BAB II

LANDASAN TEORI

2.1. Tata Laksana Laboratorium Komputer

Tata Laksana atau Ketatalaksanaan adalah sistem kerja dalam rangka penyelesaian suatu pekerjaan yang didalamnya memuat tata kerja dan prosedur kerja. Direktorat Jenderal Pendidikan Menengah Kementerian Pendidikan dan Kebudayaan telah mengeluarkan panduan teknis yang mencakup empat kegiatan utama yaitu: perencanaan, pengelolaan, perawatan dan keberlanjutan. Dalam setiap laboratorium terdiri atas koordinator laboratorium, kepala laboratorium, teknisi, dan laboran yang masing-masing mempunyai tugas yang besar dan tanggungjawab yang spesifik.

Mengingat pentingnya peranan laboratorium komputer dalam akselerasi proses pembelajaran, maka perlu dilakukan upaya manajemen laboratorium komputer baik untuk mendukung peran dan fungsi laboratorium secara optimal. Laboratorium komputer yang ideal, setidaknya dilengkapi dengan berbagai alat dan bahan yang dapat mendukung kegiatan laboratorium. Peralatan utama yang harus dimiliki oleh lab komputer adalah komputer, meja komputer, LCD Proyektor, layar proyektor dan papan tulis (*whiteboard*).

Di samping peralatan utama, lab komputer juga semestinya didukung oleh adanya jaringan internet baik yang berbasis kabel maupun nir kabel, sistem jaringan *Local Area Networking* (LAN), *Operation System* (OS) legal, aplikasi *office* legal, anti virus dan sebagainya. (Sutanta, 2005)

2.2. Web Service

Web service adalah komponen perangkat lunak disimpan pada suatu komputer yang dapat diakses oleh aplikasi (atau komponen perangkat lunak lainnya) di komputer lain melalui jaringan. Web service berkomunikasi menggunakan teknologi seperti XML, JSON, dan HTTP. (Deitel,2012)

Web service memiliki layanan terbuka untuk kepentingan integrasi data dan kolaborasi informasi yang bisa diakses melalui internet oleh berbagai pihak menggunakan teknologi yang dimiliki oleh masing-masing pengguna. (Budi, 2008)

Beberapa alasan mengapa digunakannya *web service* adalah sebagai berikut:

1. *Web service* dapat digunakan untuk mentransformasikan satu atau beberapa bisnis logik atau *class* dan objek yang terpisah dalam satu ruang lingkup yang menjadi satu, sehingga tingkat keamanan dapat ditangani dengan baik.
2. *Web service* memiliki kemudahan dalam proses *deployment*, karena tidak memerlukan registrasi khusus ke dalam suatu sistem operasi.

Web service cukup di *upload* ke web server dan siap diakses oleh pihak-pihak yang telah diberikan otorisasi.

3. *Web service* berjalan di *port* 80 yang merupakan protokol standar HTTP, dengan demikian *web service* tidak memerlukan konfigurasi khusus di sisi *firewall*.

2.3. Android

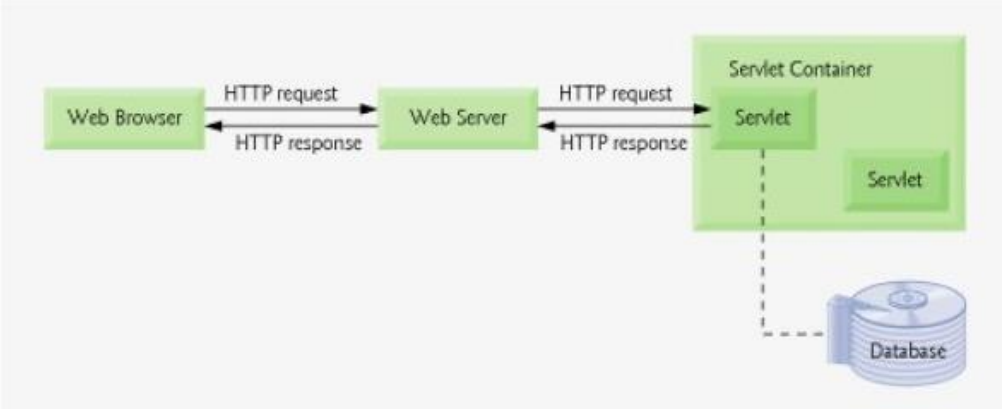
Android adalah sebuah sistem operasi untuk perangkat *mobile* berbasis linux yang mencakup sistem operasi, *middleware* dan aplikasi. Android menyediakan *platform* terbuka bagi para pengembang untuk menciptakan aplikasi mereka. Awalnya, Google Inc. membeli Android Inc. yang merupakan pendatang baru yang membuat peranti lunak untuk ponsel/*smartphone*. 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. (Safaat, 2014)

Aplikasi Android ditulis dalam bahasa pemrograman java. Kode java dikompilasi bersama dengan data *file resource* yang dibutuhkan oleh aplikasi, dimana prosesnya *dipackage* oleh *tools* yang disebut “apt tools” ke dalam paket android sehingga menghasilkan file dengan ekstensi apk. File apk itu yang disebut dengan aplikasi dan nantinya dapat diinstall di perangkat *mobile*.

2.4. Java Servlet

Servlet adalah sebuah *class* dalam bahasa pemgrograman java yang digunakan untuk meningkatkan kapabilitas dari server sebagai *host* dari aplikasi yang diakses melalui *request-response programming* model (diadaptasi dari tutorial J2EE). Servlet adalah sebuah *class* java yang meng-*implement interface* servlet dan menerima *request* yang berasal dari *class* java, *web client*, atau servlet lain yang membangkitkan *response*. "Servlet" juga dipanggil sebagai HTTP Servlet. Hal ini disebakan karena servlet biasanya digunakan dengan HTTP, akan tetap servlet bukanlah merupakan salah satu spesifikasi spesifik dari protokol *client-server*. (Deitel, 2012)

Servlet menunjukkan komunikasi antara *client* dan server melalui protokol HTTP. *Client* mengirimkan permintaan HTTP ke server. Servlet container menerima permintaan dan mengarahkan untuk diproses oleh servlet yang sesuai. Servlet melakukan pengolahan, yang termasuk berinteraksi dengan *database* atau komponen server lain, seperti servlet atau JSP lainnya. Servlet mengembalikan hasilnya kepada *client* di form dari HTML, XHTML atau XML dokumen untuk menampilkan di *browser*. (Deitel, 2012)

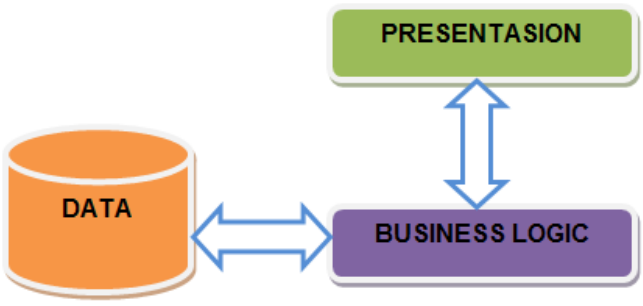


Gambar 2.1. Arsitektur Servlet

2.5. iBATIS

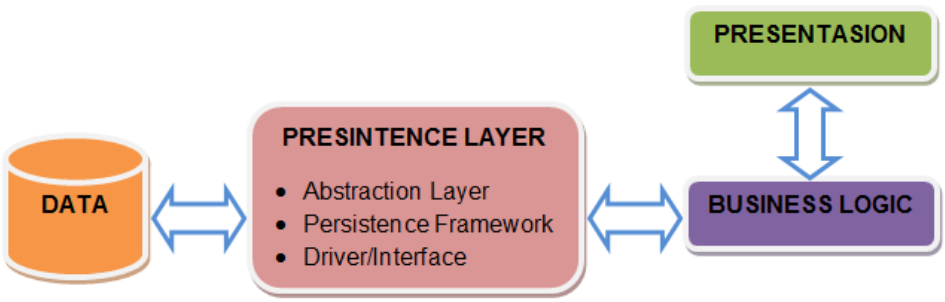
iBATIS merupakan sebuah *persintence framework* yang mengotomatisasi penjembatanan atau pemetaan (*mapping*) antara database SQL dan objek-objek di dalam Java. (Djuandi, 2009)

Pada sebuah sistem aplikasi *client-server* dengan arsitektur *three-tier* dikenal tiga buah lapisan (*layer*) yang disebut sebagai *presentasion, business logic,* dan *data.*



Gambar 2.2 Arsitektur Three-Tier

Seiring dengan berkembangnya evolusi pemrograman dan ditemukannya teknik-teknik yang baru yang berdasarkan pengalaman diakui dapat meningkatkan efisiensi serta kemudahan maka diperkenalkan sebuah lapisan *persintence* yang berada di antara *business logic* dan data sehingga yang semula kedua lapisan itu langsung berinteraksi maka sekarang tersedia jembatan di tengah-tengahnya yang mengatur komunikasi keduanya secara tidak langsung.



Gambar 2.3. Arsitektur dengan Presintance Layer

Lapisan persistence terbagi menjadi tiga bagian dengan masing-masing fungsi sebagai berikut :

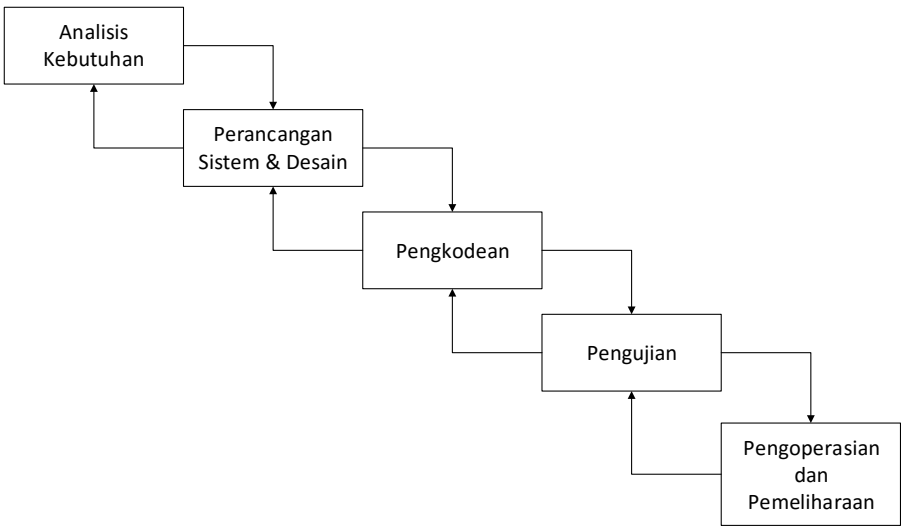
1. Abstraction Layer, berfungsi sebagai antar muka bagi lapisan persistence agar lapisan *business logic* dapat berinteraksi dengannya.
2. Persistence Framework, dalam hal ini adalah iBATIS itu sendiri. Pada prakteknya iBATIS adalah sebuah library (file JAR) yang diasosiasikan dengan sebuah project java sehingga *class-class* di dalam iBATIS dapat digunakan.

3. Driver/Interface, adalah antar muka untuk mengakses *database*. Driver *database* yang digunakan dalam pemrograman java adalah JDBC.

2.6. Model Pengembangan Perangkat Lunak

Model SDLC air terjur (*waterfall*) sering juga disebut model sekuensial linier (*sequential linear*) atau alur hidup klasik (*classic life cycle*). Model air terjun menyediakan pendekatan alur hidup perangkat lunak secara sekuensial atau terurut dimulai dari analisis, desain, pengodean, pengujian, dan tahap pendukung (*support*). (Shalahuddin dan Rosa, 2013)

Pada hal ini Penulis menggunakan model *waterfall* dari proses SDLC karena *waterfall* merupakan SDLC yang bersifat natural. Model *waterfall* melakukan pendekatan secara sistematis dan urut mulai dari *level* kebutuhan sistem lalu menuju ke tahap *analysis*, *design*, *coding*, *testing*, dan *maintenance*.



Gambar 2.4 Model Pengembangan Perangkat Lunak *Waterfall*

Berikut ini adalah tahapan dari model waterfall:

a. Analisis Kebutuhan

Proses menganalisis dan pengumpulan kebutuhan sistem yang sesuai dengan domain informasi tingkah laku, unjuk kerja, dan antar muka (interface) yang diperlukan.

Software merupakan bagian dari sebuah sistem yang besar, maka pengerjaan dimulai dengan mengumpulkan kebutuhan bagi semua elemen-elemen sistem kemudian mengalokasikan beberapa subset dari kebutuhan-kebutuhan tersebut ke *software*. Hal ini sangat penting ketika *software* harus berhubungan dengan elemen lain seperti *hardware*, manusia dan basis data. Tahap ini meliputi pengumpulan kebutuhan pada tingkat sistem dengan sedikit analisa dan perancangan ditingkat atas.

b. Desain

Dalam tahap ini penulis akan merancang desain dan model aplikasi yang akan dirancangan untuk menerjemahkan kebutuhan elemen sistem yang direpresentasikan ke dalam suatu *software* yang diperkirakan kualitasnya sebelum dilakukan pengkodean.

c. Kode

Pengkodean (*coding*) merupakan proses menerjemahkan desain ke dalam suatu bahasa yang bisa dimengerti oleh komputer dengan menerjemahkan ke dalam bentuk yang dapat dibaca oleh

mesin jika perancangan dilaksanakan secara detail. Pengkodean dapat dilakukan secara mekanis.

d. Pengujian

Proses pengujian ini untuk menemukan kesalahan-kesalahan dan memastikan bahwa input yang di buat akan memberikan hasil aktual yang sesuai dengan hasil yang dibutuhkan.

e. Pengoperasian dan Pemeliharaan

Tahap ini merupakan tahapan akhir dalam model *waterfall*. Perangkat lunak yang sudah jadi dijalankan serta dilakukan pemeliharaan (*maintenance*). Pemeliharaan ini termasuk memperbaiki kesalahan yang tidak ditemukan pada langkah sebelumnya. Perbaikan Implementasi unit sistem dan peningkatan jasa sistem sebagai kebutuhan baru. (Herlawati, 2011)

2.7. Unified Modeling Language (UML)

Ada beberapa definisi dari *Unified Modeling Language* (UML) yang pada dasarnya memiliki maksud yang sama. Unified Modeling Language (UML) adalah bahasa grafis untuk mendokumentasikan, menspesifikasikan, dan membangun sistem perangkat lunak. UML adalah bahasa pemodelan untuk menspesifikasikan, memvisualisasikan, membangun dan mendokumentasikan artifak-artifak sistem. (Bambang Hariyanto, Ir., MT.,). Sedangkan menurut Whitten L. Jeffery *Unified Modeling Language (UML)* merupakan satu kumpulan konvensi pemodelan yang digunakan untuk menentukan atau menggambarkan

sebuah sistem software yang terkait dengan objek.Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi *software*, dimana aplikasi tersebut dapat berjalan pada *hardware*, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. *Unified Modeling Language* (UML) biasa digunakan untuk :

1. Menggambarkan batasan sistem dan fungsi-fungsi sistem secara umum, dibuat dengan *use case* dan *actor.*
2. Menggambarkan kegiatan atau proses bisnis yang dilaksanakan secara umum, dibuat dengan *interaction diagrams.*
3. Menggambarkan representasi struktur statik sebuah sistem dalam bentuk *class diagrams.*
4. Membuat model *behavior* ” yang menggambarkan kebiasaan atau sifat sebuah sistem ” dengan *state transition diagrams.*
5. Menyatakan arsitektur implementasi fisik menggunakan *component and development diagrams.*
6. Menyampaikan atau memperluas fungsionalitas dengan *stereotypes*. (Fowler, 2005)

UML berfungsi sebagai jembatan dalam mengkomunikasikan beberapa aspek dalam sistem melalui sejumlah elemen grafis yang bisa dikombinasikan menjadi diagram.

UML mempunyai banyak diagram yang dapat mengakomodasi berbagai sudut pandang dari suatu perangkat lunak yang akan dibangun. Diagram-diagram tersebut digunakan untuk :

1. Mengkomunikasikan ide,
2. Melahirkan ide-ide baru dan peluang-peluang baru,
3. Menguji ide dan membuat prediksi,
4. Memahami struktur dan relasi-relasinya.

Seperti bahasa-bahasa lainnya, UML mendefinisikan notasi dan *syntax* / semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML *syntax* mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya, yaitu diantaranya Grady Booch OOD (*Object-Oriented Design*), Jim Rumbaugh OMT (*Object Modeling Technique*), dan Ivar Jacobson OOSE (*Object-Oriented Software Engineering*). (Booch, 2005)

Terdapat berbagai diagram dalam merancang *system* menggunakan UML, diantaranya adalah :

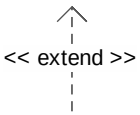
2.8.1. *Use Case Diagram*

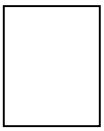
Use Case Diagram secara grafis menggambarkan interaksi antara sistem, sistem eksternal, dan pengguna. Dengan kata lain *Use Case Diagram* secara grafis mendeskripsikan siapa yang akan menggunakan sistem dan dalam cara apa pengguna (*user*) mengharapkan interaksi dengan sistem itu. *Use Case* secara naratif digunakan secara tekstual untuk menggambarkan urutan langkah-langkah dari setiap interaksi.

Dalam pembicaraan tentang *use case*, pengguna biasanya disebut sebagai aktor. Aktor adalah sebuah peran yang bisa dimainkan oleh pengguna dalam interaksinya dengan sistem. (Herlawati, 2011)

Tabel 2.1 Simbol *Use Case Diagram*

NO	GAMBAR	NAMA	KETERANGAN
1		*Actor*	Menspesifikasikan himpuan peran yang pengguna mainkan ketika berinteraksi dengan *use case*.
2		*Use Case*	Deskripsi berdasarkan keperluan aktor, merupakan “apa” yang dikerjakan sistem, bukan “bagaimana” sistem mengerjakannya. *Use case* dibuat berdasarkan keperluan actor.
3		*Association*	Apa yang menghubungkan antara objek satu dengan objek lainnya.

4		*Association 1 arah*	Mengindikasikan bila actor berinteraksi secara ***pasif*** dengan system anda
5		*Generalization*	Hubungan dimana objek anak (*descendent*) berbagi perilaku dan struktur data dari objek yang ada di atasnya objek induk (*ancestor*).
6		*Dependency*	Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri *(independent)* akan mempengaruhi elemen yang bergantung padanya elemen yang tidak mandiri (*independent*).
7		*Include*	Menspesifikasikan bahwa *use case* sumber secara *eksplisit*.
8		*Extend*	Menspesifikasikan bahwa *use case* target memperluas perilaku dari *use case* sumber pada suatu titik yang diberikan.

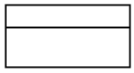
9		*System Boundary*	Menspesifikasikan paket yang menampilkan sistem secara terbatas.
10		*Note*	Elemen fisik yang eksis saat aplikasi dijalankan dan mencerminkan suatu sumber daya komputasi

2.8.2. *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. (Herlawati, 2011)

Tabel 2.2 Simbol *Class Diagram*

NO	GAMBAR	NAMA	KETERANGAN
1		*Class*	Himpunan dari objek-objek yang berbagi atribut serta operasi yang sama.
2		*Directional Assoctiation*	Asosiasi ini menggambar bahwa pesan atau urutan kejadian terjadi dari hanya salah satu kelas sedangkan kelas yang lain pasif
3		*Bidirectional Assoctiation*	Asosiasi ini terjadi ketika salah satu kelas mengirimkan pesan kepada kelas yang lain kemudian kelas yang lain mengirimkan pesan kepada kelas yang mengirimnya pesan.
4		*Generalization*	Hubungan dimana objek anak *(descendent)* berbagi perilaku dan struktur data dari objek yang ada di atasnya objek induk (*ancestor*).
5		*Dependency*	Relasi jenis ini menunjukkan bahwa sebuah kelas mengacu

			kepada kelas lainnya1. Oleh sebab itu perubahan pada kelas yang diacu akan sangat berpengaruh pada kelas yang mengacu
6		*Agregation*	Suatu bentuk relasi yang jauh lebih kuat dari asosiasi. Agregasi dapat diartikan bahwa suatu kelas merupakan bagian dari kelas yang lain namun bersifat tidak wajib.
7		*Composite*	Merupan ralasi yang paling kuat dibandingkan asosiasi dan agregasi. Komposisi diartikan bahwa suatu kelas merupana bagian yang wajib dari kelas yang lain.
8		*Realization*	Operasi yang benar-benar dilakukan oleh suatu objek. Realisasi, bisa disebut juga implementasi merupakan suatu relasi yang menunjukkan penerapan

			terhadap suatu interface kepada sebuah *Class*.

Class memiliki tiga area pokok, yaitu sebagai berikut.

1. Nama *Class*

   Nama *Class* digunakan untuk membedakan antara satu kelas dan kelas yang lain. Nama class menggunakan huruf besar di awal kalimatnya dan diletakkan di atas kotak. Bila class mempunyai nama yang terdiri dari 2 suku kata digabungkan tanpa spasi dengan huruf awal tipa suku kata menggunakan huruf besar.

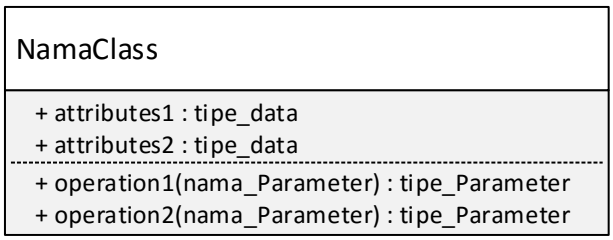
2. *Attribute*

   Attribute adalah *property* dari sebuah *class*. Attribute ini melukiskan batas nilai yang mungkin ada pada obyek dari *class*. Sebuah *class* mungkin mempunyai nol atau lebih *attribute.* Secara konvensi, jika nama *attribute* terdiri dari atas satu suku kata, maka ditulis dengan huruf kecil. Akan tetapi, jika nama *attribute* mengandung lebih dari satu suku kata pertama menggunakan huruf kecil dan awal suku kata berikutnya menggunakan huruf besar.

3. *Method* (*Operation*)

   Operation adalah sesuatu yang bisa dilakukan oleh sebuah *class* atau *class* yang lain dapat dilakukan untuk sebuah *class.* Seperti halnya *attribute* , nama *operation* juga

menggunakan huruf kecil demua jika terdiri dari satu suku kata. Akan tetapi, jika lebih dari satu suku kata pertama menggunakan huruf kecil dan awal suku kata berikutnya menggunakan huruf besar. (Munawar, 2005)



Gambar 2.5 Notasi Class di UML

Pada relasi terdapat suatu penanda yang disebut *multiplicity*. *Multiplicity* ini akan mengindikasikan berapa banyak obyek dari suatu kelas terelasi ke obyek lain. Notasi UML untuk multiplicity ini adalah sebagai berikut : (Herlawati, 2011)

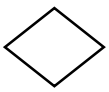
Tabel 2.3 Tabel *Multiplicity*

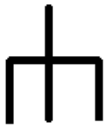
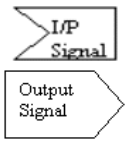
Multiplicity	Arti
*	Banyak
0	Nol
1	Satu, bisa ditulis bisa tidak
0..*	Antara nol sampai banyak
1..*	Antara satu sampai banyak
0..1	Nol atau 1
1..1	Tepat satu

2.8.3. *Activity Diagram*

Activity Diagram secara grafis digunakan untuk menggambarkan rangkaian aliran aktivitas baik proses bisnis maupun *use case*. *Activity diagram* dapat juga digunakan untuk memodelkan *action* yang akan dilakukan saat sebuah operasi di eksekusi, dan memodelkan hasil dari *action* tersebut. *Activity diagram* mempunyai peran seperti halnya *flowchart*, akan tetapi perbedaannya dengan *flowchart* adalah *activity diagram* bisa mendukung perilaku paralel sedangkan flowchart tidak bisa. (Herlawati, 2011)

Tabel 2.4 Simbol *Activity Diagram*

No	Simbol	Nama	Keterangan
1.		***Initial Activity***	sebagai awal dari aktivitas modul sistem aplikasi.
2.		***Activity***	menunjukkan aktivitas yang dilakukan.
3.		***Final Activity***	menunjukkan akhir dari aktivitas.
4.		***Decisions***	menunjukkan aktivitas yang harus dipilih apakah pilihan pertama atau kedua.

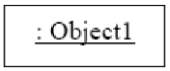
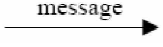
5		***Fork***	Digunakan untuk menunjukkan kegiatan yang dilakukan secara paralel atau untuk menggabungkan dua kegiatan paralel menjadi satu.
6		***Rake***	Menunjukkan adanya dekomposisi
7			Tanda waktu
8		***Flow Final***	Aliran akhir
9		***Signal***	sebagai pengirim dan penerima pesan dari aktivitas yang terjadi. Sinyal terdiri dari sinyal penerima yang digambarkan dengan poligon terbuka dan sinyal pengirim dengan yang digambarkan dengan *convex* poligon.

2.8.4. *Sequence Diagram*

Sequence Diagram secara grafis menggambarkan bagaimana objek berinteraksi dengan satu sama lain melalui pesan pada sekuensi sebuah *use case* atau operasi. Diagram ini

mengilustrasikan bagaimana pesan terkirim dan diterima di antara objek dan dalam sekuensi. (Herlawati, 2011)

Tabel 2.5 Simbol *Sequence Diagram*

No.	Simbol	Nama	Keterangan
1.		***Object***	*Object* merupakan *instance* dari sebuah *class* dan dituliskan tersusun secara horizontal.
2.		***Actor***	*Actor* juga dapat berkomunikasi dengan *object*, maka *actor* juga dapat diurutkan sebagai kolom.
3.		***Lifeline***	*Lifeline* mengindikasikan keberadaan sebuah *object* dalam basis waktu.
4.		***Activation***	*Activation* dinotasikan sebagai sebuah kotak segi empat yang digambar pada sebuah *lifeline.*
5.		***Message***	*Message* digambarkan dengan anak panah horizontal antara *activation*.