BAB II
TINJAUAN PUSTAKA

2.1 Teori Judul

2.1.1 Pengertian Aplikasi

Menurut Solichin (2016), "Aplikasi adalah bagian yang tidak terpisah dari suatu sistem komputer".

Menurut Santoso (2017:3), "Istilah program dan aplikasi lebih sering disebut untuk menyatakan perangkat lunak. Di kalangan profesional teknologi informasi, istilah program biasa digunakan untuk menyatakan hasil karya mereka yang berupa instruksi-instruksi untuk mengendalikan komputer. Disisi pemakai, hal seperti itu biasa disebut sebagai aplikasi".

Berdasarkan definisi para ahli diatas maka dapat disimpulkan Aplikasi merupakan salah satu bagian dari perangkat lunak yang digunakan untuk melakukan suatu pekerjaan yang diinginkan penggunanya.

2.1.2 Pengertian Monitoring

Menurut Nasir dkk (2013), "*Monitoring* adalah pengumpulan informasi secara teratur yang akan membantu menjaga agar pekerjaan tetep pada jalurnya dan dapat memperingatkan anda ketika terjadi sesuatu yang salah".

Menurut mulyono (2017). "*Monitoring* adalah ketentuan-ketentuan yang disepakati dan diberlakukan selanjutnya sustainability kegiatanya harus terjaga, dalam pelaksanaannya objektivitas sangat diperhatikan dan orientasi utamanya adalah pada tujuan program itu sendiri".

Jadi dapat disimpankan dari beberapa pengertian diatas *monitoring* adalah kegiatan penilaian pola kerja yang dilakukan dengan cara mengkaji maupun mengamati sesuatu kegiatan yang dilaksanakan telah sesuai dengan rencana.

2.1.3 Pengertian Tujuan Sistem Monitoring

Berdasarkan Wahyuniarti (2015), proses monitoring adalah proses rutin pengumpulan data dan pengukuran kemajuan atas objektif program. Memantau perubahan yang fokus pada proses dan keluaran. Monitoring memiliki beberapa tujuan, yaitu

Tujuan monitoring:

1. Mengkaji apakah kegiatan-kegiatan yang dilaksanakan telah sesuai dengan rencana.
2. Mengidentifikasi masalah yang timbul agar langsung dapat diatasi.
3. Melakukan penilaian apakah pola kerja dan manajemen yang digunakan sudah tepat untuk mencapai tujuan kegiatan.
4. Mengetahui kaitan antara kegiatan dengan tujuan untuk memperoleh ukuran kemajuan.
5. Menyesuaikan kegiatan dengan lingkungan yang berubah tanpa menyimpang dari tujuan.

Prinsip dari monitoring atau pemantauan adalah :

1. *Monitoring* harus dilakukan terus-menerus.
2. *Monitoring* harus menjadi umpan terhadap perbaikan kegiatan program organisasi.
3. *Monitoring* harus memberi manfaat baik terhadap organisasi maupun terhadap pengguna produk atau layanan.
4. *Monitoring* harus dapat memotifasi staf dan sumber daya lainnya untuk berprestasi.
5. *Monitoring* harus berorientasi pada peraturan yang berlaku.
6. *Monitoring* harus obyektif.
7. *Monitoring* harus berorientasi pada tujuan program.

2.1.4 Pengertian Checklist

Menurut Herdiansyah (2009:136). "*Checklist* merupakan suatu metode dalam observasi yang mampu memberikan keterangan mengenai muncul atau tidaknya perilaku yang diobservasi dengan memberikan tanda cek jika perilaku yang diamati muncul".

Menurut Herdiansyah (2009:136). "*Checklist* adalah salah satu alat *observasi*, yang ditunjukan untuk memperoleh data, berbentuk daftar berisi faktor-faktor berikut subjek yang ingin diamati oleh *observer*, di mana observer dalam pelaksanaan *observasi* dilapangan tinggal memberi tanda *check* (cek, atau biasanya dicentang) pada *list* faktor-faktor sesuai perilaku sebjek yang muncul, di lembar *observasi*, sehingga memungkinkan observer dapat melakukan tugasnya secara cepat dan objektif, sebab *observer* sudah "membatasi diri" pada ada–tidaknya aspek perbuatan subjek, sebagaimana telah dicantumkan didalam list".

Checklist merupakan suatu pencatatan yang bersifat sangat selektif karena berisi suatu daftar kriteria yang spesifik dan dibatasi padahal-hal yang bersifat *observable* (dapat diamati tingkah lakunya) serta harus dijawab dengan 'YA" atau 'TIDAK". *Checklist* juga biasanya digunakan bersama-sama dengan metode pencatatan lain agar dapat mendokumentasikan dengan baik hati atau area yang spesifik tersebut.

2.1.5 Pengertian Peralatan

Menurut karim dkk (2003), "Peralatan adalah mencangkup setiap benda atau alat yang dapat digunakan untuk memperlancar aktivitas atau pekerjaan pemerintah daerah. Peralatan yang baik (harus praktis, efisien dan efektif) dalam hal ini jelas diperlukan bagi terciptanya suatu pemerintahan yang baik seperti peralatan kantor, alat-alat komunikasi, transportasi, dan sebagainya.

Jadi dapat disimpulkan peralatan adalah alat-alat yang memiliki fungsi masing-masing untuk membantu mempermudah pekerja atau memperlancar aktivitas dalam menyelesaikan segala kegiatannya.

2.1.6 Pengertian Scadatel

Menurut PT. PLN, "Scadatel adalah peralatan yang berfungsi mulai pengambilan data pada peralatan pembangkit atau gardu induk pengolahan informasi yang diterima sampai reaksi yang timbul dari hasil pengolahan informasi agar operasi bekerja dengan baik".

2.1.7 Pengertian Android

Menurut Silvia dkk (2014:2), "*Android* adalah *platformopen source* yang komprehensif dan dirancang untuk *mobile devices*. Dikatakan komprehensif karena *Android* menyediakan semua *tools* dan *frameworks* yang lengkap untuk pengembangan aplikasi pada suatu *mobile device*. Sistem *Android* menggunakan *database* untuk menyimpan informasi penting yang diperlukan agar tetap tersimpan meskipun *device* dimatikan". Sedangkan menurut Vavru dan Ujbanyai (2014:9): "*Android is an extensive operating system created by Google, based on open source platform. It is computer software with open source code*. (*Android* adalah sistem operasi yang luas yang dibuat oleh *Google*, berdasarkan pada *platform open source*. Ini adalah perangkat lunak komputer dengan kode sumber terbuka)".

Menurut Andi (2018), "Android adalah nama dari salah satu varian sistem operasi yang berbasis karnel linux yang khusus diperuntukkan untuk perangkat telepon seluler dengan fitur layar sentuh (tounchscreen)".

Android dari awal *lounching* sampai sekarang mengeluarkan beberapa versi dari sistem operasinya. Uniknya, penamaan versi dari Sistem Operasi *Android* menggunakan penamaan berbagai macam jenis kue. Beberapa versi Sistem Operasi *Android* yang telah dikeluarkan:

1. Versi 1.5 *Android Cupcake.*
2. Versi 1.6 *Android Donut.*
3. Versi 2.0.-2.1. *Android Éclair.*
4. Versi 2.2 *Android Frozen – Yoghurt (FroYo).*
5. Versi 2.3. *Android Gingerbread (GB).*
6. Versi 3.0-.2. *Android Honeycomb.*
7. Versi 4.0.*Android Ice-Cream-Sandwich (ICS).*
8. Versi 4.1-4.3. *Android JellyBean (JB).*
9. Versi 4.4. *Android KitKat.*

Peningkatan versi ke versi lain tentu disertai dengan penambahan fungsi yang spesifik. Meskipun *android* telah mengeluarkan beberapa versi dan menguasai pasar, tapi bukan berarti *android* tidak pernah mengeluarkan sistem *operasi* yang gagal. *Honeycomb* merupakan *android* yang gagal. Sistem Operasi *Android* yang pada awalnya ini dikhususkan sebagai OS Tablet PC ini minim sekali dukungan dari berbagai pengembang. *Google* pun merilis *Ice Cream Sandwich* (ICS) yang multifungsi yakni bisa mendukung *smartphone* dan tablet.

Berdasarkan defini diatas android adalah sistem operasi untuk perangkat lunak bergerak layar sentuh seperti telepon pintar dan komputer".

2.1.8 Pengertian *Android Studio*

Menurut Yudhanto dan Wijayanto (2017). "*Android Studio* merupakan sebuah software tools Intergrated Development Environment (IDE) untuk platform Android".

Menurut Juansyah (2015). "*Android Studio*adalah IDE *(Integrated Development Environment)* resmi untuk pengembangan aplikasi *Android* dan bersifat *open source* atau gratis. Peluncuran *Android Studio* ini diumumkan oleh *Google* pada 16 mei 2013 pada *event Google I/O Conference* untuk tahun 2013.

Sejak saat itu, *Android Studio* mengantikan *Eclipse* sebagai IDE resmi untuk mengembangkan aplikasi*Android*".

Sumber : *https://en.wikipedia.org/wiki/Android_Studio*

Gambar 2.1 *Logo Android Studio*

Android studio sendiri dikembangkan berdasarkan *IntelliJ IDEA* yang mirip dengan *Eclipse*disertai dengan ADT plugin *(Android Development Tools). Android studio* memiliki fitur :

a. Projek berbasis pada *Gradle Build*.
b. *Refactory* dan pembenahan *bug* yang cepat.
c. *Tools* baru yang bernama *"Lint"* dikalim dapat memonitor kecepatan, kegunaan, serta kompetibelitas aplikasi dengan cepat.
d. Mendukung*Proguard And App-signing* untukkeamanan.
e. Memiliki GUI aplikasi android lebih mudah.
f. Didukung oleh *Google Cloud Platfrom* untuk setiap aplikasi yang dikembangkan.

Android Studio berbasis *IntelliJ IDEA* dari *JetBrains* dan menggunakan Bahasa Java. Selain *code editor* dan *tools* pengembangan *IntelliJ* yang kuat, *Android Studio* menawarkan lebih banyak fitur yang meningkatkan produktivitas ketika membangun aplikasi *Android* seperti:

1. *Build system* berbasis *Gradle* yang fleksibel.
2. Emulator yang cepat dan kaya fitur.
3. Lingkungan terpadu yang dapat digunakan untuk mengembangkan aplikasi untuk semua perangkat *Android*.
4. Alat pengujian dan *framework* yang ekstensif.
5. *Instant Run* untuk menggabungkan perubahan pada aplikasi yang sedang berjalan tanpa membangun APK baru.
6. *Lint* untuk menangkap kinerja, kegunaan, kompatibilitas versi dan masalah lainnya.
7. Template kode dan integrasi GitHub untuk membantu membangun fitur aplikasi umum dan import contoh kode.
8. Mendukung C++ dan NDK.
9. *Built-in support* untuk *Google Cloud Platform*.

2.1.9 Pengertian Java

Menurut Sukamto dan Shalahuddin, (2014:103) Java adalah nama untuk sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada komputer *standalone* ataupun pada lingkungan jaringan. Java 2 adalah generasi kedua dari java *platform*.

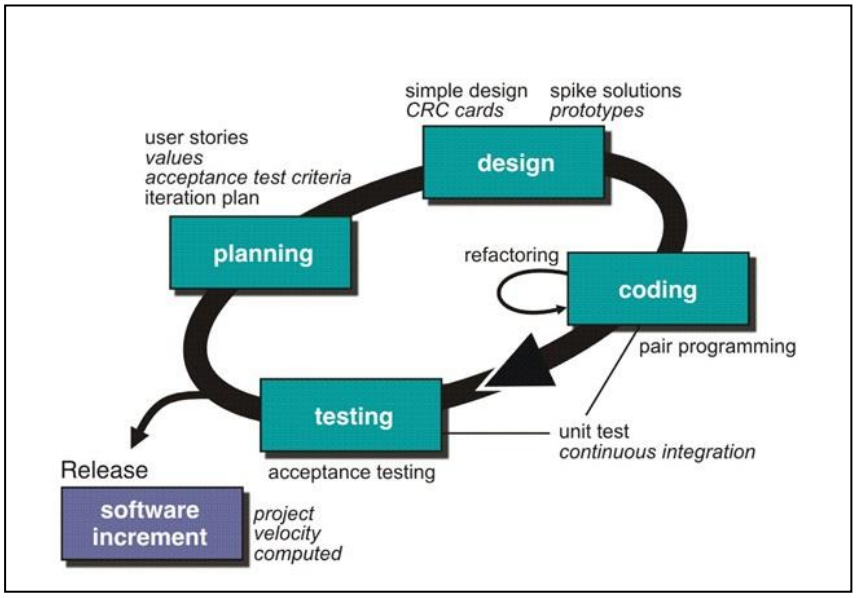
Menurut Sugiarti (2018) Java merupakan bahasa pemrograman tingkat tinggi, namun demikian pemrograman ini bahsanya mudah dipahami karena menggunakan bahasa sehari-hari.

Berdasarkan dari definisi para ahli diatas maka disimpulkan bahwa java merupakan bahasa programan untuk membuat dan menjalankan perangkat lunak pada komputer.

2.1.10 Metode *Extreme Programming*

Menurut Suryantara (2017). "*Extreme Programming* (XP) merupakan salah satu metodologi rekayasa perangkat lunak yang banyak digunakan untuk mengembangkan aplikasi oleh para developer".

Sedangkan menurut Ferdiana dalam (Lubis, 2016). "*Extreme Programming* (XP) dikenal dengan metode atau *"technical how to"* bagaimana suatu tim teknis mengembangkan perangkat lunak secara efisien melalui berbagai prinsip dan teknik praktis pengembangan perangkat lunak. *Extreme Programming* (XP) menjadi dasar bagaimana tim bekerja sehari- hari".



Sumber : *https://astizardiaz.wordpress.com*

Gambar 2.2 *Extreme Programming*

Terdapat empat tahapan yang harus dikerjakan pada metode *extreme programming* (xp) yaitu:

1) *Planning* (Perencanaan).

Tahapan ini merupakan langkah awal dalam pembangunan sistem dimana dalam tahapan ini dilakukan beberapa kegiatan perencanaan yaitu, identifikasi

permasalahan, menganalisa kebutuhansampai dengan penetapan jadwalpelaksanaan pembangunan sistem.

2) *Design* (Perancangan).

Tahapan berikutnya adalah perancangan dimana pada tahapan ini dilakukan kegiatan pemodelan yang dimulai dari pemodelan sistem, pemodelan arsitektur sampai dengan pemodelan basis data. Pemodelan sistem dan arsitektur menggunakan diagram *Unified Modelling Language* (UML) sedangkan pemodelan basis data menggunakan *Entity Relationship Diagram* (ERD).

3) *Coding* (Pengkodean).

Tahapan ini merupakan kegiatan penerapan pemodelan yang sudah dibuat kedalam bentuk *user inteface* dengan menggunakan bahasa pemrograman. Adapun bahasa pemrograman yang digunakan adalah PHP dengan metode terstruktur untuk sistem manajemen basis data menggunakan piranti lunak MySQL.

4) *Testing* (Pengujian).

Setelah tahapan pengkodean selesai,kemudian dilakukan tahapan pengujian sistem untuk mengetahui kesalahan apa saja yang timbul saat aplikasi sedang berjalan serta mengetahui apakah sistem yang dibangun sudah sesuai dengan kebutuhan pengguna. Metode pengujian yang digunakan pada tahapan ini adalah metode *blackbox testing*, dimana pengujian yang dilakukan terhadap form beberapa masukkan apakah sudah berjalan sesuai dengan fungsinya masing-masing.

2.1.11 Prinsip Dasar Extreme Programming

Terdapat lima prinsip dasar yang sangat fundamental dalam *Extreme Programming*, dimana prinsip-prinsip ini digunakan untuk menentukan apakah semua tindakan/pekerjaan yang telah dilakukan akan selalu sukses atau sebaliknya (dalam konteks *Extreme Programming*). Kelima prinsip tersebut adalah:

1. Aliran umpan balik *(Rapid Feedback).*
2. Asumsi kesederhanaan *(Asume Simplicity).*
3. Penambahan perubahan *(Incremental Change).*
4. Pemelukan pekerjaan *(Embrace Work).*
5. Kualitas kerja *(Quality Work).*

2.1.12 Kunci Utama Extreme Programming

Extreme Programming (XP) sebagai sebuah metode yang dinamis diperlihatkan dalam empat values yang dimilikinya dan keempatnya merupakan dasar-dasar yang diperlukan dalam *Extreme Programming* (XP), Kent Beck menyatakan bahwa tujuan jangka pendek individu sering berbenturan dengan tujuan sosial jangka panjang. Karena itu dibuatlah values yang menjadi aturan, hukuman, dan juga penghargaan. Keempat values tersebut adalah:

1. Komunikasi *(Communication)*

   Komunikasi menekankan pada pendekatan yang lebih menekankan pada orang per orang secara langsung, dimana hal tersebut akan lebih baik daripada hanya berdasarkan pada dokumen yang menjelaskan tentang software yang dibangun. Dan juga, kemungkinan penggunaan dari beberapa praktis XP sehingga dibutuhkan alokasi waktu yang banyak untuk berkomunikasi dengan customer.

2. Kesederhanaan *(Simplicity)*

Simplicity adalah sebuah nilai *(value)* dari XP yang digunakan untuk memberikan solusi dari *problem* atau permasalahan yangdihadapi oleh *customer* sehingga *problem* atau masalah tersebut bisa disederhanakan. Kedua pihak yaitu *developer* dan *customer* bisa mengerti solusi *software* jika itu tidak nyata dimana solusi tersebut merupakan kandidat untuk praktis yang lain yang disebut dengan refactoring (salah satu inti dari praktis XP).

3. Umpan Balik *(Feedback)*

Feedback berarti bahwa segala sesuatu yang telah dilakukan atau dicapai dievaluasi dengan respek/reaksi untuk mengetahui bagaimana agar pekerjaan tersebut berjalan dengan baik dan menghasilkan *software* yang sesuai dengan kebutuhan *customer*. Pernyataan bagaimana agar pekerjaan tersebut berjalan dengan baik mengindikasikan bahwa *feedback* diperoleh dari evaluasi setiap bagian pekerjaan dari solusi. *Feedback* yang merupakan hasil kerja dari *Nyquist* di *Laboraturium Bell* pada awal tahun 1930, digunakan untuk menjamin bahwa solusi tersebut adalah benar. Pada saat *engineers* (insinyur) mulai mempelajari *feedback* hal tersebut telah menjadi salah satu jalan didalam mencari solusi dari suatu masalah.

4. Keberanian *(Courage)*

Courage berarti bahwa pihak pengembang mempersiapkan segala sesuatunya untuk membuat keputusan yang sangat penting yang mendukung praktis XP pada saat membangun dan merilis (meluncurkan) *software* kepada *customer* untuk masing-masing iterasi.

2.1.13 Kelebihan Extreme Programming

Adapun kelebihan dari pemakaian *Extreme Programming* dibandingkan dengan model pengembangan perangkat lunak yang lain adalah sebagai berikut: *Extreme Programming* adalah metode yang semi-formal yang artinya metode ini bersifat fleksibel dan perubahan akan selalu diterima oleh *developer* tidak seperti pada metode *Waterfall* yang tidak *fleksibel* sehingga akan sulit mengadaptasinya ketika terjadi perubahan.

2.1.14 Aplikasi Monitoring Checklist Peralatan Supervisory Control Data Acquisition and Telecommunication (SCADATEL) Berbasis Android menggunakan Metode Extreme Programming (XP) pada PT.Perusahaan Listrik Negara (Persero) Unit Induk Penyalur dan Pusat Pengatur Beban di Sumatera Bagian Selatan

Sebuah perangkat lunak dengan cara mengkaji maupun mengamati suatu kegiatan untuk membantu PT. PLN (Persero) P3B unit SUMBAGSEL dalam *memonitoring checklist* peralatan SCADATEL secara cepat, mudah dan akurat menggunakan sistem operasi berbasis *android*.

2.2 Teori Khusus

2.2.1 *Unified Modelling Language*

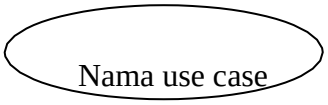
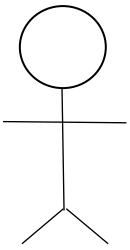
Menurut Sukamto dan Shalahuddin (2018:13), *Unified Modeling Language* (UML) adalah salah satu standar bahasa yang banyak digunakan di dunia industri untuk mendefinisikan *requirement*, membuat analisis dan desain, serta menggambarkan arsitektur dalam pemrograman berorientasi objek. UML merupakan bahasa visual untuk pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram dan teks-teks pendukung. UML hanya berfungsi untuk melakukan pemodelan. Jadi penggunaan UML tidak terbatas pada

metodologi tertentu, meskipun pada kenyataannya UML paling banyak digunakan pada metodologi berorientasi objek.

2.2.2 *Usecase Diagram*

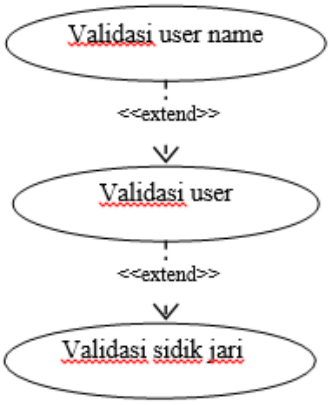
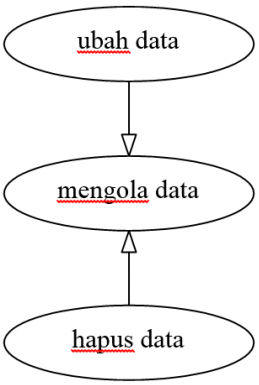
Sukamto (2018:155), *use case* atau diagram *use case* merupakan pemodelan untuk kelakuan (*behavior*) sistem informasi yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi itu. Berikut adalah simbol-simbol yang ada pada diagram *use case* :

Tabel 2.1 Simbol-simbol diagram *use case*

No.	Simbol	Deskripsi
1.	*Use case* 	Fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor, biasanya dinyatakan dengan menggunakan kata kerja diawal frase nama *use case*.
2.	Aktor/*actor* 	Orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang, biasanya dinyatakan menggunakan kata benda

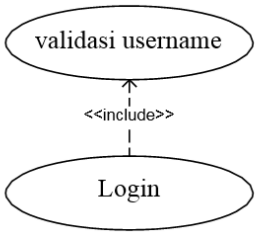
		di awal frase nama aktor.
3.	Assosiasi/*association* _____	Komunikasi antara aktor dan *use case* yang berpartisilpasi pada *use case* atau *use case* memiliki interaksi dengan aktor.
4.	Exstensi/*extend* <<exten d>>	Relasi *use case* tambahan kesebuah *use case* dimana *use case* yang ditambahkan dapat berdiri sendiri walau tanpa *use case* tambahan itu, mirip dengan prinsip *inheritance* pada pemrograman berorientasi objek, biasanya *use case* tambahan memiliki nama depan yang sama dengan *use case* yang ditambahkan, misal

Lanjutan Tabel 2.1 Simbol-simbol diagram *use case*

No.	Simbol	Deskripsi
		 Arah panah mengarah pada *use case* yang ditambahkan, biasanya *use case* yang menjadi *extend*-nya merupakan jenis yang sama dengan *use case* yang menjadi induknya.
5.	Generalisasi/*generalization* 	Hubungan generalisasi dan spesialisasi (umum-khusus) antara dua *buah use case* dimana fungsi yang satu adalah fungsi yang lebih umum dari lainnya, misalnya :  arah panah mengarah pada *use case*

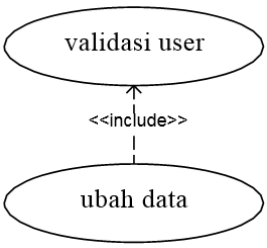
		yang menjadi generalisasinya (umum)

Lanjutan Tabel 2.1 Simbol-simbol diagram *use case*

No.	Simbol	Deskripsi
6.	Menggunakan / *include* / *uses* <<include>> «uses»	Relasi *use case* tambahan ke sebuah *use case* dimana *use case* yang ditambahkan memerlukan *use case* ini untuk menjalankan fungsinya atau sebagai syarat dijalankan *use case* ini Ada dua sudut pandang yang cukup besar mengenai *include* di *use case* : - *Include* berarti *use case* yang ditambahkan akan selalu di panggil saat *use case* tambahan dijalankan, missal pada kasus berikut :  - *Include* berarti *use case* yang tambahan akan selalu melakukan pengecekan apakah *use case* yang di tambahkan telah dijalankan

		sebelum *use case* tambahan di jalankan, misal pada kasus berikut :

Lanjutan Tabel 2.1 Simbol-simbol diagram *use case*

No.	Simbol	Deskripsi
		 Kedua interpretasi di atas dapat dianut salah satu atau keduanya tergantung pada pertimbangan dan interpretasi yang dibutuhkan.

Sumber : Sukamto dan shalahuddin(2018:156-158)

2.2.3 *Activity Diagram*

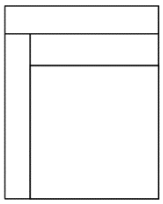
Sukamto (2018:161), diagram aktivitas atau *activity diagram* menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sebuah sistem atau proses bisnis atau menu yang ada pada perangkat lunak. diagram aktivitas menggambarkan aktivitas sistem bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem. Berikut adalah simbol-simbol yang ada pada diagram aktivitas :

Tabel 2.2 Simbol-simbol *activity diagram*

No.	Simbol	Deskripsi
1.	Status awal 	Status awal aktivitas sistem, sebuah diagram aktivitas memiliki sebuah status awal.
2.	Aktivitas 	Aktivitas yang dilakukan sistem, aktivitas biasanya diawali dengan kata kerja.

Lanjutan Tabel 2.2 Simbol-simbol *activity diagram*

No.	Simbol	Deskripsi
3.	Percabangan/*decision* 	Asosiasi percabangan dimana jika ada pilihan aktivitas lebih dari satu.
4.	Penggabungan/join 	Asosiasi penggabungan dimana lebih dari satu aktivitas digabungkan menjadi satu.
5.	Status akhir 	Status akhir yang dilakukan oleh sistem, sebuah diagram aktivitas memiliki sebuah status akhir.

6.	Swimlane 	Memisahkan organisasi bisnis yang bertanggunga jawab terhadap aktivitas yang terjadi.

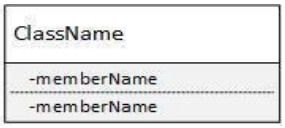
Sumber : Sukamto (2018:162-163)

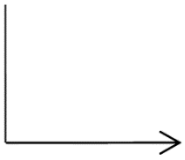
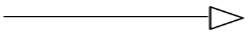
2.2.4 *Class Diagram*

Sukamto (2018:141), diagram kelas atau *class diagram* menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan *method* atau operasi. Berikut penjelasan atribut dan *method* :

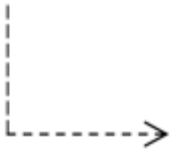
1. Atribut merupakan variable-variabel yang dimiliki oleh suatu kelas.
2. Operasi atau *method* adalah fungsi-fungsi yang dimiliki oleh suatu kelas.

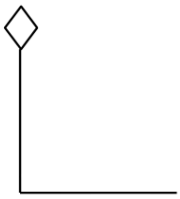
Tabel 2.3 Simbol-simbol *class diagram*

No.	Simbol	Deskripsi
1.	Kelas 	Kelas pada struktur sistem
2.	Antarmuka/*interface*	Sama dengan konsep *interface* dalam pemrograman berorientasi objek

		
3.	Asosiasi/*association* 	Relasi antar kelas dengan makna umum, asosiasi biasanya juga disertai dengan *multiplicity*
4.	Asosiasi berarah/ *directed association* 	Relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan *multiplicity*
5.	Generalisasi 	Relasi antar kelas dengan makna generalisasi – spesialisasi (umum - khusus)

Lanjutan Tabel 2.3 Simbol-simbol *class diagram*

No.	**Simbol**	**Deskripsi**
6.	Kebergantungan/*dependensi* 	Relasi antar kelas dengan makna kebergantungan antar kelas

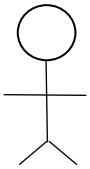
7.	Agregasi/*aggregation* 	Relasi antar kelas dengan makna semua-bagian (*whole-part*)

Sumber : Sukamto (2018:146-147)

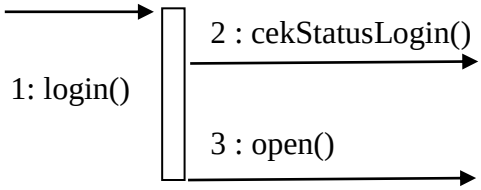
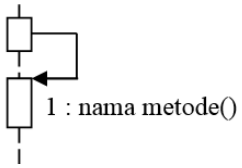
2.2.5 Pengertian *Sequence Diagram*

Sukamto (2018:165), diagram sekuen menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dengan *massage* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambarkan diagram sekuen maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu. Membuat diagram sekuen juga dibutuhkan untuk melihat skenario yang ada pada *use case*. Banyaknya diagram sekuen yang harus digambar adalah minimal sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup dalam diagram sekuen sehingga semakin banyak *use case* yang didefinisikan maka diagram sekuen yang harus dibuat juga semakin banyak. Berikut adalah simbol-simbol yang ada pada diagram sekuen :

Tabel 2.4. Simbol-simbol *sequence diagram*

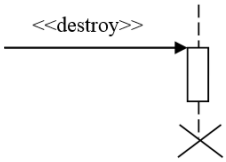
No.	Simbol	Deskripsi
1.	Aktor  Atau Nama aktor Tanpa waktu aktif	Orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat diluar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang, biasanya dinyatakan dalam menggunakan kata benda diawal frase nama aktor.
2.	Garis hidup/*lifeline* 	Menyatakan kehidupan suatu objek
3.	Objek Nama objek : nama kelas	Menyatakan objek yang berinteraksi pesan
4.	Waktu aktif 	Menyatakan objek dalam keadaan aktif dan berinteraksi, semuanya yang terhubungdengan waktu aktif ini adalah sebuah tahapan yang dilakukan di dalamnya, misalnya

Lanjutan Tabel 2.4. Simbol-simbol *sequence diagram*

No.	Simbol	Deskripsi
		 Maka cekStatusLogin() dan open() dilakukan didalam metode login(). Aktor tidak memiliki waktu aktif
5.	Pesan tipe *create* <<create>>	Menyatakan suatu objek membuat objek yang lain, arah panah mengarah pada objek yang dibuat
6.	Pesan tipe *call* 1 : nama_metode()	Menyatakan suatu objek memanggil operasi/metode yang ada pada objek lain atau dirinya sendiri,  Arah panah mengarah pada objek yang memiliki operasi/metode, karena ini memanggil operasi/metode maka operasi/metode yang dipanggil harus ada pada diagram kelas sesuai dengan kelas objek yang berinteraksi
7.	Pesan tipe *send* 1 : masukkan	Menyatakan bahwa suatu objek mengirimkan data/masukkan/informasi ke objek lainnya, arah panah mengarah

		pada objek yang dikirim.

Lanjutan Tabel 2.4. Simbol-simbol *sequence diagram*

No.	Simbol	Deskripsi
8.	Pesan tipe *return* 1 : keluaran 	Menyatakan bahwa suatu objek yang telah menjalankan suatu operasi atau metode menghasilkan suatu kembalian ke objek tertentu, arah panah mengarah pada objek yang menerima kembalian.
9.	Pesan tipe *destroy* 	Menyatakan suatu objek mengakhiri hidup objek yang lain, arah panah mengarah pada objek yang diakhiri, sebaliknya jika ada *create* maka ada *destroy*

Sumber: : Sukamto (2018:165-167)