baik [28]. Untuk menganalisis LCOM dibutuhkan dua hal yaitu menghitung jumlah metode dan jumlah koneksi lalu didapatkan rumus LCOM sebagai berikut :

LCOM = Jumlah Metode - Jumlah Koneksi

Hasil untuk CBO dan LCOM konsisten dengan hasil penelitian dalam *software engineering* yang menunjukkan bahwa *coupling* yang tinggi dan *cohesion* yang rendah akan merugikan produktivitas pengembangan. Adapun klasifikasi metrik untuk analisis *reusability* dapat dilihat pada Tabel 2.5

Tabel 2.5 Klasifikasi *Reusability Metric*

| Metrik | Good | Medium | Bad |
|---|---|---|---|
| CBO | $x < 14$ | $14 \leq x \leq 150$ | $150 < x$ |
| LCOM | $x = 0$ | $0 \leq x \leq$ Jumlah Atribut dalam Kelas | x > Jumlah Atribut dalam Kelas |

### 2.2.16 Use Case Diagram

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah “apa” yang diperbuat sistem, dan bukan “bagaimana”. Sebuah use case merepresentasikan sebuah interaksi antara aktor dengan sistem. Use case merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-create sebuah daftar belanja, dan sebagainya [29].

Use case diagram adalah alat visual yang sangat penting dalam pengembangan perangkat lunak. Tujuan utamanya adalah menggambarkan fungsionalitas sistem dengan jelas, menunjukkan bagaimana pengguna (aktor) berinteraksi dengan sistem untuk mencapai tujuan tertentu. Dengan demikian, use

case diagram membantu menangkap persyaratan pengguna secara efektif, memastikan bahwa sistem yang dibangun memenuhi kebutuhan mereka. Selain itu, diagram ini berfungsi sebagai alat komunikasi yang efektif antara pengembang, pengguna, dan pemangku kepentingan lainnya, memastikan semua pihak memiliki pemahaman yang sama tentang cara kerja sistem. Lebih jauh lagi, use case diagram berperan dalam perencanaan dan desain sistem, membantu mengidentifikasi komponen, interaksi, dan alur kerja yang diperlukan. Terakhir, diagram ini juga berguna dalam validasi dan verifikasi, memastikan sistem yang dikembangkan sesuai dengan persyaratan yang ditetapkan. Secara keseluruhan, use case diagram adalah alat yang sangat berharga dalam memastikan keberhasilan pengembangan perangkat lunak.

### 2.2.17 Class Diagram

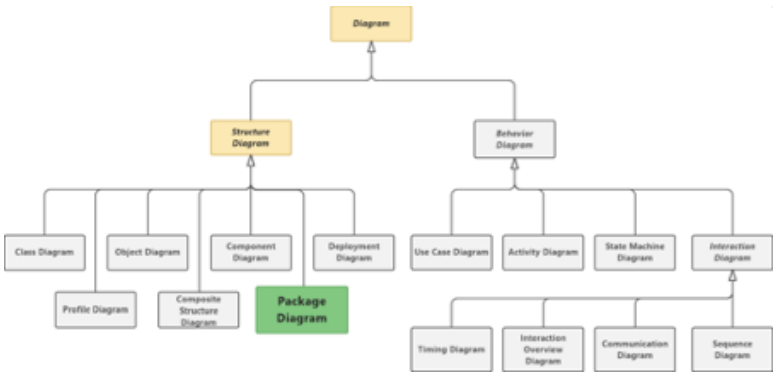
Class Diagram merupakan setiap objek atau data yang memiliki anggota, baik itu attributes (field dan properties), operations (methods), dan events. melalui class diagram nantinya program design didesain di tahapan perancangan sistem. class diagram merupakan model structural [30].

Class diagram digunakan untuk melakukan visualisasi struktur kelas-kelas dari suatu sistem dan merupakan tipe diagram yang paling banyak digunakan [31]. Class diagram juga dapat memperlihatkan hubungan antar kelas dan penjelasan detail tiap-tiap kelas di dalam model desain (logical view) dari suatu sistem. Selama proses desain, class diagram berperan dalam menangkap struktur dari semua kelas yang membentuk arsitektur sistem yang dibuat.

### 2.2.18 Package Diagram

*Package Diagram* merupakan salah satu Structure UML Diagram yang menggambarkan "paket" kelas, use case, atau komponen sistem lainnya dan disertai dengan keterangan ketergantungan kelas satu dengan kelas lainnya [32]. Tujuan utama penggunaan Package Diagram yakni adalah memberikan overview sekumpulan kebutuhan dan desain arsitektur dari sebuah sistem yang

memiliki hubungan logis dalam diagram modularnya (memecah sistem menjadi bagian kecil).




Gambar 2.8 Kedudukan Package Diagram dalam UML

*Package Diagram* jarang digunakan di dalam sistem yang berukuran kecil karena memang dari awalnya, sistem berukuran kecil tidak memiliki kelas atau komponen yang banyak, sehingga tidak perlu "dipaketkan" dengan Package Diagram. Package biasanya juga sering bergabung dengan package lain agar fungsi kedua package berjalan dengan lebih efisien; Operasi ini disebut sebagai Package Merge. Penggabungan paket adalah hubungan terarah antara dua paket yang menunjukkan bahwa konten suatu paket diperpanjang oleh konten paket lain. Mekanisme ini digunakan ketika elemen di sebuah package memiliki nama yang sama dengan package dan mewakili konsep yang sama (2 package dengan nama yang sama). Package Merge sering digunakan untuk memberikan definisi yang berbeda dari konsep yang diberikan untuk tujuan yang berbeda, dimulai dari definisi dasar yang sama.