Framework dan *Code Generator* Pengembangan Aplikasi Android dengan Menerapkan Prinsip *Clean Architecture*

(*Framework and Code Generator for Android Development with Clean Architecture Principles Implementation*)

Aflah Taqiu Sondha[1], Umi Sa'adah[2], Fadilah Fahrul Hardiansyah[3], Maulidan Bagus Afridian Rasyid[4]

***Abstract*— Android is one of smartphone operating systems that has highest market share in Indonesia. Due to its high market share, Android developers must develop Android applications faster and produce maintainable code. Unfortunately, the existing Android development system is not effective because of its dependency on the developer's experiences and pieces of knowledge that differ from each other. Therefore, there must be a new Android development model to produce maintainable code that implements clean architecture principles code with shorter time development. This system produces a code generator in Android Studio's template plugin that will generate a framework of the Android project with MVP architecture and implements clean architecture inside that framework. This generated framework is also directly integrated with an Android library dependency that contains common functions that are frequently used by Android developers. Testing result shows that this system saves 42% of Android application time development and generates code that has an 81% maintainability level.**

***Intisari*—Android merupakan salah satu sistem operasi *smartphone* yang memiliki pangsa pasar terbesar di Indonesia. Tingginya pangsa pasar Android tersebut memaksa pengembang aplikasi Android menghasilkan aplikasi Android dengan kualitas kode yang baik dan cepat. Akan tetapi, sistem pengembangan aplikasi Android yang sudah ada saat ini kurang efektif karena sangat tergantung pada pengetahuan dan pengalaman yang berbeda-beda antar para pengembang aplikasi Android. Oleh karena itu, diperlukan sebuah pengembangan pemodelan baru untuk mempercepat dan memperingan beban dalam pengembangan aplikasi Android, dengan menghasilkan kode berkualitas baik yang menerapkan prinsip *clean architecture*. Sistem ini menyediakan sebuah *code generator* dalam bentuk *plugin template* di Android Studio yang menghasilkan sebuah *framework* proyek Android dengan menggunakan arsitektur MVP dan menerapkan prinsip-prinsip *clean architecture* di dalamnya. *Framework* tersebut sudah terintegrasi langsung dengan sebuah dependensi *library* yang berisi fungsi-fungsi umum yang sering digunakan dalam pengembangan aplikasi Android. Hasil pengujian menunjukkan bahwa sistem ini mampu menghemat 42% waktu dan beban dalam pengembangan aplikasi Android serta menghasilkan kode dengan tingkat *maintainability* sebesar 81%.**

***Kata Kunci*—Android, *Framework*, *Code Generator*, *Clean Architecture*, Arsitektur MVP, *Maintainability*.**

[1,2,3] *Program Studi D4 Teknik Informatika, Departemen Teknik Informatika dan Komputer Politeknik Elektronika Negeri Surabaya, Jl. Raya ITS, Sukolilo, Surabaya, Jawa Timur 60117, INDONESIA (telp: 031-5947280; fax: 031-5946114; e-mail: Aflahts@it.student.pens.ac.id, umi@pens.ac.id, fahrul@pens.ac.id)*

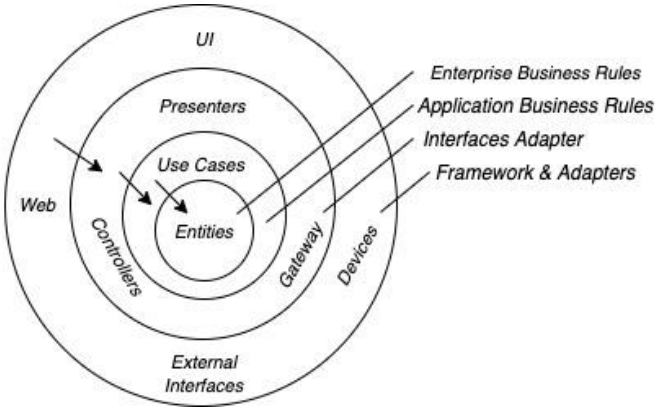
[4] *Research and Development Depertment, PT. Maulidan Teknologi Kreatif, Jl. Klampis Anom VIII No. F-150, Klampis Ngasem, Sukolilo, Surabaya 60117, INDONESIA (telp: 08122224298; e-mail: contact@maulidangames.com)*

I. PENDAHULUAN

Android merupakan sistem operasi *smartphone* yang memiliki pangsa pasar terbesar di Indonesia. Pangsa pasar Android mencapai 93,32% pada Mei 2019 [1]. Di Google Play Store, terdapat 2.978.362 aplikasi berbasis Android [2]. Tingginya pangsa pasar Android di Indonesia dan jumlah aplikasi berbasis Android di Google Play Store membuktikan bahwa kebutuhan Indonesia akan pengembang Android sangat besar, bahkan profesi pengembang Android merupakan salah satu profesi yang paling dicari oleh perusahaan-perusahaan di Indonesia pada tahun 2019 [3].

Variabel waktu dalam pengembangan aplikasi Android menjadi salah satu pertimbangan klien dan merupakan tuntutan pasar saat ini [4]. Suatu aplikasi yang sudah dirilis akan selalu berkembang mengikuti permintaan pasar, teknologi, persaingan bisnis, dan kepentingan-kepentingan organisasi [5]. Hal ini menuntut perusahaan untuk terus selalu melakukan eksplorasi terhadap aplikasi yang dimiliki agar tidak kalah bersaing dengan kompetitornya. Hal ini didukung oleh survei mengenai pengembangan perangkat lunak yang menemukan bahwa 75-90% biaya dalam pengembangan aplikasi dibutuhkan untuk proses *maintenance* [6]. Pengembangan suatu aplikasi oleh sebuah perusahaan tidak berhenti ketika aplikasi tersebut dirilis. Akan tetapi, aplikasi tersebut harus terus dikembangkan agar tetap berguna dan memenuhi kebutuhan pasar yang terus berkembang dengan cepat [7]. Perubahan dan perkembangan aplikasi yang dilakukan oleh suatu perusahaan menuntut suatu perusahaan untuk mengembangkan aplikasi yang mudah dan adaptif untuk dikembangkan dan dirawat di kemudian hari.

Maintainability dari suatu aplikasi adalah tingkat dari aplikasi tersebut untuk dapat dipahami, diperbaiki, atau dikembangkan di kemudian hari untuk menjaga aplikasi tersebut tetap berjalan dengan baik. *Maintenance* merupakan sebuah aktivitas yang membutuhkan banyak waktu, uang, dan sumber daya manusia sebagaimana dalam pengembangan aplikasi, yang bertujuan untuk menjaga dan memastikan sebuah aplikasi tetap berjalan dengan baik dalam waktu yang lama. Meningkatkan kualitas kode yang lebih bersih dan rapi dapat meningkatkan tingkat *maintainability* dari sebuah aplikasi [8].



Gbr. 1 *Layer clean architecture.*

Dalam pengembangan aplikasi Android, kualitas kode adalah salah satu variabel yang sangat penting. Perusahaan selalu meminta aplikasi yang memiliki kualitas baik untuk dapat dikembangkan di kemudian hari sehingga diperlukan sumber kode yang *maintainable*. Selain itu, perusahaan juga membutuhkan pengembangan aplikasi Android dengan cepat sehingga variabel beban dan waktu sangat memengaruhi proses pengembangan aplikasi Android bagi para pengembang Android. Pengembang Android dituntut untuk mengembangkan aplikasi Android dengan cepat dan menghasilkan sumber kode yang *maintainable*.

Sistem pengembangan aplikasi Android yang sudah ada sangat bergantung pada pengetahuan dan pengalaman pengembang dalam mengembangkan aplikasi Android. Hal ini membuat pengembangan aplikasi Android tidak efektif karena terlalu banyaknya permintaan pengembangan aplikasi Android terhadap pengembang yang jumlahnya terbatas dan menghasilkan kode dengan kualitas yang berbeda-beda.

II. METODOLOGI

Penerapan prinsip *clean architecture* dapat meningkatkan tingkat *maintainability* dari suatu aplikasi. Hal ini karena ada pemisahan antara komponen pada suatu aplikasi menjadi beberapa komponen yang independen dan lebih modular sehingga jika terjadi sebuah *bug* pada sebuah komponen, pengembang dapat fokus memperbaiki *bug* pada komponen tersebut tanpa memengaruhi komponen lainnya. Penambahan sebuah fitur pada aplikasi yang menerapkan prinsip *clean architecture* juga sangat mudah dilakukan.

Penerapan prinsip *clean architecture* dalam sebuah aplikasi dilakukan dengan memisahkan antar *layer* pada aplikasi tersebut. Sebuah aplikasi memiliki beberapa *layer*, di antaranya *entities, use cases, interface adapters*, dan *framework and drivers*, seperti ditunjukkan pada Gbr. 1. Masing-masing *layer* disebut dengan *single actionable idea* [9].

Interaksi antar *single actionable idea* harus berurutan, tidak boleh melewati salah satu *single actionable idea*. Prinsip *dependency inversion* digunakan dalam komunikasi dan pengiriman data antar *single actionable idea* [9]. Prinsip *dependency inversion* dapat mengatasi permasalahan ketergantungan antar komponen dalam aplikasi. Pengiriman data pada masing-masing *single actionable idea* haruslah berupa struktur data sederhana atau berupa tipe data primitif. Pengiriman data berupa objek antar *single actionable idea* akan melanggar *dependency rule* dalam pengembangan aplikasi [9].

Framework yang dihasilkan pada makalah ini menerapkan prinsip-prinsip *clean architecture*. Penerapan prinsip *clean architecture* tersebut meliputi:

- memisahkan masing-masing *layer* dalam sebuah proyek Android,
- masing-masing *layer* dihubungkan dengan abstraksi *interface* sehingga tidak akan membuat suatu komponen tergantung dengan komponen lain, dan
- interaksi antar masing-masing *single actionable idea* hanya dengan menggunakan data dengan tipe data primitif.

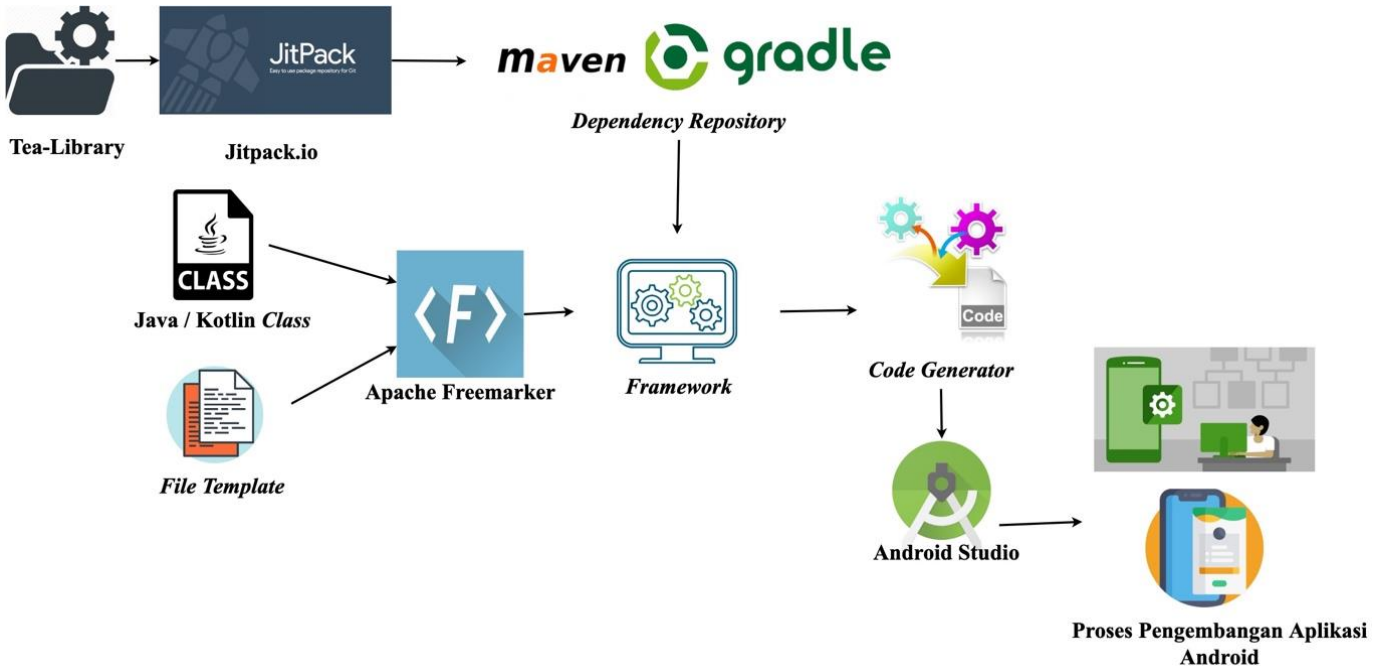
Dalam pengembangan aplikasi Android, terdapat beberapa arsitektur, di antaranya arsitektur (Model-View-Controller) MVC, *Model-View-Presenter* (MVP), dan (Model-View-ViewModel) MVVM. Masing-masing arsitektur tersebut memiliki kekurangan dan kelebihan dalam tahap pengembangan maupun pengujian aplikasi. Arsitektur MVP dan MVVM merupakan aplikasi yang lebih unggul daripada arsitektur MVC. Penilaian ini berdasarkan aspek *testability, modifiability,* dan kinerja aplikasi yang dikembangkan dengan masing-masing arsitektur tersebut [10], [11].

Sistem yang dikembangkan dalam makalah ini memiliki tiga komponen utama yang saling terintegrasi. Komponen pertama adalah sebuah dependensi *library* Android, komponen kedua adalah *file template generator* yang terintegrasi dengan IDE Android Studio yang menghasilkan susunan baris-baris kode dalam sebuah atau beberapa *class*, dan komponen ketiga adalah sebuah *framework* atau kerangka kerja dalam pengembangan aplikasi Android. *Framework* ini merupakan hasil *generate* dari *file template generator*. Di dalam *framework* tersebut sudah terdapat baris-baris kode yang dapat digunakan oleh pengembang. Dependensi *library* Android akan langsung ditambahkan pada *framework* tersebut secara otomatis.

Proses pengembangan ini diawali dengan membuat sebuah dependensi *library* Android. *Library* tersebut dimuat pada sebuah *dependency publisher*. Dependensi tersebut dapat diimplementasikan pada sebuah proyek Android dalam bentuk Maven maupun Gradle. Proses selanjutnya adalah membuat *template file* yang merupakan abstraksi dari *class* beserta *file* yang berfungsi untuk integrasi dengan IDE Android Studio. *Template file* tersebut terintegrasi secara langsung untuk memuat dependensi *library* yang telah dibuat sebelumnya. Kedua komponen tersebut kemudian digabung ke dalam sebuah *framework* Apache. Kemudian hasil dari proses tersebut diintegrasikan ke dalam IDE Android Studio sebagai *plugin template*.

Implementasi dari abstraksi *template file* dari *plugin generator* IDE Android Studio tersebut menerapkan prinsip-prinsip *clean architecture* untuk kemudian dapat dikembangkan untuk menghasilkan aplikasi Android yang *maintainable.* Detail alur proses penelitian ini diilustrasikan pada Gbr. 2.

Dependensi *library* Android yang dikembangkan merupakan komponen dalam pemrograman yang memiliki fungsi-



Gbr. 2 Alur proses penelitian.

fungsi tertentu yang sering dibutuhkan pengembang dalam mengembangkan aplikasi. Pembuatan *library* dimaksudkan agar aplikasi dibuat dengan kode yang efektif dan efisien serta *clean code*. Pembuatan *library* pemrograman ini bertujuan agar pengembang tidak perlu menuliskan baris-baris kode yang sama berulang-ulang sehingga akan mempercepat waktu pengembangan suatu aplikasi dan menghindari *duplicate code*. Penulisan kode yang berulang-ulang akan melanggar aturan pengulangan pada *software design* [12].

Library Android yang dikembangkan memiliki beberapa *class* yang memiliki fungsionalitas berbeda-beda. Masing-masing *class* tersebut adalah sebagai berikut.

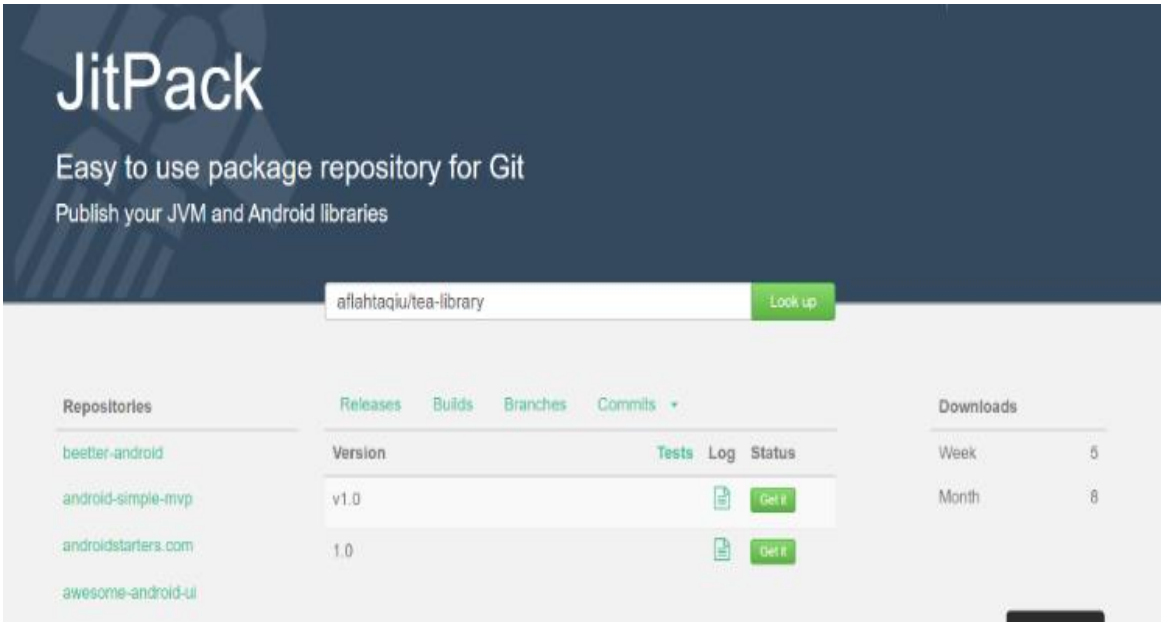
- *Class* SharedPrefUtils yang berfungsi untuk mengatur dan mempermudah pengembang dalam mengakses *shared preference* dalam Android.
- *Class* ComuunicationUtils yang berisikan fungsi-fungsi untuk mempermudah pengembang berpindah dari suatu komponen tampilan ke komponen tampilan lain dalam Android.
- *Class* LoadingUtils yang bertugas untuk menampilkan pesan tunggu, *progress*, dan *loading* dalam aplikasi Android.
- *Class* MessageUtils yang berguna untuk menampilkan notifikasi pesan pada sebuah aplikasi Android.
- *Class* NetworkUtils dan *class* ApplicationUtils yang berguna untuk mempermudah menampilkan informasi-informasi penting mengenai aplikasi secara umum dan informasi jaringan yang ditangkap oleh perangkat Android.
- *Class* DateTimeUtils yang berisikan beberapa *regex* dalam konversi *DateTime* ke dalam *String* sehingga mempermudah pengembang dalam menampilkan informasi tanggal dan waktu dalam aplikasi Android.

Makalah ini menggunakan *publisher jitpack.io*, yang merupakan sebuah tempat untuk memuat atau *publisher* dependensi-dependensi dari *library-library* JVM maupun Android. *Jitpack.io* mengintegrasikan dirinya secara langsung dengan sebuah versi rilis sebuah *library* yang dipublikasikan oleh pengembang di dalam *repository*, baik dari Github, Gitlab, maupun Bitbucket. Setelah menambahkan beberapa *class* untuk *library* Android pada modul *library* Android, *library* dikirimkan ke sebuah *repository* Github dan kemudian ditambahkan versi rilis dari *repository* modul tersebut. Versi rilis dari *repository* Github tersebut akan dibaca dan dikenali oleh *publisher jitpack.io*, seperti ditunjukkan pada Gbr. 3.

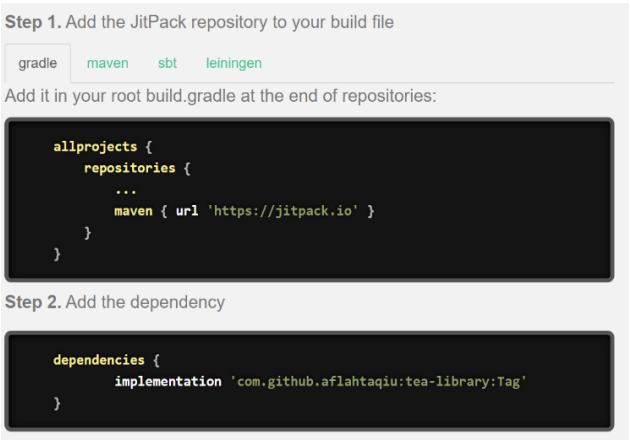
Keseluruhan fungsi yang terdapat dalam dependensi *library* Android ini menggunakan kata kunci *static*. Pengembang dapat langsung memanggil fungsi dalam pemanggilan *class* tanpa membuat objek dari *class* tersebut terlebih dahulu. Hal ini dimaksudkan karena hal-hal yang terdapat dalam *static method* adalah sesuatu yang bersifat konstan dan tidak akan berubah-ubah.

Pengembang Android dapat menggunakan dependensi *library* Android tersebut dalam sebuah pengembangan aplikasi dengan menambahkan dependensi Maven *Jitpack.io* pada dependensi *root* dan dependensi Tea-Library pada dependensi *app* sesuai dengan versi rilisnya, seperti pada Gbr. 4. Pada sistem ini, dependensi *library* Tea-Library akan langsung ditambahkan pada *framework* yang dihasilkan oleh *code generator*.

Dalam makalah ini, sistem menghasilkan sebuah *framework* melalui *code generator* yang terintegrasi dengan IDE Android Studio. *Code generator* tersebut menghasilkan beberapa *class* abstraksi maupun beberapa *class* yang menyusun sebuah modul. Integrasi antar IDE Android Studio dengan sistem ini menggunakan *file* FreeMarker Template Language (FTL)



Gbr. 3 Versi rilis *library* pada *jitpack.io*.



Gbr. 4 Penambahan dependensi *library* pada proyek Android.

dengan jenis *file .ftl*. *File* ini akan mengubah abstraksi *class* menjadi sebuah *class* Java maupun Kotlin. FTL merupakan sistem yang dikembangkan menggunakan Java dan bersifat *open source.* FTL ini bisa mengubah sebuah *template file* menjadi sebuah *class* dalam bahasa pemrograman tertentu, seperti PHP, Java, dan Kotlin.

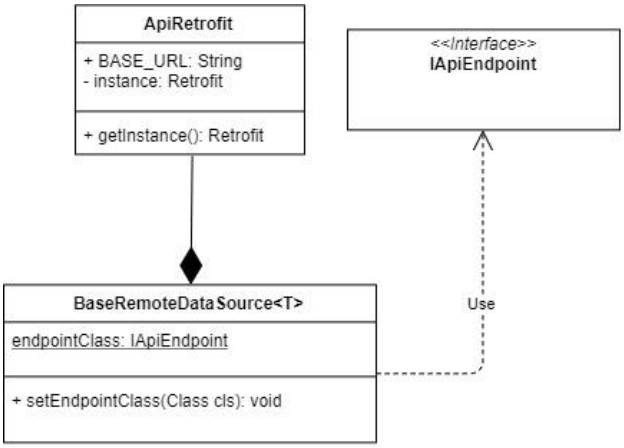
File FTL tersebut secara otomatis terbaca oleh *plugin file template generator* di IDE Android Studio untuk menghasilkan *class* Java maupun Kotlin. *Class* tersebut akan berisi baris kode abstraksi dan beberapa parameter yang disesuaikan dengan kebutuhan pengembang. Ketika pengembang ingin meng-*generate* sebuah *base project* maupun sebuah modul, akan muncul sebuah tampilan untuk mengisi parameter-parameter yang digunakan sesuai kebutuhan pengembang.

Framework akan menghasilkan dua komponen utama, yaitu komponen *base project* dan komponen per modul. Kedua komponen tersebut sudah menerapkan prinsip-prinsip *clean architecture*. Masing-masing *class* pada komponen-komponen tersebut memiliki tingkat ketergantungan antar *class* yang rendah. Ketika pengembang menambahkan sebuah modul baru dalam pengembangan aplikasi Android, *generator* membuat-kan sebuah komponen *class* modul yang memiliki arsitektur MVP.

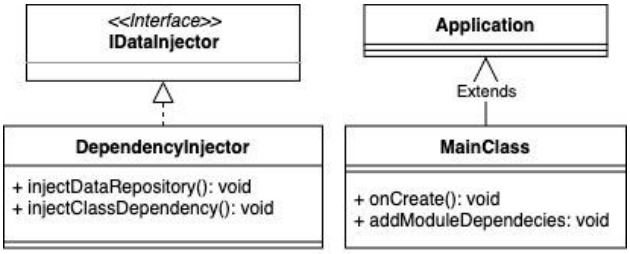
Komponen *base project* terdiri atas kelas-kelas dasar yang menjadi abstraksi *class* pada proyek aplikasi Android tersebut. Ketika terdapat penambahan modul, modul baru tersebut akan terintegrasi dengan *base project* yang sudah dihasilkan sebelumnya. Selain itu, *base project* terdiri atas beberapa *class* yang memiliki fungsi umum yang dapat diturunkan pada *class* lain. Komponen *base project* juga berisi beberapa *class* yang akan menjadi *global class* dalam mengatur pengambilan data dari server, mengatur dependensi *injection* serta sebuah *class* yang menjadi *class* utama yang akan dijalankan pertama kali pada sebuah aplikasi. Pada komponen *base project* secara otomatis ditambahkan dependensi *library* yang telah dibuat pada makalah ini.

Terdapat tiga bagian pada komponen *base project*. Bagian pertama adalah *base class* yang merupakan beberapa *class* yang menjadi *superclass* yang dapat diturunkan oleh beberapa *subclass*. *Base class* yang merupakan *superclass* ini memiliki beberapa fungsi atau karakter-karakter umum yang dapat diturunkan. Adanya *base class* ini dapat membantu penggolongan *class* berdasarkan karakteristiknya memiliki sifat yang sama dengan *class superclass* sehingga tidak ada *duplicate code* lagi. Pada makalah ini terdapat delapan *base class*: BaseActivity, BaseFragment, BaseRemoteDataSource, BaseResponse, IBaseCallback, IBasePresenter, IBaseView, dan ResponsePaging. Bagian kedua dari komponen ini adalah bagian pengaturan umum aliran data dari server atau *remote data source*. Bagian ini terdiri atas *class* ApiRetrofit dan *class* IApiEndpoint, seperti ditunjukkan pada Gbr. 5.

Kedua *class* tersebut berfungsi mengatur pengambilan data dari server. *Class* ApiRetrofit berisikan dengan *field* BASE_URL yang merupakan alamat *URL* dari API server serta



Gbr. 5 *Class diagram remote data source*.



Gbr. 6 *Class diagram* MainClass dan DependencyInjector.

terdapat sebuah inisiasi objek Retrofit untuk menghubungkan antara aplikasi dengan server. Inisiasi objek Retrofit ini dibuat menggunakan *Singleton Pattern.* Hal ini dimaksudkan karena objek Retrofit yang ada pada *class* ApiRetrofit digunakan di beberapa *class* yang berbeda. *Interface* IApiEndpoint berisi kumpulan-kumpulan abstraksi fungsi yang berisikan *endpoint*, parameter yang diminta, dan nilai hasil dalam pemrosesan data di server. Kedua *class* ini akan berinteraksi dengan *class* BaseRemoteDataSource yang nantinya akan dituruni oleh beberapa *class* yang berfungsi memanipulasi pemrosesan data dari server.

Bagian ketiga dari komponen ini terdiri atas *class* MainClass dan *class* DataInjector yang berfungsi mengatur dependensi antar *class*, seperti ditunjukkan pada Gbr. 6. *Class* DependencyInjector berfungsi sebagai jembatan pemanggilan *class* DependencyInjector oleh *class* lainnya. Hal ini dimaksudkan untuk mengurangi tingkat *coupling* ketika sebuah *class* membutuhkan atau memanggil *class layer* data melalui *class* DependencyInjector. C*lass* MainClass berfungsi sebagai *class* awal yang akan dieksekusi ketika aplikasi pertama kali dijalankan. Pada *class* ini, *class* DependencyInjection akan dijalankan sehingga pada pertama kali aplikasi dijalankan, seluruh komponen akan terinjeksi satu dengan yang lainnya.

Komponen kedua adalah komponen per modul. Komponen ini berfungsi menambahkan modul-modul tertentu untuk sebuah fitur dalam aplikasi Android. Komponen ini memiliki dua bagian, yaitu bagian *BaseMVP* dan bagian modul *Repository Pattern.*

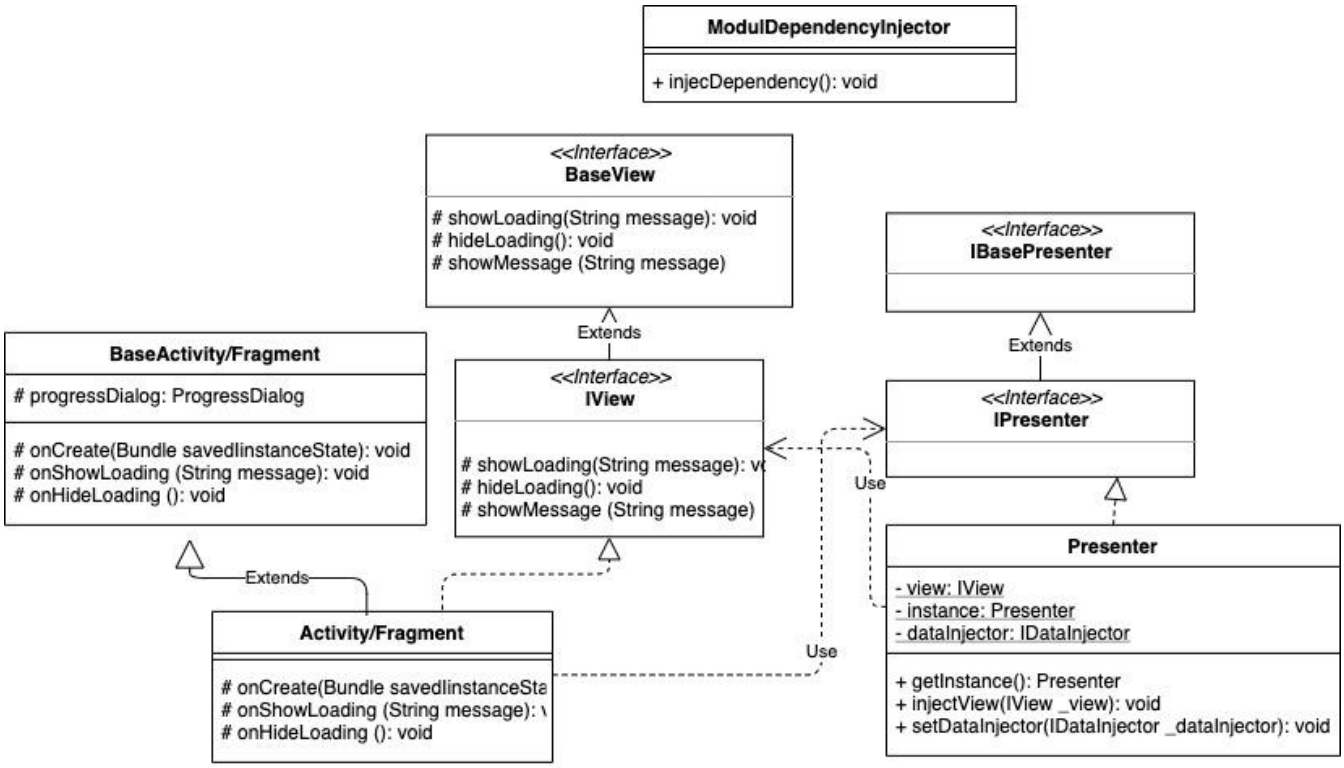
Bagian *BaseMVP* adalah modul dasar untuk menghasilkan sebuah tampilan atau fitur baru dalam Android. Pada komponen ini, pengembang dapat memilih untuk menambahkan modul dengan menggunakan *activity* maupun *fragment*, yang dibedakan dengan masing-masing *superclass*. Masing-masing *activity* dan *fragment* merupakan komponen utama dalam Android untuk interaksi antar pengguna dengan aplikasi Android. Keduanya memiliki *lifecycle* yang berbeda. Modul *BaseMVP* sudah mengimplementasi prinsip-prinsip *clean architecture*. Komponen *BaseMVP* ini menerapkan arsitektur MVP yang membedakan *layer view* dan layer *presenter*. Masing-masing *layer* dihubungkan dengan *interface.* Terdapat sebuah *class* yang berfungsi untuk menginjeksikan masing-masing *interface* seperti ditunjukkan pada Gbr. 7.

Bagian modul *Repository Pattern* yang ditunjukkan pada Gbr. 8 berfungsi untuk mengatur pengambilan data dari sumber data dan mendistribusikannya ke modul-modul aplikasi yang membutuhkannya. Bagian ini terdiri atas dua bagian utama, yaitu bagian untuk mengambil data dari sumber data dan bagian untuk mendistribusikan data tersebut dengan *callback*. Terdapat dua metode dalam pengambilan data pada *repository pattern*, yaitu dari server dan dari lokal. Pada proses pengambilan data yang berasal dari server, ada *class* RemoteDataSource yang akan menurunkan sifat dan karakteristik dari *class* BaseRemoteDataSource. *Class* RemoteDataSource berfungsi untuk mengambil data dari server.

Class RemoteDataSource akan mengimplementasi fungsi-fungsi abstraksi dari IDataSource yang berisikan fungsi-fungsi abstraksi mengenai hal-hal yang akan diambil dari sumber data server. Kemudian, *class* RemoteDataSource akan mengambil data-data sesuai yang diminta oleh IDataSource. Pada proses pengambilan data dari lokal *database* SQLite, ada *class* LocalDatabase yang berfungsi untuk membuat *database* SQLite. Terdapat *interface Data Access Object* (DAO) yang berisikan *query* pengolahan data dari *database* SQLite. Masing-masing entitas akan terdaftar ke dalam *database* lokal dan akan memiliki DAO sendiri. Terdapat *class* LocalDataSource yang mengimplementasikan *interface* IdataSource. *Class* LocalDataSource ini akan mengambil data dari *database* lokal sesuai dengan abstraksi fungsi-fungsi yang didaftarkan pada *interface* tersebut. Pengambilan data sendiri, yang dilakukan oleh *class* LocalDataSource, akan menggunakan masing-masing DAO dari model-model yang perlu diambil. Kedua sumber data tersebut digabungkan pada *class* DataRepository. *Class* ini juga mengimplementasikan fungsi-fungsi yang berasal dari abstraksi *interface* IDataSource. Setelah mengimplementasikan fungsi-fungsi tersebut, *class* DataRepository ini akan melakukan pengambilan data salah satu sumber saja dan akan mengirimkannya ke layer *presenter* dengan menggunakan *callback.*

III. Hasil dan Pembahasan

Terdapat dua hal yang diuji pada sistem yang dihasilkan dalam makalah ini, yaitu kecepatan dan beban pengembangan serta kualitas sumber kode yang *maintainable*. Skenario pengujian terhadap waktu dan beban pada sistem ini adalah dengan mengembangkan dua aplikasi yang identik. Kedua



Gbr. 7 *Class diagram BaseMVP*.

aplikasi tersebut masing-masing dikembangkan dengan menggunakan dan tidak menggunakan sistem pada makalah ini. Pengujian terhadap kualitas kode yang *maintainable* dilakukan dengan menghitung tingkat *maintainability* pada kode aplikasi Android yang dihasilkan oleh sistem pada makalah ini.

Aplikasi yang dikembangkan sebagai uji coba adalah aplikasi GoPMGo!. Aplikasi tersebut merupakan aplikasi pendeteksi *antipattern* pada sebuah tim pengembangan aplikasi. Terdapat dua fitur utama pada aplikasi tersebut. *Developer* maupun *project manager* dalam tim dapat mengisi sebuah kuesioner. *Developer* dan *project manager* dapat mengetahui *antipattern* berdasarkan kuesioner yang telah diisi dan terdapat rekomendasi *treatment* dari masing-masing *antipattern*. Aplikasi tersebut dikembangkan dari awal dan menggunakan Android *native* dengan bahasa pemrograman Java.

Pengujian terhadap waktu dan beban dilakukan dengan menghitung banyaknya *interaction cost*. *Interaction cost* adalah banyaknya interaksi pengembang dengan perangkat (*keyboard*, *mouse*, dan sebagainya) dalam melakukan suatu aksi [13], [14]. Besar *interaction cost* dalam pengembangan aplikasi menandakan besarnya waktu dan beban yang diperlukan dalam mengembangkan aplikasi Android. Sebagai contoh, pengembang yang menekan satu tombol *keyboard* atau menekan tombol *mouse* akan dihitung masing-masing sebagai satu *interaction cost.*

Kualitas kode yang dinilai pada pengujian ini menggunakan *McCall's Software Quality Model*, dengan *maintainability* merupakan salah satu *software quality metric* dalam matriks tersebut [15], [16]. Penilaian *quality metric* pada *McCall's Software Quality Model* dapat dihitung menggunakan (1) [16].

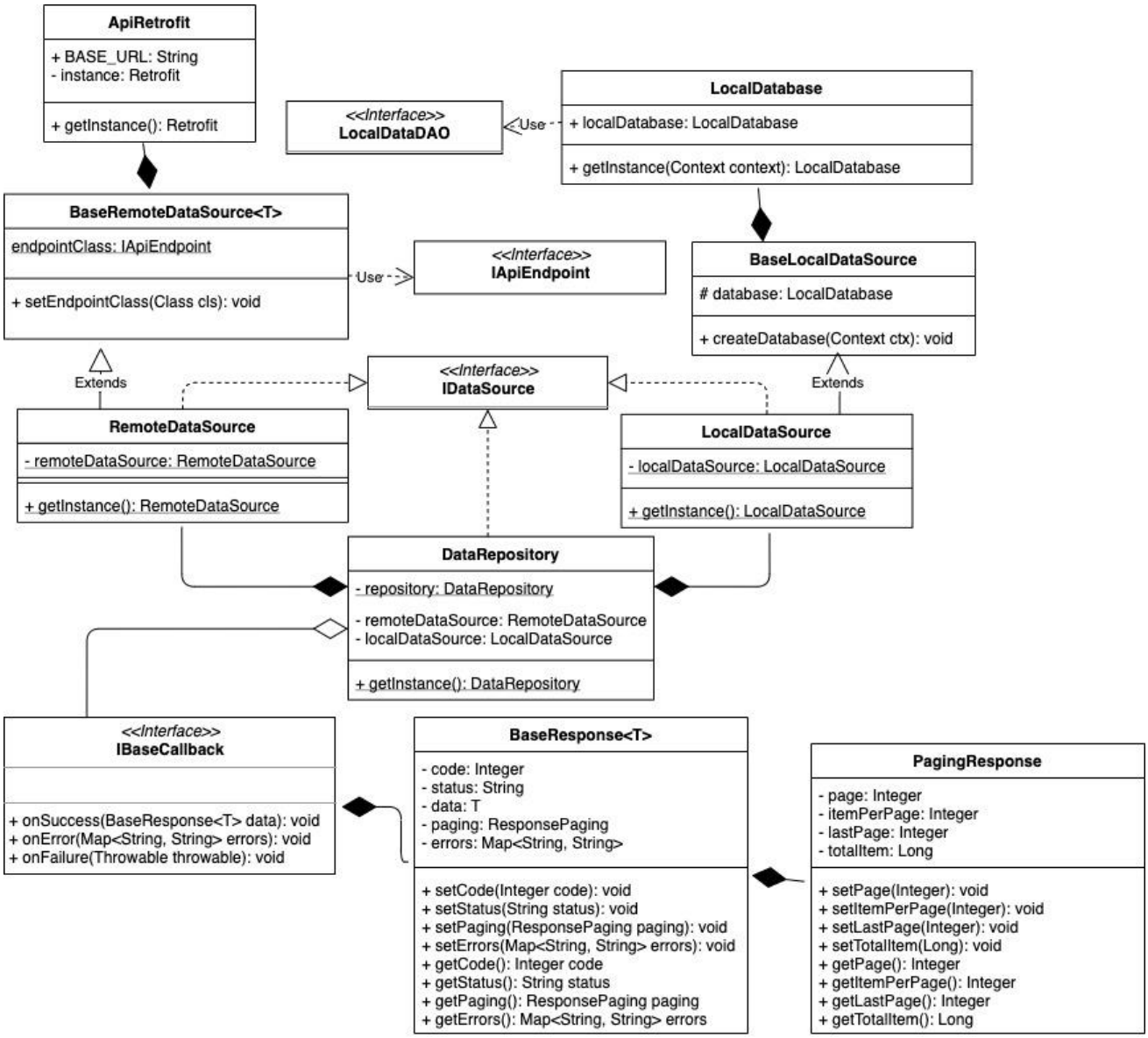
$$F = C_1.M_1 + C_2.M_2 + \cdots + C_n.M_n$$
$$C_1 + C_2 + \cdots + C_n = 1 \quad (1)$$
$$0 \leq M \leq 1 \;\; 0 \leq C \leq 1.$$

Quality Metric (*F*) dihitung dengan menjumlahkan seluruh perkalian antara *quality factor* (*M*) dengan masing-masing beban (*C*). Jika seluruh beban pada *quality metric* dijumlahkan, nilainya satu. Pada *McCall's Software Quality Model*, masing-masing *quality metric* memiliki *quality factor* tertentu yang menentukan nilai *quality metric*. Masing-masing nilai *quality factor* didapatkan dengan membandingkan *positive case* dan *test case* masing-masing *quality factor*.

Quality metric maintainability ditentukan oleh beberapa *quality factor*, yaitu *conciseness*, *consistency*, *instrumentation*, *modularity*, *self-documentation*, dan *simplicity* [16] Pada makalah ini, ditentukan beban masing-masing *quality factor* untuk menentukan nilai *maintainability* sama rata, yaitu sebesar 0,167.

Conciseness merupakan *quality factor* yang merepresentasikan keringkasan dan kepadatan program dalam baris kode. *Quality factor* ini membandingkan total *class* dengan *logical line of code* (LLOC) yang ada pada suatu aplikasi. LLOC merupakan baris kode dalam sebuah program yang tidak termasuk dalam komentar dan *blank line*. Baris kode tersebut harus berupa deklarasi atau baris kode yang berisi *executable statement* [16].

Modularity merupakan *quality factor* yang menghitung tingkat ketergantungan sebuah komponen terhadap komponen lainnya. Dalam menentukan sebuah *class* adalah *class* yang



Gbr. 8 *Class diagram* modul *Repository Pattern*.

coupling atau tidak, makalah ini menghitung nilai *coupling* dari masing-masing *class* dengan rumus *coupling* (2) [17]. Sebuah *class* dapat dikatakan *coupling* jika memiliki nilai *coupling* lebih dari 0,67.

$$C = 1 - \frac{1}{d_i + 2\,c_i + d_o + 2c_o + g_d + 2g_c + w + r} \quad (2)$$

dengan

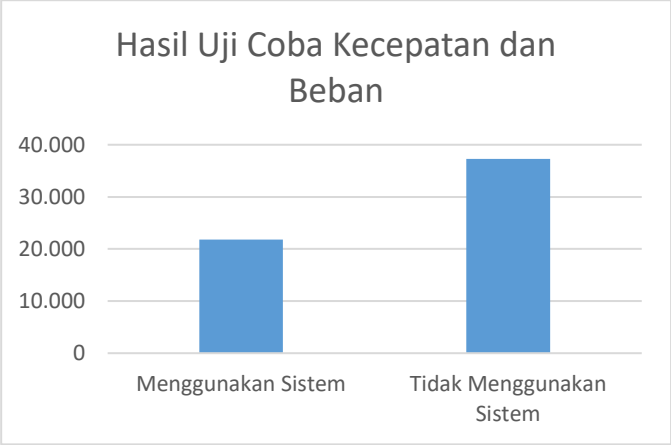
d_i = banyaknya parameter data masuk
c_i = banyaknya parameter kontrol masuk
d_0 = banyaknya parameter data keluar
c_o = banyaknya parameter kontrol keluar
g_d = banyaknya variabel global yang digunakan sebagai data
g_c = banyaknya variabel global yang digunakan sebagai kontrol
w = banyaknya modul terpanggil (*fan-out*)
r = banyaknya modul yang dipanggil ke dalam modul tersebut (*fan-in*).

Self-documentation adalah tingkatan sebuah *source code* menyediakan dokumentasi yang *meaningful*, sedangkan *simplicity* adalah tingkat kemudahan suatu program untuk dipahami. Masing-masing *self-documentation* dan *simplicity* akan menghitung nilai *clean code*. Nilai *clean code* yang dihitung adalah *naming convention*, *capitalization rule*, dan *comments*, berdasarkan aturan penulisan *clean code* [18]. Perbedaan keduanya adalah *self-documentation* menghitung *clean code* pada sebuah *class*, sedangkan *simplicity* menghitung *clean code* pada sebuah fungsi.

Uji coba pada makalah ini menggunakan *threshold* 95% atau 0,95 untuk menentukan sebuah aplikasi sangat *maintainable*. Penentuan nilai 95% atau 0,95 berdasarkan *rule of thumb*. Spesifikasi perangkat keras dan perangkat lunak yang digunakan ditunjukkan pada Tabel I.

TABEL I
SPESIFIKASI PERANGKAT KERAS DAN PERANGKAT LUNAK UNTUK UJI COBA

Jenis Spesifikasi	Spesifikasi yang Digunakan
Operating System	Linux Ubuntu 18.04
Processor	Intel Core i5-8250U 3.4 GHz
RAM	16 GB DDR4 2133MHz
HDD	HDD 1TB 5400 *rpm*
Graphic Card	Intel UHD Graphics 620 dan Nvidia GeForce GT 940MX VRAM 2GB GDDR3
Versi Android Studio	Android Studio 3.6



Gbr. 9 Hasil uji coba kecepatan dan beban pada sistem.

Setelah kedua aplikasi GoPMGo! yang identik dikembangkan menggunakan dua metode yang berbeda, hasil perhitungan *interaction cost*-nya ditunjukkan pada Gbr. 9. Masing-masing dihitung dengan perhitungan manual.

Berdasarkan hasil uji coba yang ditunjukkan pada Gbr. 9, pengembangan aplikasi dengan menggunakan sistem hanya membutuhkan 21.778 *interaction cost*, sedangkan pengembangan aplikasi tanpa menggunakan sistem membutuhkan 37.254 *interaction cost*. Pengembangan aplikasi Android dengan menggunakan sistem yang dibuat mampu menghemat 42% waktu dan beban pengembangan dari waktu dan beban saat pengembangan aplikasi dengan manual.

Pengembang aplikasi Android dapat terbantu dalam meningkatkan tingkat *maintainability* dari *source code* dengan menggunakan sistem pada makalah ini. Seperti yang ditunjukkan pada Tabel II, *source code* yang dikembangkan dengan bantuan sistem ini memiliki nilai *maintainability* sebesar 0,81 atau 81%. Sebelumnya sudah disebutkan bahwa nilai ambang batas yang digunakan sebesar 95%.

Nilai *maintainability* tersebut dapat ditingkatkan untuk memenuhi nilai ambang batas dengan melakukan *extract method* pada *source code*. Hal ini karena *quality factor conciseness* sangat rendah dan merepresentasikan *source code* yang kurang ringkas dan padat. Dengan melakukan *extract method*, *source code* akan menjadi lebih ringkas dan padat karena jumlah LLOC pada masing-masing *class* berkurang atau semakin padat. Selain itu, nilai *maintainability* dapat ditingkatkan dengan menerapkan *design pattern* pada *framework* di sistem ini.

TABEL II
HASIL PERHITUNGAN TINGKAT *MAINTAINABILITY*

Quality Factor	***Positive Case* dan *Test Case***	**Nilai**	**M**	**C**	**F**
Conciseness	Banyaknya *Class*	118	0,14	0,17	**0,81**
	LLOC	829			
Consistency	Fitur sesuai desain	4	1	0,17	
	Fitur yang didesain	4			
Instrumentation	Fitur punya instrumentasi	4	1	0,17	
	Fitur yang seharusnya punya instrumentasi	4			
Modularity	*Class* yang *loose coupling*	103	0,87	0,17	
	Banyaknya *Class*	118			
Self-Documentation	*Class* yang *clean code*	100	0,85	0,17	
	Banyaknya *Class*	118			
Simplicity	Fungsi yang *clean code*	385	0,93	0,17	
	Banyaknya fungsi	413			

Tingginya nilai *modularity* pada *source code* yang dikembangkan menggunakan sistem ini membuktikan bahwa sistem ini berhasil menerapkan prinsip *clean architecture*. Sistem berhasil memisahkan antar *layer* pada *source code*. Masing-masing *layer* berkomunikasi dengan menggunakan tipe data primitif sehingga tidak melanggar prinsip *dependency rule*.

IV. KESIMPULAN

Makalah ini berfokus pada pengembangan sistem pengembang aplikasi Android. Waktu, beban, dan kualitas kode yang dihasilkan adalah variabel-variabel yang sangat penting dalam pengembangan aplikasi Android. Klien membutuhkan kode dengan kualitas baik yang *maintainable* untuk dikembangkan di kemudian hari, selain waktu yang cepat dalam pengerjaannya. Pengembang membutuhkan efisiensi waktu dan beban dalam proses pengembangan, tetapi tidak mengurangi kualitas kode yang dihasilkan. Oleh karena itu, diajukan solusi sistem pengembangan aplikasi Android dengan pendekatan baru untuk menerapkan prinsip *clean architecture*.

Sistem pengembang ini terdiri atas *plugin template* yang dapat diintegrasikan dengan IDE Android Studio. *Plugin* tersebut akan menghasilkan *framework* dan langsung terintegrasi dengan *library* Android. *Framework* dan *library* tersebut berisikan beberapa baris kode yang umum digunakan oleh pengembang Android sehingga mengurangi *duplicate code* dan *boilerplate*. Pengembang dapat menambahkan modul sesuai kebutuhan dengan menggunakan *plugin* tersebut. Dalam makalah ini diuji waktu dan beban pengembangan aplikasi serta tingkat *maintainability* yang dihasilkan.

Uji coba dilakukan dengan mengembangkan suatu aplikasi Android. Untuk menguji waktu dan beban, dikembangkan aplikasi tersebut dengan menggunakan sistem dan tidak menggunakan sistem. Setelah itu, masing-masing nilai

interaction cost dibandingkan. Hasil dari pengujian tersebut membuktikan bahwa pengembangan aplikasi dengan menggunakan sistem dapat menghemat waktu dan beban sebesar 42% daripada pengembangan aplikasi yang tidak menggunakan sistem. Makalah ini juga melakukan pengujian terhadap tingkat *maintainability* dari kode yang dihasilkan sistem. Pengujian tersebut menggunakan matriks *McCall's Software Quality Model*. Hasilnya membuktikan bahwa aplikasi yang dikembangkan dengan menggunakan sistem memiliki tingkat *maintainability* sebesar 0,81 atau 81%.

UCAPAN TERIMA KASIH

Ucapan terima kasih disampaikan kepada pembimbing penelitian dan PT. Maulidan Teknologi Kreatif (Maulidan Games) yang telah mensponsori penelitian ini.

REFERENSI

[1] (2020) "StatCounter GlobalStats," [Online], https://gs.statcounter.com/os-market-share/mobile/indonesia/, tanggal akses: 27-Jun-2020.

[2] (2020) "AppBrain" [Online], https://www.appbrain.com/stats/number-of-android-apps/, tanggal akses: 27-Jun-2020.

[3] LinkedIn, "LinkedIn Emerging Jobs Report Indonesia," LinkedIn Report, 2019.

[4] N. Jenkins (2020) "A Project Management Primer" [Online], https://www.leaxr.com/pluginfile.php/6189/mod_resource/content/2/BUS402-1.5-projectPrimer-CCBYNCSA.pdf, tanggal akses: 27-Jun-2020.

[5] C. Chen, R. Alfayez, K. Srisopha, B. Boehm, dan L. Shi, "Why Is It Important to Measure Maintainability and What are the Best Ways to Do It?," *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, 2017, hal 377-378.

[6] S. Kollanus dan J. Koskinens, "Survey of Software Inspection Research," *The Open Software Engineering Journal*, Vol. 3, hal. 15-34, 2009.

[7] I. Sommerville, *Software Engineering*, 9th ed., Boston, USA: Addison-Wesley, 2011.

[8] R. Malhotra dan A. Chug, "Software Maintainability: Systematic Literature Review and Current Trends," *International Journal of Software Engineering and Knowledge Engineering*, Vol. 26, No. 8, hal. 1221-1253, 2016.

[9] R.C. Martin, *Clean Architecture: A Craftman's Guide to SoftwareStructure and Design*, London, England: Pearson Education Inc, 2017.

[10] T. Lou, "A Comparison of Android Native App Architecture-MVC, MVP and MVVM," Thesis, Eindhoven University of Technology, Eindhonven, Netherlands, 2016.

[11] S. Lappalainen dan T. Kobayashi, "A Pattern Language for MVC Derivatives," *Proc. 6th Asian Conference on Pattern Languages of Programs (AsianPLoP)*, 2017, hal. 1-8.

[12] M. Fowler, "Avoiding Repetition Software Design," *IEEE Software*, Vol. 18, No. 1, hal. 97-99, 2001.

[13] J.J. Garrett, *The Elements of User Experience: User-centered Design for the Web and Beyond*, London, UK: Pearson Education, 2010.

[14] R. Budiu (2013) "Interaction Cost" [Online], https://www.nngroup.com/articles/interaction-cost-definition/, tanggal akses: 26-Jun-2020.

[15] R.E. Al-Qutaish, "Quality Models in Software Engineering Literature: An Analytical and Comparative Study," *Journal of American Science*, Vol. 6, No. 3, hal. 166-175, 2016.

[16] L.J. Arthur, *Measuring Programmer Productivity and Software Quality*, Hoboken, USA: John Wiley & Sons, Inc., 1985.

[17] R.S. Pressman, *Software Engineering: A Practitioner's Approach*, London, England: Palgrave Macmillan, 2005.

[18] R.C. Martin, *Clean Code*, London, UK: Pearson Education, Inc., 2008.