BAB 2 TINJAUAN PUSTAKA

2.1 Tinjauan Pustaka

Pada penyusunan penelitian menggunakan beberapa penelitian sebelumnya termasuk jurnal - jurnal yang terkait dengan penelitian yang akan dilakukan, diantaranya :

2.1.1 Teknologi *push notification* berbasis Android untuk informasi perkuliahan(studi kasus : STMIK royak kisaran)[9]

Penelitian yang disusun oleh Mohd Siddik ini membahas mengenai pemanfaatan teknologi *FCM* dalam mengirimkan *push notification* untuk digunakan sebagai media penyampaian informasi untuk mahasiswa dalam hal perkuliahan dan juga informasi kegiatan

Hasil dari penelitian ini adalah sebuah layanan sistem informasi yang bekerja dilingkup kampus STIMIK Royal. Informasi atau aktifitas terbaru akan dikirimkan dengan bentuk notifikasi ke setiap mahasiswa yang menginstall aplikasi tersebut, sehingga walaupun mahasiswa tidak membuka aplikasi, mahasiswa dapat mengetahui adanya informasi atau aktifitas terbaru.

2.1.2 Implementasi *push notification* pada informasi perkuliahan dan kegiatan mahasiswa berbasis Android[10]

Jefferson kurniawan dalam penelitiannya melakukan pengintegrasian teknologi dengan sistem perkuliahan untuk diterapkan di kampus Ukrida. Hasil akhir dari Penelitian ini adalah aplikasi yang memudahkan mahasiswa dalam mendapatkan informasi perkuliahan dan informasi kegiatan yang ada di Ukrida. penintegrasian aplikasi dengan *push notification* memungkinkan Setiap informasi baru akan langsung dikirimkan ke smartphone pengguna dan ditampilkan dalam bentuk notifikasi.

Dalam pengembangannya Jefferson Setiawan menggunakan metode *waterfall* dikarenakan waterfall memilki beberapa kelebihan diantaranya analisis kebutuhan diawal dan perkembangan dari pengembangan dari sistem yang dapat diukur. Namun salah satu layanan teknologi yang digunakan untuk pengiriman notifikasi aplikasi masih menggunakan (GCM) Google *Cloud* Messaging, saat tulisan ini dibuat, GCM sendiri sudah di-*deprecated* dan tidak direkomendasikan untuk digunakan oleh Google[11].

2.1.3 Pengembangan Perpustakaan Digital Berbasis Android Dengan Metode Scrum[12]

Kurangnya layanan berbasis sistem yang disediakan diperpustakaan SMA Negeri 88 seperti pencarian buku dan historis peminjaman buku mendorong agung wahyudi untuk mengembangkan sistem Android yang mengatasi masalah tersebut. Menggunakan *framework scrum* dalam proses pengembangannya membuat produk yang dihasilkan akan disesuaikan dengan lingkungan seiring proses pengembangan sistem.

Hasil dari penelitian ini adalah sebuah Aplikasi sistem informasi perpustakaan berbasis Android, dibuat menggunakan Java dengan Android SDK sebagai bahasa pemrograman, MySql sebagai basis data pendukung. Aplikasi ini dikembangkan melalui tahapan pendekatan scrum pada perancangan aplikasi. aplikasi yand dibangun dapat memberikan kemudahan dalam mengendalikan, memanipulasi dan mengontrol data informasi perpustakaan secara *mobile*.

2.1.4 Pengembangan Fitur Notifikasi Pada Website Application Comic Strip rupi.co Menggunakan Metode Agile[13]

Kurangnya layanan berbasis sistem yang disediakan diperpustakaan SMA Negeri 88 seperti pencarian buku dan historis peminjaman buku mendorong agung wahyudi untuk mengembangkan sistem Android yang mengatasi masalah tersebut. Menggunakan *framework scrum* dalam proses pengembangannya membuat produk yang dihasilkan akan disesuaikan dengan lingkungan seiring proses pengembangan sistem.

Hasil dari penelitian ini adalah sebuah Aplikasi sistem informasi perpustakaan berbasis Android, dibuat menggunakan Java dengan Android SDK sebagai bahasa pemrograman, MySql sebagai basis data pendukung. Aplikasi ini dikembangkan melalui tahapan pendekatan scrum pada perancangan aplikasi. aplikasi yand dibangun dapat memberikan kemudahan dalam mengendalikan, memanipulasi dan mengontrol data informasi perpustakaan secara *mobile*.

Tabel 2.1 Penelitian Terdahulu

No	Judul, Penulis, Tahun	Masalah	Hasil	Perbedaan dengan penelitian yang dilakukan
1	**Judul:** Teknologi *push notification* berbasis Android untuk informasi perkuliahan (studi kasus :STMIK royak kisaran) **Penulis:** Mohd Siddik[1], Akmal Nasution[2] **Tahun:** 2018	Bagaimana merancang aplikasi informasi perkuliahan dan kegiatan mahasiswa di Ukrida berbasis Android	Hasil dari penelitian ini adalah sebuah layanan sistem informasi yang bekerja dilingkup kampus STIMIK Royal. Informasi atau aktifitas terbaru akan dikirimkan dengan bentuk notifikasi ke setiap mahasiswa yang menginstall aplikasi tersebut, sehingga walaupun mahasiswa tidak membuka aplikasi, mahasiswa dapat mengetahui adanya informasi atau aktifitas terbaru.	Fitur untuk mem*broadcast* pesan belum tersedia secara mobile dari aplikasi Saat notifikasi di klik, notifikasi tidak membuka halaman khusus yang menampilkan isi notifikasi secara lebih detail
2	**Judul:** Implementasi *push notification* pada informasi perkuliahan dan kegiatan mahasiswa berbasis Android	1.Bagaimana merancang aplikasi informasi perkuliahan dan kegiatan mahasiswa di Ukrida berbasis Android, yang terintegrasi dengan *push notification* dengan menggunakan data yang telah disediakan oleh sistem informasi Ukrida?	Penelitian ini menghasilkan aplikasi yang mampu memberikan informasi jadwal ujian bagi setiap mahasiswa, tetapi juga dapat mendaftarkan jadwal ujian ke sistem android untuk nantinya di-*input*-kan sebagai reminder melalui aplikasi Calendar. Selain itu layanan *push notification* juga mampu mengirimkan	teknologi *push notification* yang digunakan menggunakan GCM yang pada saat tulisan ini dibuat sudah *deprecated* dan diganti menjadi FCM.

No	Judul, Penulis, Tahun	Masalah	Hasil	Perbedaan dengan penelitian yang dilakukan
	Penulis: Jefferson Setiawan[1], Edy Kristianto[2], Fredrica[3] **Tahun**: 2015	2. Apakah fitur-fitur pada smartphone dapat dimaanfaatkan untuk membantu mahasiswa dalam perkuliahan?	pesan notifikasi kepada pengguna tertentu secara *on the way.*	Metode pengembangan yang digunakan merupakan SDLC
3	**Judul:** Pengembangan Perpustakaan Digital Berbasis Android Dengan Metode Scrum **Penulis:** agung wahyudi **Tahun** : 2018	Dalam menyelenggarakan kegiatan operasional sekolah SMA NEGERI 88 JAKARTA Belum menggunakan sistem komputerisasi sehingga sulit menghasilkan informasi tentang nilai, buku induk guru dan buku induk siswa.Hal tersebut disebabkan karena data-data yang tersimpan sangat banyak dan disimpan sangat kurang efisien	Hasil dari penelitian ini adalah sebuah Aplikasi sistem informasi perpustakaan berbasis *Android,* yang dapat memberikan kemudahan dalam mengendalikan, memanipulasi dan mengontrol data informasi perpustakaan secara *mobile.* Aplikasi dibuat menggunakan Java dengan Android SDK sebagai bahasa pemrograman, MySql sebagai basis data pendukung. Aplikasi ini juga dikembangkan melalui tahapan pendekatan scrum pada perancangan aplikasi.	Aplikasi yang dikembangkan merupakan perpustakaan digital aplikasi yang dikembangkan menggunakan bahasa pemrograman java
4	**Judul:** Pengembangan Fitur Notifikasi Pada Website Application	Dari segi tampilan Rupi.co masih sederhana, pada menu bar pengguna dapat melihat menu utama rupi yang	Hasil dari pengujian *test case* menggunakan metode *black box* menunjukan semua sistem yang	Fitur notifikasi yang dikembangkan merupakan

No	Judul, Penulis, Tahun	Masalah	Hasil	Perbedaan dengan penelitian yang dilakukan
	Comic Strip rupi.co Menggunakan Metode Agile **Penulis:** Moh. Roziq Bahtiar, Anggraini Mulwida **Tahun** : 2016	terdiri dari new, popluar, top user, top post, upload dan kategori komik yang terdapat dalam menu subrupi. Penulis ingin mengembangkan fitur notifikasi pada website ini dengan menerapkan metode agile scrum dalam pengembangannya	dikembangkan dapat berjalan dengan baik sesuai yang direncanakan. Analisis user experience melibatkan 22 responden diperoleh informasi bahwa kategori daya tarik, kejelasan, efisiensi, dan kebaruan bernilai baik. Sementara ketepatan dan stimulasi bernilai di atas rata-rata. Hal ini berarto bahwa pengembangan fitur ini dapat memberikan pengalaman pengguna (user experience) yang baik kepada penggun saat berinteraksi dengan rupi.	*push notification* di aplikasi web sever database yang digunakan bertype RDBMS *firebase* tidak digunakan sebagai backend

2.2 Landasan Teori

2.2.1 Android

Android adalah platfrom yang terdiri dari *operating system, software libraries, application frameworks, software development kit, pre-built applications* dan *reference design*. *platfrom* maupun eko-sistem dari Android yang terus berkembang dari waktu ke waktu[[14]. mayoritas aplikasi Android dikembangkan dengan menggunakan bahasa pemrograman Java ataupun Kotlin agar nanti untuk selanjutnya di *compile* oleh Android API. dan di terjemahkan kedalam *bytecode* untuk VM Android-Spesific[[15].

Android studio adalah *Integrated Development Environment* (IDE) resmi untuk pengembangan aplikasi Android dan bersifat *open source* atau gratis. Android studio sendiri dikembangkan berdasarkan IntelliJ IDEA yang dikembangkan oleh JetBrains[16]. Hingga laporan penelitian ini dibuat sistem operasi Android terdiri dari beberapa versi, sampai versi ke-9 setiap versinya memiliki nama-nama unik yang terdiri dari nama makanan atau kue yang berurut abjad seuai jadwal rilisnya.

1) Android versi 1.0 dirilis tanggal 23 September 2008
2) Android versi 1.1 dirilis tanggal 9 Februari 2009
3) Android versi 1.5 Cupcake dirilis tanggal 30 April 2009
4) Android versi 1.6 Donut dirilis tanggal 15 September 2009
5) Android versi 2.0 Éclair dirilis tanggal 26 Oktober 2009
6) Android versi 2.2 Froyo dirilis tanggal 10 Mei 2010
7) Android versi 2.3 Gingerbread dirilis tanggal 6 Desember 2010
8) Android versi 3.0 Honeybomb dirilis tanggal 20 Februari 2011
9) Android versi 4.0 ICS dirilis tanggal 19 Oktober 2011
10) Android versi 4.1 Jelly Bean dirilis tanggal 9 Juli 2012
11) Android versi 4.4 Kitkat dirilis tanggal 31 Oktober 2013
12) Android versi 5.0 Lollipop dirilis tanggal 17 Oktober 2014
13) Android versi 6.0 Marshmallow dirilis tanggal 28 Mei 2015
14) Android versi 7.0 Nougat dirilis tanggal 22 Agustus 2016
15) Android versi 8.0 Oreo dirilis tanggal 21 Agustus 2017
16) Android versi 9.0 Pie dirilis tanggal 6 Agustus 2018 [17]

17) Androd versi 10 dirilis tanggal 3 september 2019
18) Android versi 11 dirilis tanggal 30 september 2020

Setelah versi 9.0, Android tidak lagi mengadopsi nama-nama makanan penutup pada penamaan versinya, hanya disebut Android 10 dan Android 11. Merepresentasikan versi ke 10 dan 11 di Android. Perubahan nama ini dilakukan oleh Google untuk menghindari kebingungan di pasar *global* karena tidak setiap orang di dunia mengerti akan makna dari nama-nama makanan penutup tersebut. Google meyakini nama Android tanpa codename akan terasa lebih jelas dan dapat diterima oleh semua orang [[18].

2.2.2 Kotlin

Kotlin adalah bahasa baru yang men-targetkan *Java platfrom*, dibuat agar bisa berjalan diatas *Java virtual mechine* (JVM) . Kotlin dirilis dibawah naungan Jetbrains, perusahaan sama yang juga merilis Intelij, PyCharm, WebStrom, ReSharper dll. Di tahun 2011 Jetbrains merilis Kotlin secara resmi dibawah lisensi apache 2. dan ditahun 2017 Google mengumumkan *first class support* untuk Kotlin dalam pengembangan Android.

Seperti halnya bahasa pemrograman Java yang terinspirasi dari pulau jawa di Indonesia dalam penamaannya. Nama Kotlin juga terinspirasi dari pulau dengan nama sama yang terletak di Rusia [14]. sebagai bahasa baru dengan kemiripan *syntax* dengan Java, Kotlin memiliki beberapa kunggulan dibandingkan dengan Java adalah sebagai berikut :

a. More Concise

Sebuah *class / function* dapat ditulis lebih singkat dengan menggunakan Kotlin dibandingkan dengan java tanpa mengurangi kegunaan dari *class / function* tersebut. Misalkan sebuah *class Plain Old Java Object* (POJO) dengan hanya 3 *property, class pojo* dapat memiliki lebih dari 20 baris, dengan Kotlin dapat dipersingkat menjadi 1 baris *data class* tanpa mengurangi fungsionalitas dari *class* tersebut

b. Null Safety

Kotlin adalah bahasa pemrograman dengan fitur *null-safe*, yang artinya *programmer* akan dituntut untuk berurusan dengan situasi yang dapat memicu *null* saat *compile time*, sehingga dapat meminimalisir *throw exception* saat *running time*. Dalam

Kotlin *programmer* harus mendefinisikan secara *explisit* apakah suatu *variabel* dapat berisi *null*, atau tidak. Jika sebuah *variabel* didefinisikan dengan dapat berisi *null*, maka setiap pemanggilan dari *variable* tersebut Android Studio akan memberikan peringatan untuk melakukan *null-check* sebelum menggunakannya.

c. Functional Programming support

Dasar dari Kotlin sendiri adalah *object oriented programming* (OOP), tidak murni *functional*. Tetapi, sama halnya dengan bahasa pemrograman modern lainnya, yang berfokus pada kefisienan dalam penulisan *syntax*, Kotlin juga mengadopsi beberapa konsep dari functional programming, seperti lambda dan beberapa func*tional operat*or seperti *map, filter, foreac*h dll.

d. Extension Function

programmer memiliki kemampuan untuk menambah fitur dari sebuah kelas tanpa perlu meng-*extends* kelas tersebut dan bahkan jika *programmer* tidak memilki akses kedalam *source code* kelas tersebut sekalipun, *extension function* tetap dapat dilakukan [19]

e. Interopable

Mayoritas program yang ditulis menggunakan Java dapat langsung digunakan di Kotlin tanpa perlu melakuan konversi *class* dikarenakan interopabilitas dari kedua bahasa yang sangat tinggi[19].

Salah satu keunggulan lain dari Kotlin adalah *coroutine. Coroutine* pada dasarnya adalah struktur kontrol umum di mana kontrol dari aliran program secara kooperatif dilewatkan di antara dua *routine* yang berbeda saat kembali. *Coroutine* memungkinkan sebuah program komputer untuk melakukan suspend saat eksekusi dan akan dipanggil kembali saat program tersebut telah memiliki *result* [20].

Gambar 2.1 Konsep Coroutine[21]

Singkatnya, Kotlin *coroutine* adalah semacam cara untuk menuliskan asynchrounus kode secara sekuensial seperti pada Gambar 2.1 diatas. Cara ini lebih baik dibandingkan dengan menggunakan *callback* dengan *coroutine programmer* dapat menuliskan kode dari atas kebawah, dan jika proses pada kode atas belum selesai , tetapi kode dibawah membutuhkan *result* dari kode diatasnya maka proses akan menunggu hingga kode diatas mendapatkan *return*-nya[[19].

Dalam pengembangan aplikasi yang digunakan di penelitian ini, *coroutine* akan digunakan untuk proses yang membutuhkan waktu *running* lama. Diharapkan dengan dengan adanya implementasi dari *coroutine* di aplikasi dapat membuat struktur kode dari aplikasi lebih sederhana dan lebih mudah dipahami

2.2.3 *Firebase*

Ditahun 2016 Google merilis *Firebase* dengan bertujuan untuk menyediakan seperangkat alat dan infrastruktur yang dibutuhkan dalam mendevelop sebuah aplikasi. mengeliminasi kebutuhan dalam membuat *back-end database*, *secure authentication, messaging*, dll, agar *developer* dapat lebih terfokus pengembangan aplikasinya[7].

Firebase mendukung pengembangan aplikasi berbasis *mobile* dan *web* atau disebut juga sebagai *Backend as a Service (BaaS).* Dalam penelititan ini, untuk memudahkan dalam pengembangan aplikasi, beberapa fitur yang ada pada *firebase* akan digunakan sebagai pendukung aplikasi, diantaranya :

a. Firebase Authentification

Firebase authentication di bangun agar pengambang aplikasi dapat membangun proses autentifikasi yang aman tetapi juga meningkatkan *experience sign-in* dan *auto-login*

bagi *end users. Firebase* Authentication mendukung banyak proses *authentication* seperti *e-mail* dan *password login*, *telephone*, Google , Twitter, Facebook, and GitHub[[22].

b. Firebase Cloud Messaging (FCM)

FCM didesain untuk menyediakan koneksi ke perangkat mu melalui pesan dan notificaiton, *FCM* bertujuan untuk bisa menjadi fitur yang dapat diandalkan, dengan 98% dari pesan yang dikirim dapat tersampaikan ke perangkat selama 500ms atau kurang, *FCM* juga mampu untuk mengirim pesan secara efektif ke pengguna tertentu secara tepat untuk menghindari notifikasi *scam. FCM* menawarkan berbagai bermacam-macam variasi penggunaan yang masih dapat dikontrol oleh pengembang.

Selain kemampuan untuk mengirim pesan ke pengguna spesifik, *firebase* juga memilki kemampuan dalam bagaimana pesan dikirim, misalkan mengatur prioritas, tanggal kadaluarsa, dan masih banyak lagi. Pengembang juga dapat melihat bagaimana pesan diterima dan berinteraksi dengan pengguna[7]

c. Firebase Cloud Storage

Firebase Cloud Storage dirancang untuk *developer* aplikasi yang perlu menyimpan dan menyajikan konten yang dibuat pengguna contoh foto atau file lainnya. Ini memberikan transfer dokumen yang aman dan unduh untuk aplikasi *Firebase*, terlepas dari kualitas jaringan. *Firebase Storage* didukung oleh *Google Cloud Storage*, layanan penyimpanan objek yang mampu, dasar, dan hemat biaya [23].

d. Cloud Firestore

Cloud Firestore atau yang sering disebut *Firestore* adalah sebuah *database* yang memiliki sistem adaptasi tinggi dan mampu digunakan dalam berbagi hal dalam pengembangan aplikasi. Sama halnya seperti *Firebase Realtime database*, *Firestore* menjaga semua informasi dapat ter singkronisasi di berbagai aplikasi melalui *callback listener* secara *real-time*.

Firestore juga menawarkan kemampuan untuk *offline in-app* sehingga aplikasi tetap dapat merespon tugas perlu khawatir terganggu *network latency* ataupun ketidakstablian jaringan internet. Firestore juga menawarkan kemudahan dalam pengintegrasian dengan fitur *Firebase* lain maupun produk dari Google *Cloud* Platfrom, termasuk *Cloud* Functions[22]

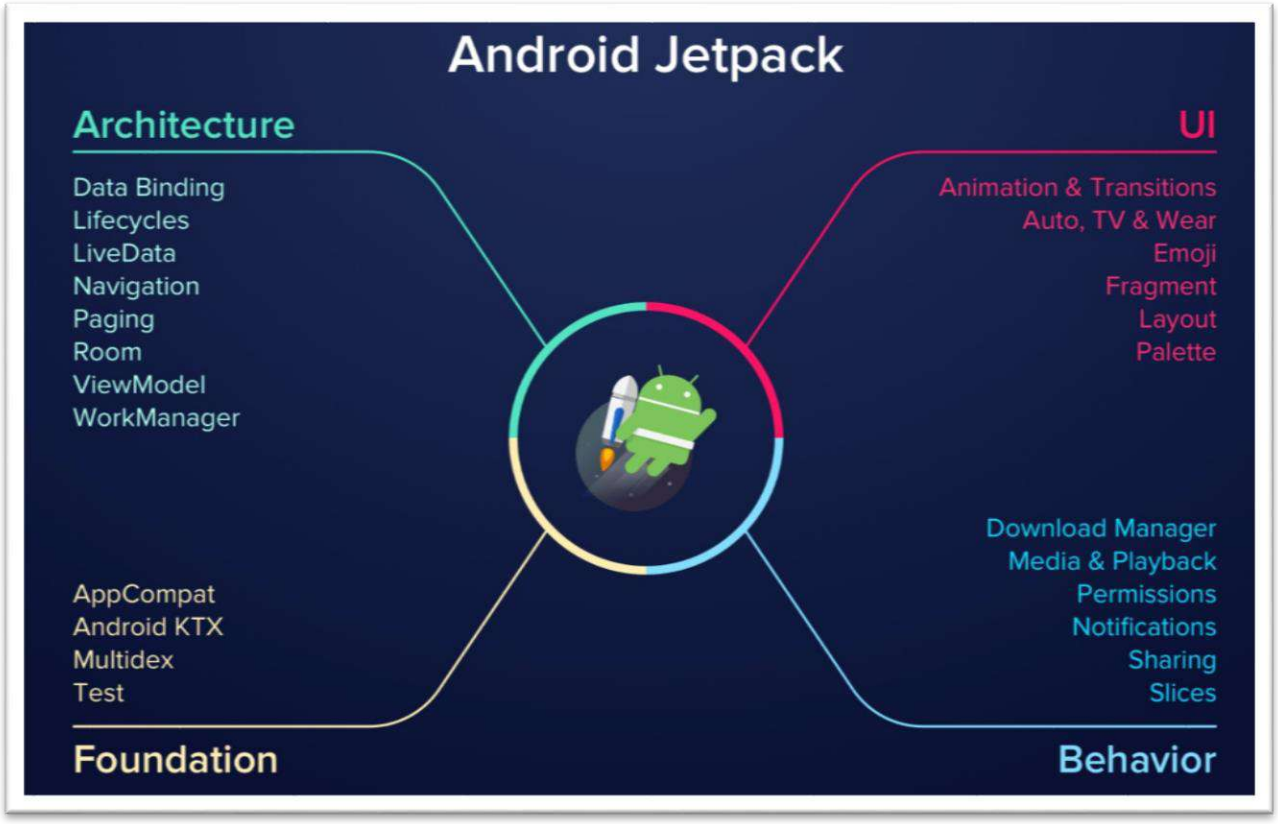
2.2.4 *Push Notification*

Push notification adalah sebuah pesan singka atau semacam peringata yang “dikirim” melaui *server* aplikasi yang nantinya pesan tersebut dapat diterima oleh pengguna aplikasi bersangkutan. Aplikasi tidak harus terbuka agar *push notification* terkirim[24].

Dalam Penelitian ini, *push notification* yang digunakan adalah FCM. FCM nantinya akan digunakan sebagai perantara *broadcast* bagi dosen dengan mahasiswa. Saat adanya pengumuman jam kosong atau perubahan jadwal yang akan dilakukan oleh dosen, dosen dapat membuat pesan tersebut dari aplikasi yang nantinya dikirim ke *Firebase*. *Firebase* selanjutnya akan meneruskan pesan tersebut ke mahasiswa yang terkait menggunakan FCM

2.2.5 Android Jetpack Component

Android Jetpack Component adalah kumpulan dari berbagai macam *library* yang masing-masing komponennya dapat digunakan secara individu maupun bersamaan, dengan memanfaatkan kelebihan dari bahasa pemrograman Kotlin dapat membuat *developer* menjadi lebih produktif.



Gambar 2.2 Daftar Android Jetpack Component[25]

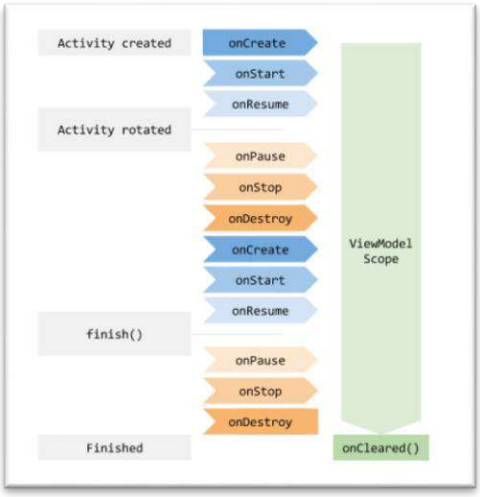
sesuai dengan Gambar 2.2 diatas, *Android jetpack Component* terdiri dari 4 bagian yang saling melengkapi satu sama lain, terdiri dari :

1) Architecture : architecture component membantu developer dalam merancang aplikasi sehingga lebih powerfull, *test*able dan maintainable
2) *Behavior* : *behavior component* membuat aplikasi dapat terintegrasi dengan standar android servis seperti notifikasi, perizinan, berbagi, dan asisten
3) *Foundation* : *foundation component* menyediakan fungsionalitas untuk *backwards compatibility*, *testing*, dan dukungan bahasa Kotlin
4) *UI* : *UI* komponen menyediakan *widgets* dan *helpers* untuk aplikasi. Sehingga aplikasi tidak hanya mudah, tetapi juga menyenangkan digunakan[26].

Dari empat jenis *component* dan belasan *library* yang ada, *architerture* menjadi pelengkap dalam mengembangkan aplikasi android. dalam penelitian ini, penulis akan menggunakan beberapa *library* yang ada di *architecture* diantaranya *ViewModel, LiveData, Room.*

a. *ViewModel*

Tujuan dari viewmodel adalah untuk memeisahkan *user interface-related data model* dan logika aplikasi dari *Source code UI*. Dengan cara seperti ini *activity* dari aplikasi akan bertindak sebagai *UI controllers,* dimana viewmodel bertanggung jawab dalam mengatur data yang dibutuhkan *activity* atau *Fragment.*



Gambar 2.3 ViewModel[27]

Pembagian semacam ini akan mengatasi isu yang berkaitan dengan *lifecycle* dari *activity* atau Fragment. Pada Gambar 2.3 diatas menggambarkan Seberapa banyak pun sebuah *Activty* (maupun *Fragment*) ter-*recreated* selama aplikasi berjalan, *instance* dari *ViewModel* akan tetap sama, dikarenakan *instance* tersebut tidak terikat di *activity* melainkan berada di memory sehingga data di *ViewModel* tetap terjaga[28].

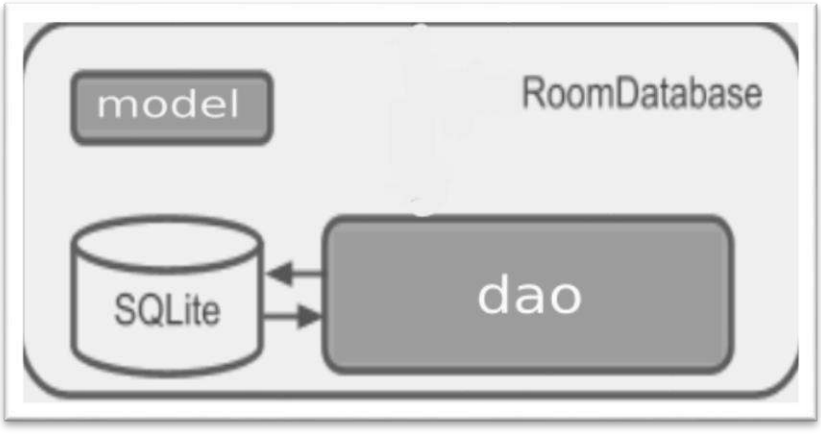
b. *LiveData*

LiveData adalah *data holder* yang memuat sebuah *value dan* memiliki kemampuan *Observable*. Artinya *LiveData* memiliki kemampuan untuk memberitahu *object* lain ketika adanya perubahan pada *value* di livedata terjadi, dengan demikian *livedata* mengatasi *issue* dalam memastikan data yang ditampilkan sama dengan data yang ada di *viewmodel*.

Kelebihan lain dari livedata adalah *lifecyle-aware* dari *observer*-nya. Jika *activity* memiliki livedata *observer*, maka objek *livedata* tersebut akan tahu jika *lifecycle state* dari *activity* tersebut berubah sehingga *livedata* dapat bertindak sesuai *state* dari *activity*. Jika *activity* ter-*pause livedata* objek tidak akan melakukan pengiriman *update event* ke *observer*. Begitu juga sebaliknya jika *activity* dalam keadaan *started* atau *resumed livedata* akan melakukan *update* data sehingga *activty* akan memiliki data terbaru[28].

c. Room

Room persistence library menyediakan sebuah abstraksi *layer* diatas SQLite sehingga pengaksesan *database* menjadi lebih sederhana dengan tetap mempertahankan keunggulan dari SQLite



Gambar 2.4 Room Database Diatas SQLite

Berdasarkan Gambar 2.4 diatas *Room* terdiri dari beberapa elemen pendukung diantaranya:

1) *Room database : Room database* objek adalah antarmuka untuk *SQLite database*. Room database juga menyediakan
2) *Data Access Object (DAO)* : Dao adalah sekumpulan *SQL statements* yang dibutuhkan oleh *Room* untuk melakukan manipulasi data seperti *insert, update, queryin*g dan *delete* didalam S*QLite database.*
3) *Model Entities : Model Entities* adalah kelas yang mendefeinisikan *schema table* di *database*. *Model Entity* mendefinisikan nama tabel, noma kolom, dan tipe data, dan kolom mana yang akan menjadi *primary key*.
4) *Sqlite database* : *Database* sebenarnya yang bertanggung jawab untuk menyimpan dan menyediakan data kedalam *database*. *Source code* dari *app* tidak boleh melakukan akses langsung ke dalam *database* ini. Semua operasi database dilakukan dengan kombinasi dari *Room database, DAO* dan *entity*[28].

2.2.6 Android Architecture Pattern

Architecture pattern adalah tata cara yang berupa batasan / aturan yang berfungsi untuk membantu *programmer* dalam men-*design* struktur kodingan aplikasi dengan memberikan yang mengizinkan aplikasi untuk dapat lebih terawat sejalan dengan berkembangnya aplikasi[29]. Pada umumnya dua konsep umum dari architecture component adalah *Separation of concern,* dan *Testability.*

a. Separation of concern

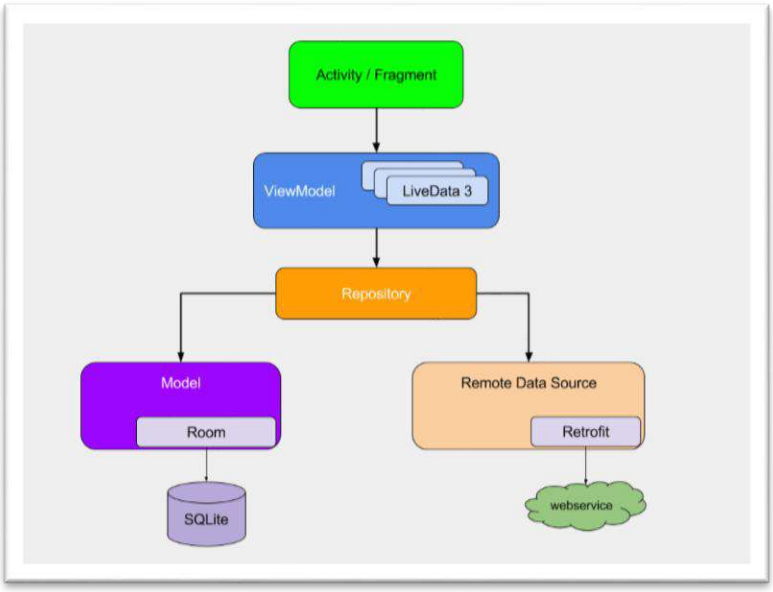
Separation of concern adalah pemisahan masing-masing komponen aplikasi berdasarkan tugas dan kemiripannya. Setiap kali *programmer* melakukan penambahan fitur di aplikasi *programmer* dapat melakukannya tanpa harus mengubah terlalu banyak *code*

b. *Test*ability

Dilakukannya *separation of concern* secara tidak langsung juga membuat proses *testing* menjadi lebih mudah. Dengan dilakukannya pemisahan *source code*, diharapkan dapat meningkatkan *testability* dari suatu fungsi

Beberapa *architecture pattern* yang ada saat tulisan ini dibuat ada setidaknya 5 archtecture pattern di Android diantaranya : *model-view-controller (*MVC), *model-view-presenter* (MVP), *model-view-viewmodel* (MVVM), *model-view-intent* (MVI), *view-interactor-presenter-entitiy-router* (VIPER). dalam proses pengembangan aplikasi notifikasi jadwal kuliah, Penulis memutuskan untuk menggunakan architecture yang lebih familiar yaitu MVVM

MVVM *architecture* memiliki 3 komponen, yaitu *model*, *view* dan *viewmodel*. *view* bertugas untuk menampilkan *UI* (*User Interface*) dari aplikasi, *model* merepresentasikan data yang akan ditampilkan oleh *view*, dan *viewmodel* bertugas dalam proses pengiriman data ke *view[24]*[30]. Contoh architecture MVVM dapat dilihat seperti Gambar 2.5 dibawah ini.



Gambar 2.5 MVVM architecture[31]

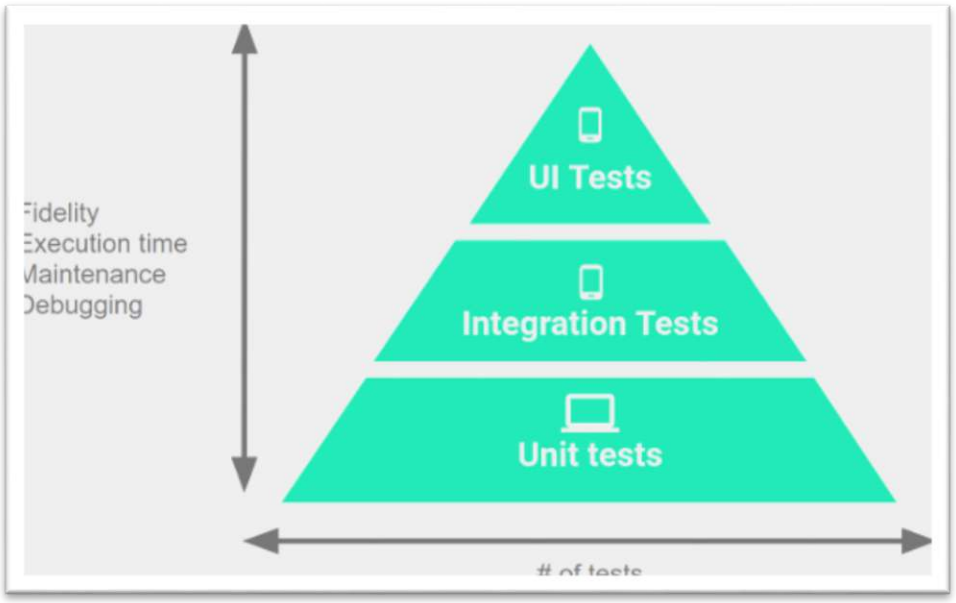
Dalam prosesnya, *view* tidak boleh tahu / terlibat dalam bagaimana data diterima seperti yang digambarkan di Gambar 2.5, *view* hanya mendapatkan data dari *viewmodel* lalu mengolahnya sesuai kebutuhan dari si *view* itu sendiri. Di sisi lain *viewmodel* juga tidak boleh terikat kuat dengan *view*, misalkan *viewmodel* menjadi *dependent* terhadap *activity context* yang pada umumnya dimiliki dan terikat kuat dengan *view*.

2.2.7 *Testing*

Pada saat proses pengembangan suatu aplikasi, memastikan setiap kode dapat berjalan dengan semestinya menjadi sebuah keharusan dari setiap *programmer*. *Testing* dilakukaan agar

programmer memastikan bahwa fitur-fitur yang sudah diterapkan berjalan dengan semestinya, dan juga fitur baru yang diterapkan tidak menggangu atau merusak logika dari fitur yang sudah ada [29].

Tahapan *testing* yang dilakukan oleh *developer* untuk memastikan kode yang dibuat dapat berjalan secara semestinya. Tahapan *testing* ini dilakukan dalam 3 bagian yang mencakup 3 kategori berbeda, yaitu *small testing medium testing* dan *large testing*.



Gambar 2.6 Testing Pyramid[32]

a. *Small test*

Unit test yang bertujuan untuk memvalidasi perilaku aplikasi dari satu kelas dalam satu waktu. Dengan membuat dan menjalankan *unit test* pada *code* yang dibuat, penulis dapat dengan mudah memverifikasi logika pada *unit*(yang bisa berupa *method*, *class*, atau *component*) yang di *test* benar[33].

b. *Medium test*

Medium test yang dilakukan untuk memvalidasi antara interaksi antar *level* dari *stack module*, atau interaksi antar modul. Misalkan aplikasi menggunakan komponen dimana pengguna tidak berinteraksi secara langsung dengan komponen tersebut[34].

c. *Large test*

Large test yang dilakukan secara *end-to-end* yang memvalidasi kenyamanan dari si pengguna dalam menggunakan fitur aplikasi, dimana fitur tersebut dapat terdiri dari bebarapa modul. pada tahap ini penulis menggunakan teknik *black box testing*. *black box testing* yang didasarkan pada detail aplikasi seperti tampilan aplikasi, fungsi-fungsi yang ada pada aplikasi, dan kesesuaian alur fungsi dengan bisnis proses yang diinginkan oleh pengguna[35]. *black box testing* mencoba menemukan kesalah dalam beberapa kategori berikut:

1) fungsi yang tidak benar atau hilang
2) kesalahan dalam tampilan interface
3) kesalahan daam struktur data atau akses database eksternal
4) kesalahan perilaku atau kinerja
5) inisialiasasi dan penghentian kesalahan

Dalam penelitian ini, saat melakukan *small testing* dan *medium testing* pada saat proses pengembangan, selain menggunakan *API testing standar android*, penulis juga menggunakan beberapa *library testing* tambahan untuk mempermudah dalam menulis *test*ing diantaranya :

a. *assertion Truth*

Truth merupakan library yang mempermudah proses pengecekan atau *assertion* pada *test*ing sehingga terlihat lebih *readable. Truth* dimiliki dan di rawat oleh *team* Guava, selain itu *Truth* juga sudah banyak digunakan pada mayoritas *codebase* Google. kelebihan dari *Truth* selain *code*-nya mudah dibaca, Hasil *failure message* juga lebih bisa dipahami [36]

Tabel 2.2 Perbandingan Code Dengan Dan Tanpa Assertion Truth

1	 *Gambar 2.7 Assertion Code Tanpa Truth*
2	 *Gambar 2.8 Asssertion Code Dengan Truth*

Dapat dilihat perbedaan *code* pada Tabel 2.2 diatas, *code* pada Gambar 2.8 terlihat lebih mudah dibaca dan ditulis dibandingkan dengan *code* pada Gambar 2.7

b. mockito-kotlin

Mockito-kotlin adalah *wrapper library* untuk *mockito*. *Mockito kotlin* menyediakan *top-level functions*. Sehingga membuat penulisan *test* dengan *mockito* lebih *idomatic*. *Mockito-kotlin* tidak bertujuan untuk mengganti keseluruhan fungsionalitas dari *mockito*, hanya beberapa fungsionalitas yang bersinggungan dengan kotlin[37]

c. *robolectric*

Robolectric adalah *framework* yang membawa kecepatan dan reliabilitas dari *unit test* ke android. *Robolectric* mampu melakukan *test ke activity* atau *fragment* tanpa perlu *emulator* atau perangkat. Semua *test* dilakukan diatas JVM. Sehingga proses *dexing*, *packaging*, dan instalasi app pada emulator tidak diperlukan, mengurangi proses *test* dari yang awalnya proses *test*ing *UI* dapat memakan waktu beberapa menit, dapat berkurang sehingga menjadi beberapa detik [38].

Untuk memastikan agar aplikasi dapat berjalan diberbagai macam smartphone dan versi android lain, penulis juga menggunakan *device farm* browserstack, yaitu sebuah layanan yang memungkinkan pengembangn untuk dapat melakukan *UI test* secara *cloud* pada berbagai jenis smartphone. Berdasarkan data yang dipublish oleh gs.statcounter, 3

versi android tertinggi dari september 2019 – october 2020. *market share* android di indonesia merupakan versi 9 dengan *market share* mencapai 25.82%, lalu versi 10 dengan *market share* 25.75% dan versi 8.1 dengan *market share* mencapai 16.64%[39]

2.2.8 SCRUM

Scrum adalah cara kerja beberapa orang yang berada dalam satu tim, berfokus dalam memberikan hasil kerja yang produktif, kreatif dan memiliki value yang setinggi mungkin dan tetap dapat beradaptasi dengan cepat setiap adanya perubahan [40].

Metode ini dilakukan dalam kurun waktu tertentu, dilakukan dengan mengerjakan satu modul tertentu lalu dilanjutkan ke modul berikutnya sampai menghasilkan produk yang diinginkan. scrum terdiri dari sebuah tim yang memiliki peran dan tugas masing-masing. Setiap komponen dalam kerangka melayani tujuan tertentu dan sangat penting untuk kesuksesan penggunaan scrum. langkah-angkah pengerjaan dengan metode scrum seperti yang ditunjukan pada Gambar 2.9 dibawah adalah sebagai berikut:

Gambar 2.9 Alur Scrum[41]

a. Product backlog

Scrum adalah cara kerja beberapa orang yang berada dalam satu tim, berfokus dalam memberikan hasil kerja yang produktif, kreatif dan memiliki value yang setinggi mungkin dan tetap dapat beradaptasi dengan cepat setiap adanya perubahan [40].

Metode ini dilakukan dalam kurun waktu tertentu, dilakukan dengan mengerjakan satu modul tertentu lalu dilanjutkan ke modul berikutnya sampai menghasilkan produk yang diinginkan. scrum terdiri dari sebuah tim yang memiliki peran dan tugas masing-

masing. Setiap komponen dalam kerangka melayani tujuan tertentu dan sangat penting untuk kesuksesan penggunaan scrum. langkah-angkah pengerjaan dengan metode scrum seperti yang ditunjukan pada Gambar 2.9 dibawah adalah sebagai berikut:

b. *Sprint*

Sprint merupakan sebuah kerangka waktu yang berdurasi kurang lebih 1 bulan untuk mengembangkan produk. Hasil dari 1 kali *sprint* berupa proses increment "selesai" yang memiliki potensi untuk dirilis. Dalam hasil dari perencanaan *sprint* berupa *sprint backlog*

Sprint backlog merupakan daftar *product backlog* item yang terpilih untuk *sprint* ditambah perencanaan untuk menghantarkan increment dan mencapai *sprint* goal. *Sprint backlog* merupakan prakiraan dari development team mengenai fungsionalitas yang akan masuk ke dalam increment berikutnya dan pekerjaan yang perlu dikerjakan untuk menghantarkann fungsionalitasnya menjadi increment yang "selesai"

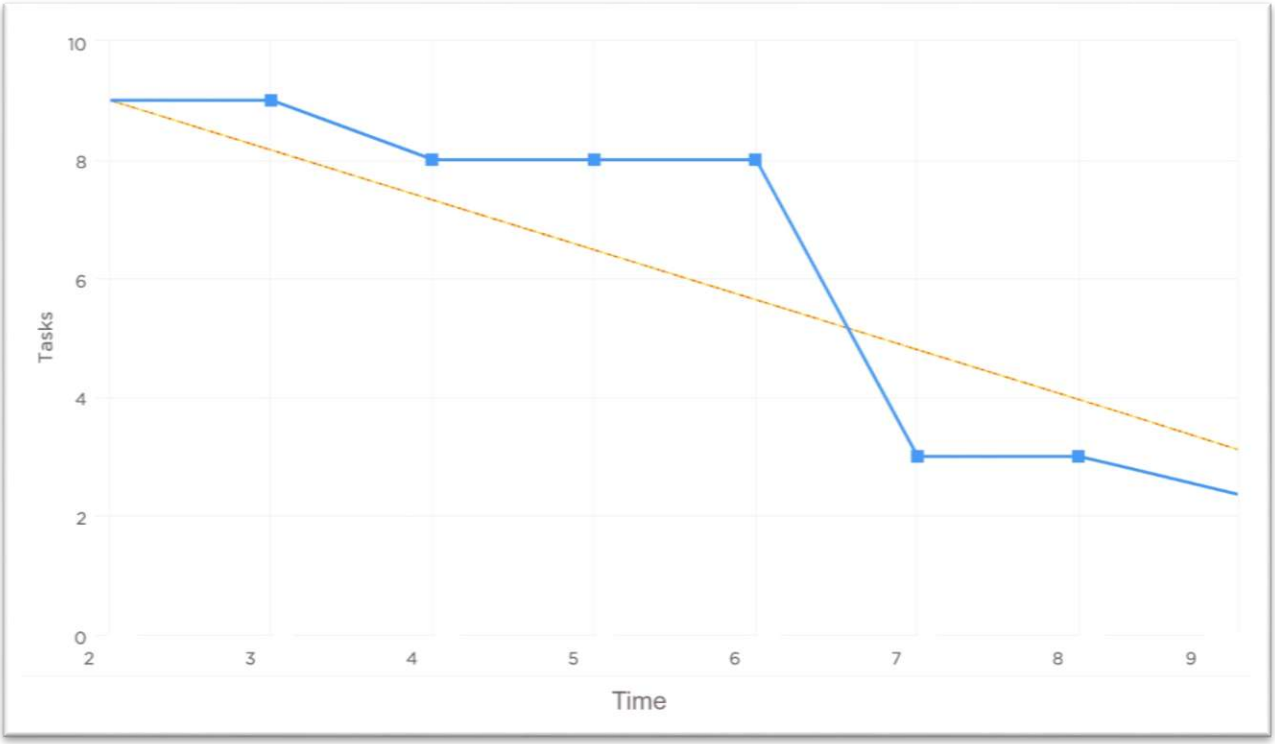
c. *Demo*

Produk didemonstrasikan kepada target pengguna secara incremental untuk kemudian dievaluasi setiap fungsi-fungsinya oleh target pengguna tersebut dan mengulangi proses dan membuat *sprint* baru jika target pengguna meminta perubahan fungsi atau menambah fungsi baru. Terakhir, produk yang sudah sesuai diberikan kepada pengguna dan proses pengembangan selesai.

d. Burndown chart

Burndown chart adalah metode yang sederhana untuk memantau proses pengerjaan suatu produk yang dikerjakan menggunakan metode *agile*. *Burndown chart* adalah cara terbaik untuk memvisualisasikan hubungan antara jumlah pekerjaan yang tersisa pada setiap titik waktu dan kemajuan Tim pengembang.

Burndown chart terdiri dari 2 parameter yang masing masing merepresentasikan sumbu *x-axis* dan sumbu *y-axis*, sumbu *x-axis* dapat berupa *story point* yang dikerjakan dalam satu kali *sprint*. Sedangkan sumbu *x-axis* merepresentasikan lama pengerjaan dalam satu kali *sprint*, dimana *unit* yang digunakan berdasarkan satuan waktu.



Gambar 2.10 Burndown Chart[42]

Story point adalah *unit* pengukuran untuk memperkirakan estimasi keseluruhan usaha yang dibutuhkan dalam menangani item *backlog*. *Story point*. agar lebih efektif, nilai *story point* memecah pekerjaan besar menjadi bagian-bagian yang lebih kecil sehingga dapat mengatasi ketidakpastian

burndown chart memiliki *ideal Task remaining line* dan *actual Task remaining line.* Yang menghubungkan titik awal ke titik akhir. Pada Gambar 2.10 diatas garis putus-putus berwarna *orange* merepresentasikan *ideal line* dan garis biru merepresentasikan *actual line.* Pada titik awal, garis yang *ideal* menunjukan jumlah dari *story point* untuk semua tugas yang perlu diselesaikan. Di titik akhir, garis *ideal* berakhir di sumbu x merepresentasikan bahwa tidak ada pekerjaan yang tersisa untuk diselesaikan. *Actual line* akan berfluktuasi di atas dan di bawah *ideal line* tergantung dari seberapa efektif pekerjaan tersebut dilakukan. jika *actual line* lebih sering berada diatas *ideal line* berarti dalam proses perancangan *sprint, Task* yang dikerjakan lebih sulit dibandingkan dari yang diperkirakan, sedangkan jika *actual line* lebih sering berada di bawah *ideal line*, maka pekerjaan yang

direncang lebih mudah dari perkiraan dan projek akan selesai lebih cepat dari jadwal yang diperkirakan

2.2.9 Unified Modelling Language

Kegunaan dari UML adalah untuk memberikan gambaran umum mengenai objek berdasarkan kondisi dan teknik *diagram* yang cukup banyak sehingga mampu untuk memodelkan project pengembangan sistem dari analysis sampai ke desain.

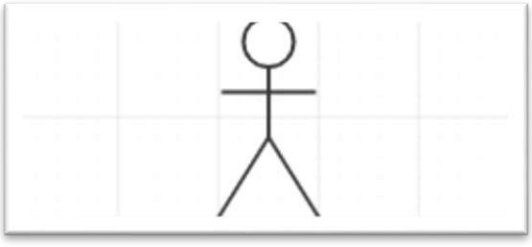
Diagram dalam UML dapat dikategorikan menjadi 2 kategori : *structure diagrams* dan *Behevior diagrams*. *Structure diagram* digunakan dalam merepresentasikan data dan hubungan mutlak yang ada sistem. Behavior *diagram*s menyediakan analis dengan cara menggambarkan hubungan dinamis di antara objek yang mewakili sistem informasi tersebut Di dalam *structure diagrams* terdapat *class diagram, object, package, deployment, component,* dan *composite structure diagram*. Sedangkan pada kategori *Behevior diagrams* terdapat *activity diagram, Sequence diagram, communication diagram, interaction overview timing diagram, behaviour state machine diagram, protocol state machine diagram,* dan *use case diagram* [43].

Dalam penelitian ini. Hanya menggunakan tiga macam pemodelan perilaku yaitu *use case diagram, sequence diagram* dan *activity diagram.*

a. UseCase Diagram

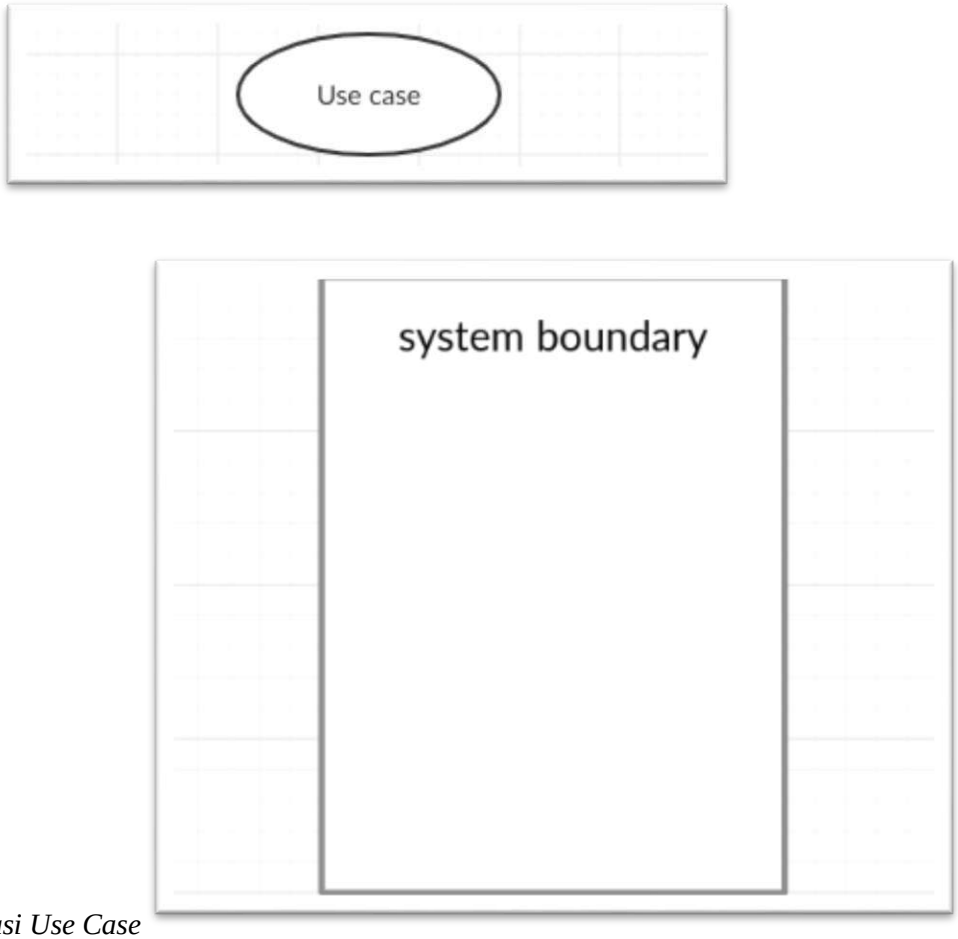
Use case menggambarkan dengan cara sederhana bagaimana fungsi utama dari sebuah sistem bekerja dan berbagai tipe pengguna saling berinteraksi. *use case* memiliki beberapa elemen diantara :

1) *Actor* : adalah orang atau sistem lain yang berinteraksi dan memberikan input ke sistem. Actory diilustrasikan dalam Gambar 2.11



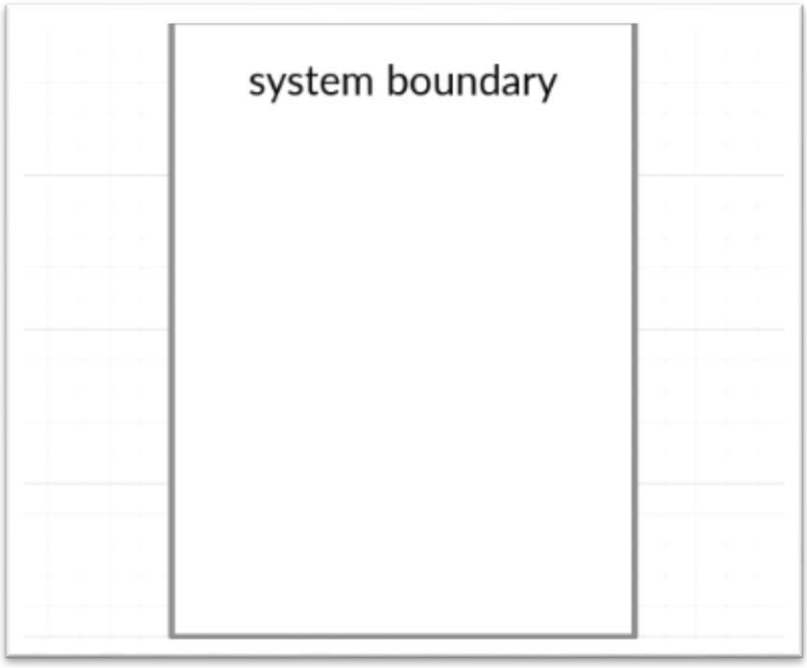
Gambar 2.11 Ilustrasi Actor

2) *use case* : adalah proses yang akan dilakukan oleh sistem dan dapat memberikan feedback untuk actor. Diilustrasikan dalam Gambar 2.12



Gambar 2.12 Ilustrasi Use Case

3) Gambar 2.13 System Boundary: use case dibatasi oleh system boundary yang berupa persegi merepresentasikan sistem dan membedakan secara jelas bagian mana saja yang termasuk internal maupun external dari sistem. Diilustrasikan dalam Gambar 2.13 dibawah :



Gambar 2.13 Ilustrasi System Boundary

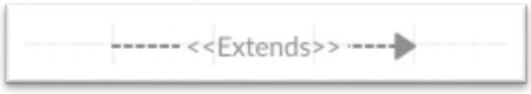
4) *Association Relationships* : melambangkan komunikasi antara *actor* dengan *use case* satu sama lain. Diilustrasikan dalam Gambar 2.14

Gambar 2.14 Ilustrasi Association Relationship

5) *Include Relationship* : menambah fungsionalitas yang tidak terdapat di *use case* standar. Diilustrasikan dalam Gambar 2.15

Gambar 2.15 Ilustrasi Include Relationship

6) *Extends Relationship* : *extend relationship* menggambarkan funsionalitas atau perilaku sistem optional. Diilustrasikan dalam Gambar 2.16

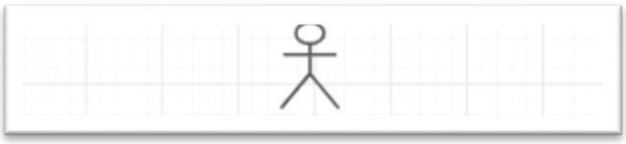


Gambar 2.16 Ilustrasi Extends Relationship

b. Sequence Diagram

Sequence Diagram mengilustrasikan objek dalam *use case* dengan mendeskripsikan interaksi antar objek secara runtut dari waktu ke waktu. elemen dari *sequence diagram* dapat dilihat dibawah ini:

1) *Actor* : merupakan orang atau sistem lain yang melakukan interaksi dengan sistem, *actor* berperan sebagai pengirim atau penerima messages. *Actor* di ilustrasikan dalam Gambar 2.17



Gambar 2.17 Ilustrasi Actor

2) *Lifeline* : merepresentasikan kehidupan dari *object* selama *sequence* berlangsung, terdapat *object destruction* yang berarti life*line* tidak lagi berinteraksi. Bentuk dari *lifeline* di ilustrasikan dalam Gambar 2.18

Gambar 2.18 Ilustrasi Lifeline

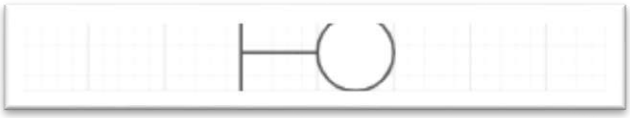
3) *Messages* : menyampaikan informasi dari satu *object* ke *object* yang lain. Bentuk dari *messages* di ilustrasikan dalam Gambar 2.19

Gambar 2.19 Ilustrasi Message

4) *Object Destruction* : simbol x dibawah *lifeline* yang merpresentasikan akhir dari *lifeline*. Bentuk dari *object* destruction di ilustrasikan dalam Gambar 2.20

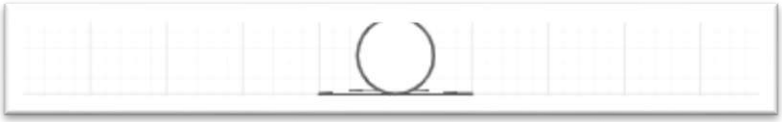
Gambar 2.20 Ilustrasi Object of Destruction

5) *Boundary: Boundary* digunakan untuk menangkap interkasi dari pengguna, alur layar, dan interaksi. Bentuk dari *Boundary* di ilustrasikan dalam Gambar 2.22



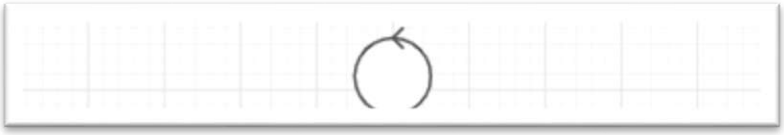
Gambar 2.21 Ilustrasi Boundary

6) *Entity: Entity* biasanya merupakan *object* yang bertugas dalam melakukan penyimpanan informasi dari sistem. Bentuk dari *Entity* di ilustrasikan dalam Gambar 2.21



Gambar 2.22 Ilustrasi Entity

7) *Control: control* bertugas untuk mengorganisir dan mengatur aktifitas dan elemen lain. Bentuk dari *control* di ilustrasikan dalam Gambar 2.23

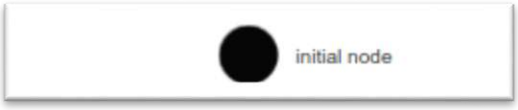


Gambar 2.23 Ilustrasi Control

c. Activity Diagram

Activity Diagram merupakan rancangan aliran aktivitas dalam sebuah sistem yang berjalan. *activity diagram* juga digunakan untuk mendefinisikan atau mengelompokan alur tampilan dari sistem tersebut. *activity diagram* memiliki beberapa elemen diantaranya :

1) *Initial State* : merepresentasikan aksi awal dari activity diagram. Bentuk dari *initial state* di ilustrasikan dalam Gambar 2.24

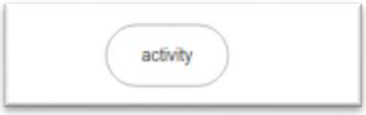


Gambar 2.24 Ilustrasi Initial State

2) *Final State* : merepresentasikan akhir dari activity diagram, Bentuk dari final state di ilustrasikan dalam Gambar 2.25

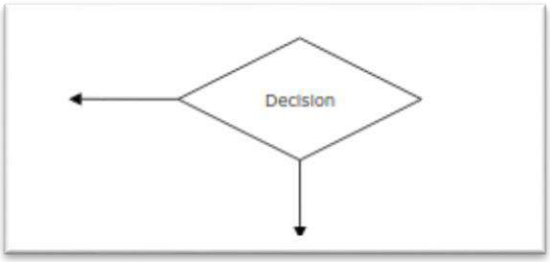
Gambar 2.25 Ilustrasi Final State

3) *activity* : merepresentasikan bagimana masing masing kelas antarmuka saling berinteraksi satu sama lain, Bentuk dari activity di ilustrasikan dalam Gambar 2.26



Gambar 2.26 Ilustrasi Activity

4) *Decision* : digunakan untuk menggambarkan adanya suatu keputusan atau tindakan yang harus diambil pada kondisi tertentu, Bentuk dari *decison* di ilustrasikan dalam Gambar 2.27



Gambar 2.27 Ilustrasi Decision

5) *Action Flow* : menggambarkan bagiamana transisi dari satu elemen ke elemen lain, Bentuk dari action flow di ilustrasikan dalam Gambar 2.28

Gambar 2.28 Ilustrasi Action Flow