BAB II

TINJAUAN PUSTAKA

II.1. Perancangan

Perancangan adalah suatu tahapan yang memiliki tujuan untuk mendesign sistem baru yang dapat menyelesaikan masalah-masalah yang dihadapi perusahaan yang diperoleh dari pemilihan alternatif sistem yang terbaik. **[8]**

Sedangkan definisi perancangan menurut *John Burch* dan *Gary Grudnitski* yang telah diterjemahkan oleh Jogiyanto Hartono (2005:196) dalam buku yang berjudul *Analisis dan Desain Sistem Informasi* menyebutkan sebagai tahapan setelah analisa siklus pengembangan sistem, pendefinisian dari kebutuhan-kebutuhan fungsional dan persiapan untuk rancang bangun implementasi, serta menggambarkan bagaimana suatu sistem dibentuk.

Desain atau perancangan dalam pengembangan perangkat lunak merupakan upaya untuk mengkonstruksi sebuah sistem yang memberikan kepuasan (mungkin informal) akan spesifikasi kebutuhan fungsional memenuhi target, memenuhi kebutuhan secara implisit dan eksplisit dari segi performansi maupun penggunaan sumber daya, kepuasan batasan pada proses desain dari segi biaya, waktu, dan perangkat. **[1]**

Berdasarkan penjelasan diatas penulis dapat mengambil kesimpulan bahwa perancangan merupakan kegiatan mendesign sistem baru yang bertujuan untuk menyelesaikan masalah yang dihadapi perusahaan atau suatu kegiatan yang memiliki tujuan untuk mendesign sistem baru yang dapat menyelesaikan masalah-

masalah yang dihadapi perusahaan yang diperoleh dari pemilihan alternatif sistem yang terbaik.

II.2. Aplikasi

Aplikasi berasal dari kata *application* yang artinya penerapan, lamaran, penggunaan. Secara istilah aplikasi adalah: program siap pakai yang direka untuk melaksanakan suatu fungsi bagi pengguna atau aplikasi yang lain dan dapat digunakan oleh sasaran yang dituju. **[8]**

Perangkat lunak aplikasi adalah suatu sub kelas perangkat lunak komputer yang memanfaatkan kemampuan komputer langsung untuk melakukan tugas yang diinginkan pengguna. Biasanya dibandingkan dengan perangkat lunak sistem yang mengintegrasikan berbagai kemampuan komputer, tapi tidak secara langsung menerapkan kemampuan tersebut untuk mengerjakan suatu tugas yang menguntungkan pengguna. Contoh utama perangkat lunak aplikasi adalah pengolah kata, lembar kerja, dan pemutar media. **[6]**

II.2.1. Aplikasi *Native*

II.2.1.1. Definisi Aplikasi *Native*

Aplikasi *native* adalah yang secara khusus ditujukan untuk *platform mobile* tertentu dan menggunakan bahasa pemrograman serta perangkat lunak pengembangan sesuai dengan *platform* tersebut. Sebagai contoh, aplikasi *native* android ditulis menggunakan bahasa pemrograman *Java* dan *Tool Eclipse,*

sementara aplikasi *iOS/iPhone* dibuat dengan menggunakan bahasa pemrograman *Objective-C* dan *Tool Xcode.*

II.2.1.2. Kelebihan Aplikasi *Native*

Adapun kelebihan Aplikasi *Native*, antara lain adalah :

1. Performa yang sangat baik karena ditulis secara *native* untuk *platform* spesifik.
2. Mampu mengakses semua fitur perangkat keras *smartphone*, seperti : info *device, accelerator, camera,* kompas, *file* dan lain sebagainya.
3. Menghasilkan antarmuka *look* dan *feel* yang alami dengan sangat baik.

II.2.1.3. Kekurangan Aplikasi *Native*

Selain mempunyai kelebihan, Aplikasi *Native* juga mempunyai kekurangan, antara lain adalah :

1. Pengembangan yang tidak mudah karena menggunakan lingkungan, bahasa dan API *(Appliation Programming Interface)* spesifik.
2. Aplikasi hanya berjalan pada *platform* yang sudah di spesifikasikan diawal pengembangan. Apabila ingin dikembangkan di *platform* lain maka harus ditulis ulang dengan *tool* pengembangan yang sesuai. [Mulyadi : 2010 : 2].

II.3. Radio

Radio merupakan salah satu media yang sejak zaman dulu hingga sekarang masih digunakan sebagai sarana untuk menyampaikan informasi.

Media sendiri berasal dari bahasa Latin *medius* yang secara harafiah berarti tengah, perantara, atau pengantar. **[10].** Sedangkan radio menurut ensiklopedia Indonesia yaitu penyampaian informasi dengan pemanfaatan gelombang elektromagnetik bebas yang memiliki frekuensi kurang dari 300GHz (panjang gelombang lebih besar dari 1 mm). Sehingga media radio dapat diartikan sebagai sebuah pengantar yang memanfaatkan gelombang elektromagnetik untuk menyampaikan informasi. Media radio secara fisik memiliki beberapa kekurangan, diantaranya adalah daya jangkau siaran yang terbatas pada suatu daerah tertentu saja dimana radio tersebut disiarkan, misalnya untuk radio AM di Indonesia yang ditetapkan pada frekuensi 530 kHz – 1600 kHz dengan daya jangkau siaran hanya 200 KM dengan modulasi *mono*, sedangkan untuk siaran radio FM yang ditetapkan pada frekuensi 87,5 MHz – 108 MHz daya jangkaunya terbatas 75 KM dengan modulasi *stereo* **[10]**

Keterbatasan ruang lingkup dan frekuensi pemancar menjadi salah satu kendala bagi stasiun radio ”tradisional” saat ini. Keterbatasan ini, akhirnya memunculkan sebuah ide, bagaimana pengguna/pendengar radio *(user)* bisa mendengarkan radio favoritnya dimana saja dan kapan saja. *Live Radio (Online Streaming)* dibuat untuk mangatasi keterbatasan ini. Melalui *Live Radio (Online Streaming)*, kita tidak hanya bisa mendengarkan radio seperti biasa. Akan tetapi, *user* bisa memanfaatkan *tools* lain seperti kita memanfaatkan *tools* di *website* pada umumnya. *Live Radio (Online Streaming)* menggunakan konsep *audio streaming.* **[12]**

Teknologi *streaming* adalah proses pengiriman data kontinu alias terus-menerus yang dilakukan secara *broadcast* melalui Internet untuk ditampilkan oleh aplikasi *streaming* pada PC (klien). Paket-paket data yang dikirimkan telah dikompresi untuk memudahkan pengirimannya melalui internet. Dengan teknologi radio *streaming,* maka memungkinkan penyiaran radio yang sebelumnya menggunakan pemancar berubah menggunakan *internet protocol,* dan pendengar bisa menggunakan laptop atau PC *(Personal Computer)* sebagai alat untuk mendengarkannya [Puspita Yuni : 2014 : 15].

Streaming berbeda dengan *download. Download* membutuhkan waktu hingga beberapa menit atau jam, sementara *streaming* hanya membutuhkan waktu beberapa detik untuk memulai dan *buffering*. Sebagai contoh, klip musik dengan durasi lima menit dikodekan dengan 128kbps mungkin memerlukan waktu lebih dari 20 menit untuk men- *download* melalui akses internet dengan *bandwidth* 20kbps. Sedangkan dengan *streaming* memungkinkan untuk pemutaran dan penerimaan konten secara bersamaan, tanpa membutuhkan *cache* yang besar. Hal itu dikarenakan *server streaming* akan terus mengirimkan paket *real-time* ke klien sehingga klien tidak perlu menunggu seluruh *file* selesai di*download* untuk mendapatkan konten *streaming. Streaming* merupakan pilihan yang lebih baik untuk ruang kerja yang terbatas seperti ponsel, PDA, dan lain-lain. Dengan keuntungan dari sistem nirkabel ini, *mobile streaming* menjadi layanan media yang menarik [Pujianto, Haris : 2010 : 5].

II.4. Android

Android adalah sistem operasi yang berbasis Linux untuk telepon seluler seperti telepon pintar dan komputer tablet. Android menyediakan *platform* terbuka bagi para pengembang untuk menciptakan aplikasi mereka sendiri untuk digunakan oleh bermacam peranti bergerak. Awalnya, *Google* Inc. membeli Android Inc., pendatang baru yang membuat peranti lunak untuk ponsel. Kemudian untuk mengembangkan Android, dibentuklah *Open Handset Alliance*, konsorsium dari 34 perusahaan peranti keras, peranti lunak, dan telekomunikasi, termasuk *Google*, HTC, Intel, Motorola, Qualcomm, *T-Mobile*, dan Nvidia.

Pada saat perilisan perdana Android, 5 November 2007, Android bersama *Open Handset Alliance* menyatakan mendukung pengembangan standar terbuka pada perangkat seluler. Di lain pihak, *Google* merilis kode–kode Android dibawah lisensi Apache, sebuah lisensi perangkat lunak dan standar terbuka perangkat seluler.

Didunia ini terdapat dua jenis distributor sistem operasi Android. Pertama yang mendapat dukungan penuh dari *Google* atau *Google Mail Services (GMS*) dan kedua adalah yang benar–benar bebas distribusinya tanpa dukungan langsung *Google* atau dikenal sebagai *Open Handset Distribution (OHD).* **[9]**

II.4.1. Android Bagi Komunitas Sumber Terbuka (*Open Source)*

Android memiliki berbagai keunggulan sebagai *software* yang memakai basis kode komputer yang bisa didistribusikan secara terbuka *(open source)* sehingga pengguna bisa membuat aplikasi baru didalamnya. Android memiliki aplikasi *native Google* yang terintegrasi seperti *pushmail* Gmail, *Google* Maps, dan *Google Calendar*. Para penggemar *open source* kemudian membangun komunitas yang membangun dan berbagi Android berbasis *firmware* dengan sejumlah penyesuaian dan *fitur*-fitur tambahan, seperti : FLAC *lossless audio* dan kemampuan untuk menyimpan *download* aplikasi pada *microSD card.* Mereka sering memperbaharui paket-paket *firmware* dan menggabungkan elemen-elemen fungsi Android yang belum resmi diluncurkan dalam suatu *carrier-sanction firmware.* [Syahid : 2012 :35].

II.4.2. *The Dalvik Virtual Machine*

Salah satu elemen kunci dari Android adalah *Dalvik Virtual Machine* (DVM). Android berjalan di dalam DVM bukan di *Java Virtual Machine* (JVM), sebenarnya banyak persamaannya dengan JVM seperti Java ME (*Java Mobile Edition*), tetapi Android menggunakan *Virtual Machine* sendiri yang dapat dikustomisasi dan dirancang untuk memastikan bahwa beberapa fitur – fitur berjalan lebih efisien pada perangkat *mobile*.

DVM adalah *"Register based"* sementara JVM adalah *"stack based"*, DVM didisain dan ditulis oleh Dan *Bornsten* dan beberapa *engineers Google* alinnya. Jadi, bisa dikatakan "*Dalvik equals(Java) == false*". DVM menggunakan

kernel Linux untuk menangani fungsionalitas tingkat rendah termasuk keamanan, *threading*, dan preses serta manajemen memori. Hal ini memungkinkan kita untuk menulis aplikasi C / C++ sama halnya seperti pada OS Linux kebanyakan. Meskipun dalam kenyataannya kita harus banyak memahami arsitektur dan proses sistem dari kernel Linux yang di gunakan Android tersebut.

Semua *hardware* yang berbasis Android dijalankan dengan menggunakan *Virtual Machine* untuk eksekusi aplikasi, pengembang tidak perlu khawatir tentang implementasi perangkat keras tertentu. DVM mengeksekusi *executable file*, sebuah format yang dioptimalkan untuk memastikan memori yang digunakan sangat kecil. *The executable file* diciptakan dengan mengubah kelas bahasa *Java* dan dikompilasi dengan menggunakan *tools* yang disediakan dalam SDK Android [Wahana Komputer : 2012 : 9-10].

II.4.3. Android SDK

Android SDK(*Software Development Kit*) adalah *tools* API(*Appplication Programing Interface*) yang diperlukan untuk memulai mengembangkan aplikasi pada platform Android menggunakan bahasa pemrograman *Java.* Android merupakan subset perangkat lunak untuk ponsel yang meliputi sistem operasi, *middleware* dan aplikasi kunci yang dirilis oleh *Google*. Saat ini disediakan Android SDK (*Software Development Kit*) sebagai alat bantu dan API untuk mulai mengembangkan aplikasi pada *platform* Android menggunakan bahasa pemrograman Java. Sebagai *platform* aplikasi-netral, Android memberi kita kesempatan untuk membuat aplikasi yang kita butuhkan yang bukan merupakan

bawaan ponsel. Beberapa fitur – fitur Android yang paling penting adalah: [**9**]

1. *Framework* aplikasi yang mendukung penggantian komponen dan *reuseable.*
2. DVM dioptimalkan untuk perangkat *mobile*.
3. *Integrated browser* berdasarkan *engine open source WebKit.*
4. Grafis yang dioptimalkan dan didukung oleg *libraries* grafis 2D, grafis 3D berdasarkan spesifikasi openGL ES 1,0 (Optional akselerasi *hardware*)
5. *SQL Lite* untuk menyimpan data
6. *Bluetooth*, EDGE, 3G, dan *Wifi*
7. *Media Support* yang mendukung audio, video, dan gambar (MPEG4, H.262, MP3, AAC, AMR, JPG, PNG, GIF) GSM telepon (tergantung *hardware*)
8. Kamera, GPS, kompas, dan accelerometer (tergantung *Hardware*)
9. Lingkungan *development* yang lengkap dan kaya termasuk perangkat emulator, *tools* untuk *debugging*, profil dan kinerja memori, dan *plugin* untuk IDE *Eclipse*.

II.4.4. Arsitektur Android

Secara garis besar arsitektur Android dapat dijelaskan dan digambarkan sebagai berikut:

1. *Applications* dan *Widgets*

Application dan *Widgets* ini adalah layer dimana kita berhubungan dengan aplikasi saja, dimana biasanya kita *download* aplikasi kemudian kita

lakukan instalasi dan jalankan aplikasi tersebut. Di *layer* terdapat aplikasi inti termasuk klien *email*, program SMS, kalender, peta, *browser*, kontak, dan lain- lain. Semua aplikasi ditulis menggunakan bahasa pemrograman *Java.*

2. Application Frameworks

Android adalah *"Open Development Platform"* yaitu Android menawarkan kepada pengembang atau memberi kemampuan kepada pengembang untuk membangun aplikasi yang bagus dan inovatif. Pengembang bebas untuk mengakses perangkat keras, akses informasi *resources*, menjalankan *service background*, mengatur *alarm*, dan menambahkan status *notification*, dan sebagainya. Pengembang memiliki akses penuh menuju API *framework* seperti yang dilakukan oleh aplikasi yang kategori inti. Arsitektur aplikasi dirancang supaya dapat menggunakan kembali komponen yang sudah digunakan dengan mudah (*reuse*).

Sehingga dapat disimpulkan bahwa *Application frameworks* ini adalah *layer* dimana para pembuat aplikasi melakukan pengembangan/pembuatan aplikasi yang akan dijalankan di sistem operasi Android, karena pada *layer* inilah aplikasi dapat dirancang dan dibuat, seperti *content-providers* yang berupa SMS dan panggilan telepon.

Komponen – komponen yang termasuk di dalam *Application Frameworks* adalah sebagai berikut:

a. Views

b. Content Provider

c. Resource Manager

d. Notification Manager

e. Activity Manager

3. Libraries

Libraries ini adalah *layer* dimana fitur – fitur Android berbeda, biasanya para pembuat aplikasi mengakses *libraries* untuk menjalankan aplikasinya. Berjalan di atas kernel, *layer* ini meliputi berbagai *library* C/C++ inti seperti Libc dan SSL, serta:

– *Libraries* media untuk pemutaran media audio dan video

– *Libraries* untuk manajemen tampilan

– *Libraries Graphics* mencakup SGL dan OpenGL untuk grafis 2D dan 3D

– *Libraries SQLite* untuk dukungan *database.*

– *Libraries* SSL dan Webkit terintegrasi dengan *web browser* dan *security*

– *Libraries LiveWebcore* mencakup modern *web browser* dengan *engine embeded web view*

– *Libraries* 3D yang mencakup implementasi OpenGL ES 1.0 API's

4. Android Run Time

Layer yang membuat aplikasi Android dapat dijalan kan dimana dalam prosesnya menggunakan implementasi Linux. *Dalvik Virtual Machine* (DVM) merupakan mesin yang membentuk dasar kerangka aplikasi Android. Di dalam Android *Run Time* dibagi menjadi dua bagian, yaitu:

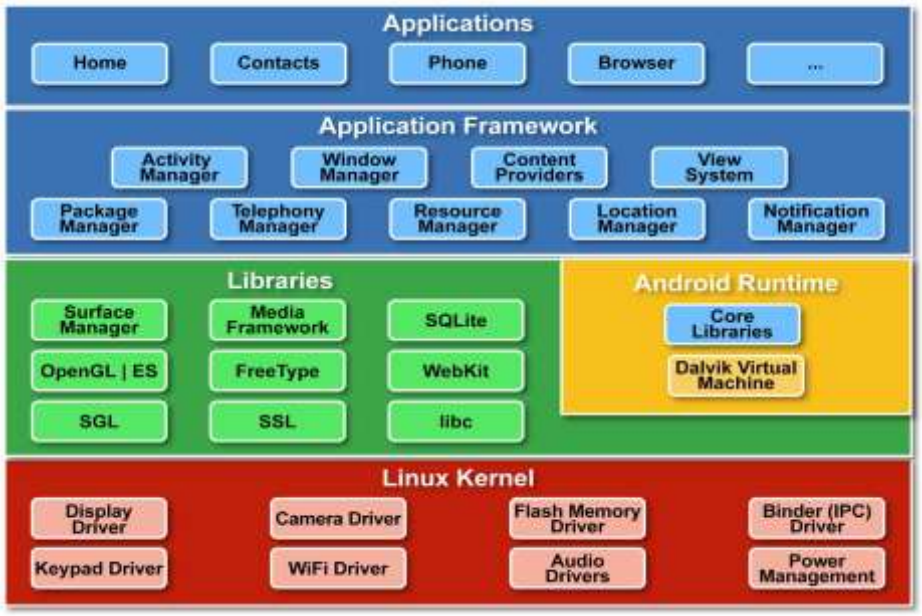
- *Core Libraries*: Aplikasi Android dibangun dalam bahasa *Java*, sementara *Dalvik* sebagai virtual mesinnya bukan virtual mesin *Java*, sehingga diperlukan *libraries* yang berfungsi untuk menterjemahkan

bahasa Java/C yang ditangani oleh *Core Libraries.*

- *Dalvik Virtual Machine*: Virtual mesin berbasis *register* yang dioptimalkan untuk menjalankan fungsi – fungsi secara efisien, dimana merupakan pengembang yang mampu membuat Linux kernel untuk melakukan *threading* dan manajemen tingkat rendah.

5. Linux *Kernel*

Linux *Kernel* adalah *layer* dimana inti dari *operating system* dari Android itu berada. Berisi *file – file system* yang mengatur sistem *processing*, *memory, resource, drivers,* dan sistem – sistem operasi Android lainnya. Linux *Kernel* yang digunakan Android adalah Linux *kernel Release* 2.6 [Wahana Komputer : 2012 : 10-15].



Gambar II.1. Arsitektur Android

(Sumber : Wahana Komputer ; 2012)

II.5. Eclipse

Eclipse adalah sebuah IDE (*Integrated Development Environment*) untuk mengembangkan perangkat lunak dan dapat dijalankan di semua *platform* (*platform-independent*). Berikut ini adalah sifat dari *Eclipse*:

1. *Multi-platform*: Target sistem operasi *Eclips*e adalah Microsoft Windows, Linux, Solaris, AIX, HP-UX dan Mac OS X.
2. *Mulit-language*: *Eclipse* dikembangkan dengan bahasa pemrograman Java, akan tetapi Eclipse mendukung pengembangan aplikasi berbasis bahasa pemrograman lainnya, seperti C/C++, Cobol, Python, Perl, PHP, dan lain sebagainya.
3. *Multi-role:* Selain sebagai IDE untuk pengembangan aplikasi, *Eclipse* pun bisa digunakan untuk aktivitas dalam siklus pengembangan perangkat lunak, seperti dokumentasi, test perangkat lunak, pengembangan web, dan lain sebagainya.

Eclipse pada saat ini merupakan salah satu IDE favorit dikarenakan gratis dan *open source*, yang berarti setiap orang boleh melihat kode pemrograman perangkat lunak ini. Selain itu, kelebihan dari *Eclipse* yang membuatnya populer adalah kemampuannya untuk dapat dikembangkan oleh pengguna dengan komponen yang dinamakan *plug-in.* [Nazruddin Safaat H: 2011 : 12-13]

II.6. Java

Java language Spesification adalah definisi teknis bahasa pemrograman *Java* yang di dalamnya terdapat aturan penulisan sintaks dan semantik *Java.* API adalah *Application Programming Interface* yaitu sebuah *layer* yang *basic class* yang sudah didefinisikan dan antarmuka pemrograman yang akan membantu para pengembang aplikasi dalam perancangan sebuah aplikasi. API memampukan para pengembang untuk dapat mengakses fungsi-fungsi sistem operasi yang di izinkan melalui bahasa *Java.* Pada saat ini dikenal ada tiga buah API dari *Java,* yaitu :

1. J2SE, *Java 2 Standard Edition* adalah API yang digunakan untuk mengembangkan aplikasi-aplikasi yang bersifat *client-server standalone* atau *applet.*
2. J2EE, *Java 2 Enterprise Edition* adalah API yang digunakan untuk melakukan pengembangan aplikasi-aplikasi yang bersifat *server-side* seperti *Java Servlet,* dan *Java Server Pages.*
3. J2ME, *Java 2 Micro Edition* adalah API yang merupakan *subset* dari J2SE tetapi memiliki kegunaan untuk mengembangkan aplikasi pada *handheld device* seperti *Smartphone* atau PDA tentu saja yang didalamnya telah ditanamkan interpreter *Java.*

JDK sebenarnya adalah sekumpulan program kecil yang akan sangat membantu para pengembang aplikasi dalam merancang dan melakukan testing program. JDK harus diakses lewat *command line* [Wahana Komputer : 2012 : 3]

II.7. UML *(Unified Modeling Language)*

UML *(Unified Modeling Language)* merupakan bahasa visual untuk pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram dan teks – teks pendukung untuk menspesifikasikan, menggambarkan, membangun, dan dokumentasi dari sistem perangkat lunak [Munawar : 2005 :12].

II.7.1. Sejarah UML *(Unified Modeling Language)*

Bahasa pemrograman berorientasi objek yang pertama dikembangkan dikenal dengan nama Simula-67 yang dikembangkan pada tahun 1967. Bahasa pemrograman ini kurang berkembang dan dikembangkan lebih lanjut, namun dengan kemunculannya telah memberikan sumbangan yang besar pada developer pengembang bahasa pemrograman berorientasi objek selanjutnya.

Perkembangan aktif dari pemrograman berorientasi objek mulai menggeliat ketika berkembangnya bahasa pemrograman Smalltalk pada awal 1980-an yang kemudian diikuti dengan perkembangan bahasa pemrograman berorientasi objek yang lainnya seperti C objek, C++, Eiffel, dan CLOS. Secara aktual, penggunaan bahasa pemrograman berorientasi objek pada saat itu masih terbatas, namun telah banyak menarik perhatian di saat itu. Sekitar lima tahun setelah Smalltalk berkembang, maka berkembang pula metode pengembangan berorientasi objek. Metode yang pertama diperkenalkan oleh Sally Shaler dan Stephen Mellor (Shaler-Mellor, 1988) dan Peter Coad dan Edward Yourdon (Coad-Yourdon, 1991), diikuti oleh Grady Booch (Booch, 1991), James R. Rumbaugh, Michael R. Blaha, William Lorensen, Frederick Eddy, William

Premerlani (Rumbaugh-Blaha-Premerlani-Eddy-Lorensen, 1991), dan masih banyak lagi [Munawar : 2005 : 12].

Buku terkenal yang juga berkembang selanjutnya adalah karangan *Ivar Jacobson* (Jacobson, 1992) yang menerangkan perbedaan pendekatan yang fokus pada *use case* dan proses pengembangan. Sekitar lima tahun kemudian muncul buku yang membahas mengenai metodologi berorientasi objek yang diikuti dengan buku – buku yang lainnya. Didalamnya juga membahas mengenai konsep, definisi, notasi, terminologi, dan proses mengenai metodologi berorientasi objek.

Karena banyaknya metodologi – metodologi yang berkembang pesat itu, maka muncullah ide untuk membuat sebuah bahasa yang dapat dimengerti semua orang. Usaha penyatuan ini banyak mengambil dari metodologi – metodologi yang berkembang saat itu. Maka dibuat bahasa yang merupakan gabungan dari beberapa konsep seperti konsep *Object Modeling Technique* (OMT) dari Rumbaugh dan Booch (1991), konsep *The Classes, Responsibilities, Collaborators* (CRC) dari *Rebecca Wirfs-Brock* (1990), konsep pemikiran *Ivar Jacobson*, dan beberapa konsep lainya dimana *James R. Rumbaigh*, *Grandy Booch*, dan *Ivar Jacobson* bergabung dalam sebuah perusahaan *Rational Software Corporation* menghasilkan bahasa yang disebut dengan *Unified Modeling Language* (UML) [Munawar : 2005 : 12].

II.7.2. Diagram UML

Berikut akan dijelaskan diagram yang akan digunakan penulis dalam perancangan aplikasi live radio berbasis android ini, yaitu *use case diagram*, *class diagram, sequence diagram,* dan *activity diagram*.

II.7.2.1. *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. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

Use case diagram dapat sangat membantu bila kita sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem.

Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal.

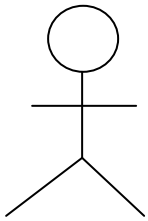
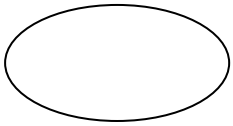
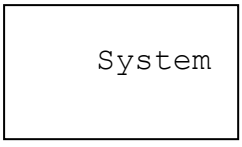
Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common* **.**

Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri.

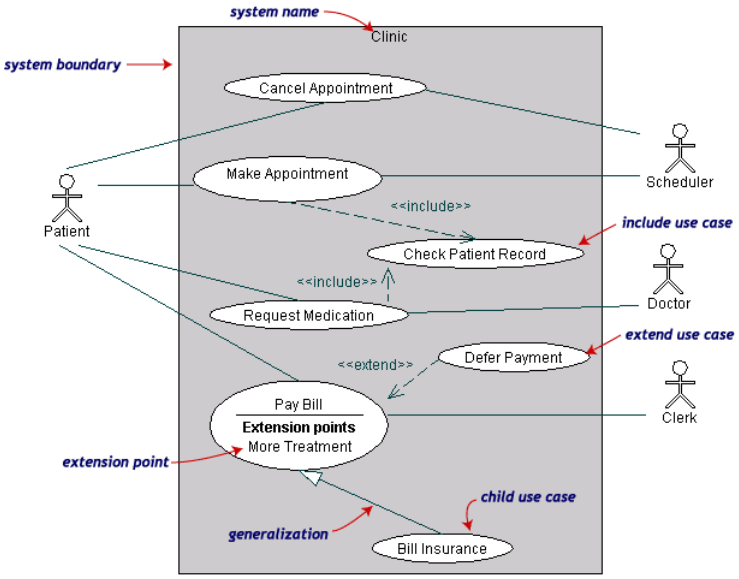
Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain. [Munawar : 2005 : 12-13].

Penjelasan tentang simbol – simbol *use case* dapat dilihat pada tabel II.1.

Tabel II.1 Definisi Simbol *Use Case Diagram*

Simbol	Nama Simbol	Fungsi
	Aktor	Merupakan orang, proses, atau sistem lain yang berinteraksi dengan aplikasi yang akan dibuat di luar aplikasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu orang.
	*Use Case*	Merupakan orang, proses, atau sistem lain yang berinteraksi dengan aplikasi yang akan dibuat di luar aplikasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu orang.
	Association	Merelasikan Aktor dengan *use case.*
	*System Boundary*	Menggambarkan batasan sistem terhadap Lingkungannya.

(Sumber : Munawar ; 2005)



Gambar II.2. Contoh *Use Case Diagram*

(Sumber : Ilmu Komputer ; 2012)

II.7.2.2. *Class Diagram*

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi).

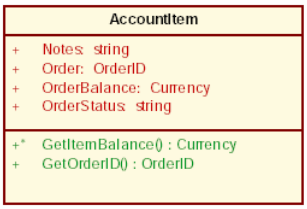
Class diagram menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. [Munawar : 2005 : 12-13].

Class memiliki tiga area pokok :

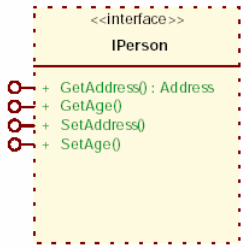
1. Nama (dan *stereotype*)
2. Atribut
3. Metoda

Atribut dan metoda dapat memiliki salah satu sifat berikut :

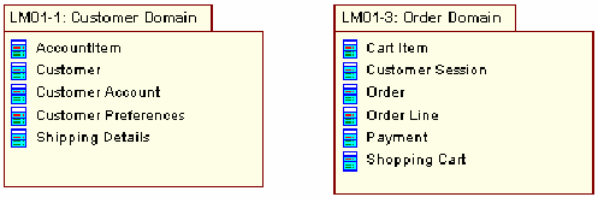
- *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan
- *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya
- *Public*, dapat dipanggil oleh siapa saja



Class dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class.* Dengan demikian *interface* mendukung resolusi metoda pada saat *run-time*.

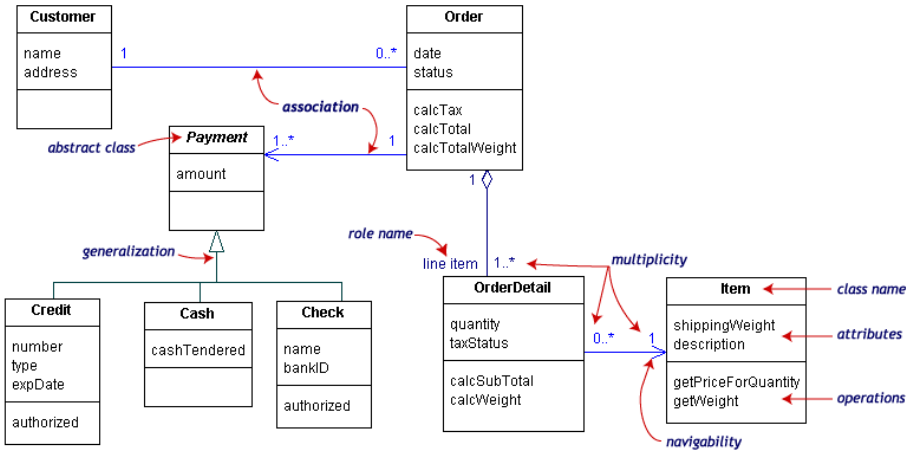


Sesuai dengan perkembangan *class* model, *class* dapat dikelompokkan menjadi *package*. Kita juga dapat membuat diagram yang terdiri atas *package*.



Hubungan Antar *Class*

1. Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah *navigability* menunjukkan arah *query* antar *class*.
2. Agregasi, yaitu hubungan yang menyatakan bagian ("terdiri atas..").
3. Pewarisan, yaitu hubungan hirarkis antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metoda *class* asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.
4. Hubungan dinamis, yaitu rangkaian pesan (*message*) yang di-*passing* dari satu *class* kepada *class* lain. Hubungan dinamis dapat digambarkan dengan menggunakan *sequence diagram* yang akan dijelaskan kemudian.



Gambar II.3. Contoh *Class Diagram*
(Sumber : Ilmu Komputer ; 2012)

II.7.2.3. *Sequence Diagram*

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait).

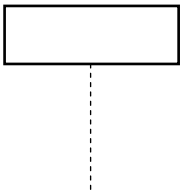
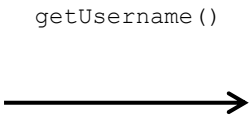
Sequence diagram biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan.

Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal. *Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, *message* akan dipetakan menjadi operasi/metoda dari *class*.

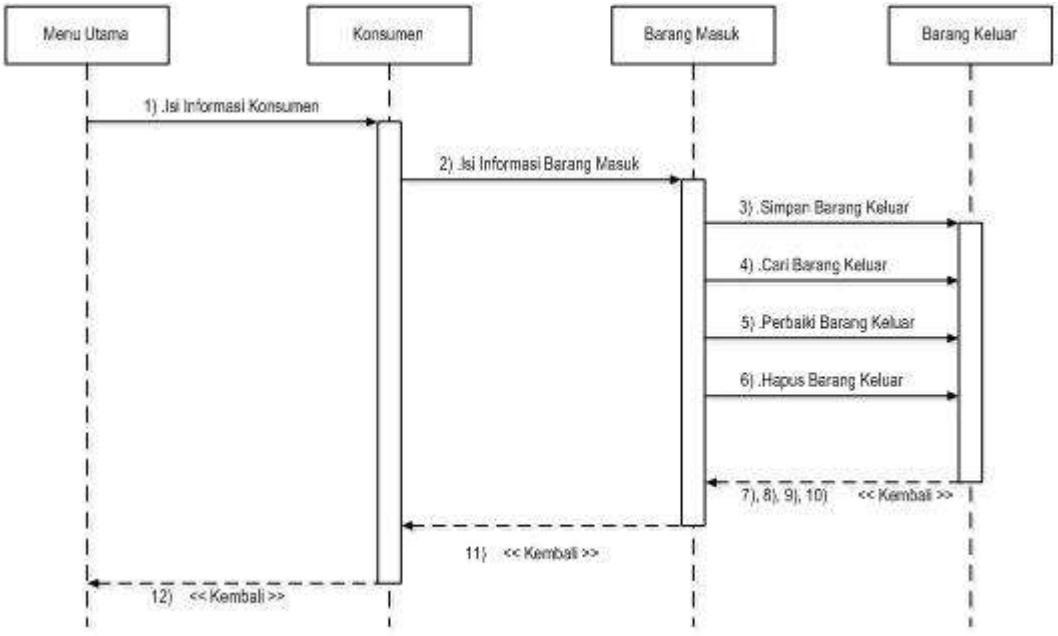
Activation bar menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah message [Munawar : 2005 : 12-13].

Penjelasan tentang simbol–simbol *class diagram* dapat dilihat pada tabel II.2.

Tabel II.2. Definisi Simbol *Sequence Diagram*

Simbol	Nama Simbol	Fungsi
	*Lifelines*	Representasi dari sebuah *class*
	*Message*	Representasi pemanggilan method oleh satu *class* kepada *class* yang lain

(Sumber : Munawar ; 2005)



Gambar II.4. Contoh *Sequence Diagram*

(Sumber : Ilmu Komputer ; 2012)

II.7.2.4. *Activity Diagram*

Activity diagrams menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

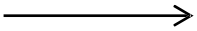
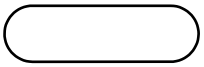
Activity diagram merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan behaviour internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas. Sama seperti *state*, standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan behaviour pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal [Munawar : 2005 : 12-13].

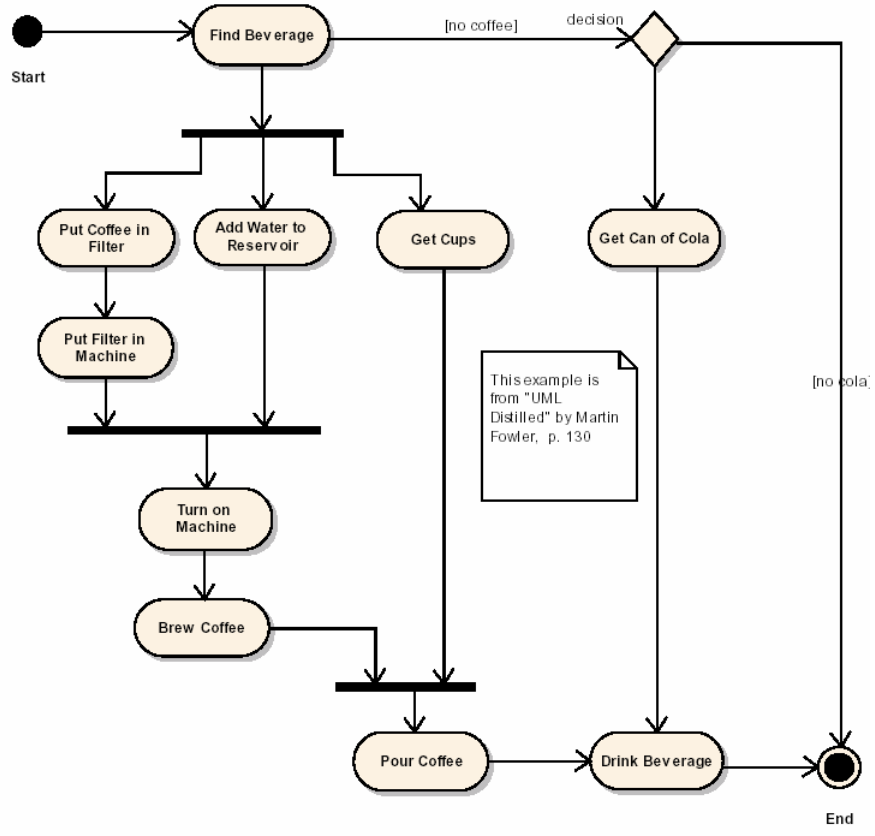
Activity diagram dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu.

Berikut adalah penjelasan tentang simbol – simbol *activity diagram :*

Tabel II.3. Definisi Simbol *Activity Diagram*

Simbol	Nama Simbol	Fungsi
	*Initial State*	Pernyataan awal dari sebuah aktivitas
	*Final State*	Pernyataan akhir dari sebuah aktivitas
	*Transisi*	Menggambarkan alur dari aksi – aksi yang ada pada aktivitas
	Aksi	Menggambarkan aksi yang ada pada aktivitas

(Sumber : Munawar ; 2005)



Gambar II.5. Contoh *Activity Diagram*

(Sumber : Ilmu Komputer ; 2012)