BAB 2

LANDASAN TEORI

2. 1. Teori Umum

2.1.1. Interaksi Manusia dan Komputer (IMK)

Menurut Shneiderman & Plaisant (2010, p. 4-5), interaksi manusia dan komputer adalah salah satu disiplin ilmu yang mempelajari hubungan antara manusia dan komputer dengan tujuan merancang, mengevaluasi, mengimplementasikan antarmuka dari sebuah aplikasi agar lebih mudah digunakan oleh manusia.

Menurut Shneider & Plaisant(2010, p. 74-75) dalam perancangan desain antarmuka (*interface*) yang baik, terdapat delapan aturan emas (*eight golden rules*) yang dapat digunakan, yaitu:

1. Berusaha untuk konsisten

   konsistensi dari sebuah urutan aksi harus sama untuk kebutuhan pada situasi yang serupa. Penggunaan *prompt, menu,* dan layar bantu yang identik juga konsistensi warna, *layout,* huruf, dan lain – lain harus konsisten.

2. Memenuhi kemampuan universal

   Harus mengenali kebutuhan dari perbedaan antarmuka dan desain untuk memfasilitasi perubahan isi aplikasi. Perbedaan yang dimaksud antara lain perbedaan antara pengguna ahli dan pemula, umur, perbedaan perkembangan teknologi pada suatu daerah. Perlu menambahkan fitur penjelasan bagi pemula, memberikan *shortcut* bagi pengguna ahli agar dapat bekerja lebih cepat.

3. Memberikan umpan balik yang informatif

   Ada umpan balik untuk setiap aksi yang dilakukan dari *user*. Untuk aksi sederhana dapat memberikan umpan balik yang biasa, untuk aksi yang penting harus diberikan respons yang lebih jelas.

4. Mendesain dialog untuk keadaan akhir

   Urutan aksi harus diorganisir secara jelas ke dalam kelompok permulaan, pertengahan, dan akhir. Umpan balik yang informatif dibutuhkan untuk penyelesaian pada tiap akhir kelompok untuk memberikan operator dari aplikasi mengetahui bahwa aksinya telah berhasil dan siap melanjutkan ke kelompok aksi berikutnya.

5. Mencegah *error*

   Sebisa mungkin mendesain sistem yang tidak memberikan error berupa kesalahan input dari *user*. Sistem harus dapat mendeteksi kesalahan dan memberikan peringatan kepada *user* untuk memperbaiki.

6. Memungkinkan pembalikkan dari aksi sederhana

   Sebisa mungkin aksi harus *reversible*. Fitur ini memberikan rasa aman kepada *user* karena *user* tahu bahwa kesalahan dapat dibatalkan dan memicu *user* untuk mengeksplorasi fungsi program yang belum terlalu *familiar* baginya.

7. Mendukung pusat kendali internal

   User ahli sangat ingin memiliki kuasa atas antarmuka dan antarmuka merespon aksinya. Aksi antarmuka yang mengejutkan, ketidakmampuan untuk mendapatkan atau kesulitan dalam

mendapat informasi yang dibutuhkan akan menyebabkan ketidakpuasan *user*.

8. Mengurangi beban ingatan jangka pendek

   Keterbatasan manusia untuk mengingat dan memproses terlalu banyak hal dalam waktu singkat akan menjadi masalah dari penggunaan sistem. Tampilan harus simpel, pergerakan pada antarmuka sederhana dan juga sesuai agar tidak membuat *user* bingung.

2. 1. 2. Rekayasa Piranti Lunak

*Computer Software (*Piranti Lunak*)* adalah sebuah produk berupa instruksi – instruksi yang berguna untuk memproduksi, mengatur, mendapat, memodifikasi, menampilkan atau mengantarkan informasi yang berupa satu *bit* hingga representasi multimedia dari data yang didapat dari banyak sumber yang berbeda (Pressman, 2011, p. 3).

Menurut Pressman (2011, p. 13) *software engineering* (rekayasa piranti lunak) dapat didefinisikan sebagai pengaplikasian dari sistematika, ilmu, pendekatan terukur untuk mengembangkan , mengoperasikan, memelihara *software* (piranti lunak).

Software Engineering process (proses rekayasa piranti lunak) terdiri atas beberapa layer yaitu:

a quality focus layer menunjukkan bahwa *software engineering* (rekayasa piranti lunak) berlandasan pada kualitas yang baik.

Process layer menunjukkan bahwa *software engineering* memiliki fondasi dasar yang menghubungkan teknologi dan juga pengembangan *software* komputer yang rasional dan tepat waktu. *Process layer* menjadi kerangka yang harus disusun untuk menghasilkan teknologi *software engineering* yang efektif.

Methods layer menyediakan acuan teknis tentang bagaimana membangun *software*. Metode dalam *software engineering* mencakup komunikasi, analisis kebutuhan, desain model, pembangunan program, *testing*, *support*.

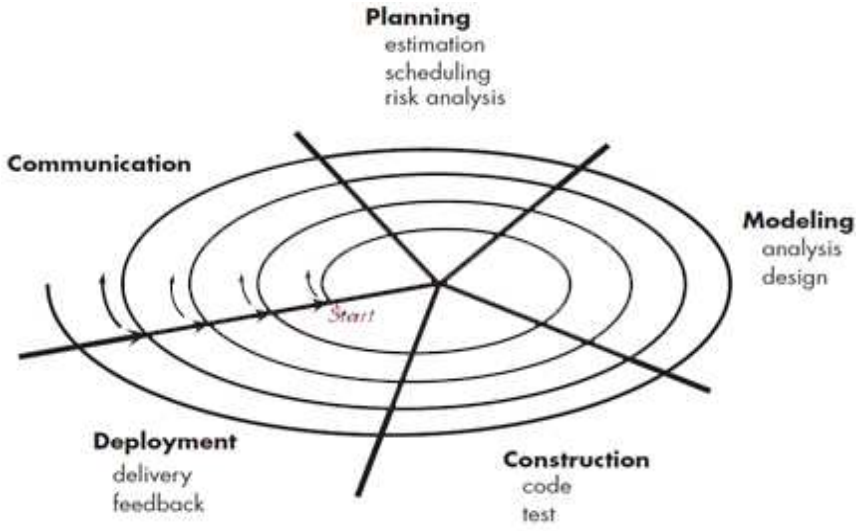
Tools layer berintegrasi dengan dengan informasi untuk menciptakan suatu alat yang dapat digunakan oleh orang lain, sistem untuk mendukung pengembangan *software*.

Gambar 2.1 Software Engineering Layers (Pressman, 2011, p. 14)

2. 1. 3. Software Development Life Cycle

Software development life cycle adalah sebuah siklus hidup dari *software* yang mencakup dari *software* direncanakan hingga *software* tersebut digunakan. Terdapat berbagai macam model untuk *software development life cycle* mulai dari *waterfall, v-model, incremental*

process models, prototyping, spiral model. Pada penelitian ini saya menggunakan *spiral model*.



Gambar 2.2 The Spiral Model (Pressman, 2011, p. 46)

Menurut Pressman (2011, p. 45-46) *spiral model* pertama kali diusulkan oleh Barry Boehm, model ini mendukung pengembangan yang dimulai dari bagian kecil dari *software* terus menerus meningkat hingga mencapai versi lengkap dari *software*.

2. 1. 4. Unified Modeling Language (UML)

Menurut Pressman (2011, p. 13) *Unified Modeling Language* (UML) adalah notasi untuk memodelkan dan pengembangan sebuah *software*.

Menurut Whitten & Bentley(2004) *Unified Modeling Languange* (UML) merupakan satu kumpulan konvensi permodelan yang digunakan untuk menentukan atau menggambarkan sebuah sistem *software* yang terkait dengan objek.

Berdasarkan perspektif dalam memodelkan suatu sistem, UML dapat dikelompokkan menjadi empat kelompok diagram, yaitu: *Use Case Diagram, Class Diagram, Activity Diagram*, dan *Sequence Diagram*.

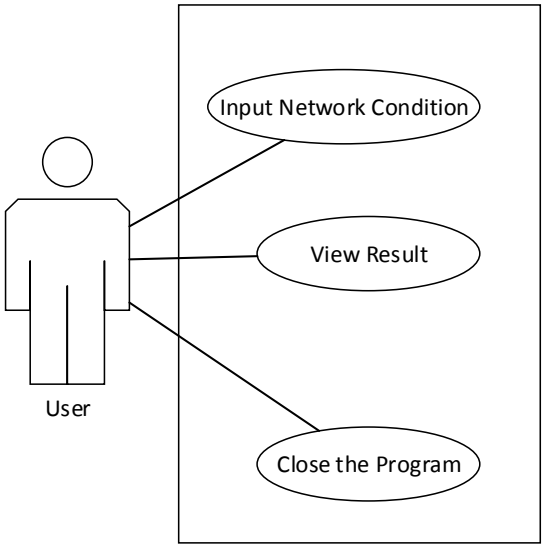
2. 1. 5. Use Case Diagram

Menurut Whitten dan Bentley (2007, p. 247) *use case diagram* adalah urutan iterasi antara sistem dan sistem eksternal dan *user*. *Use case diagram* menggambarkan skenario dan *user* yang saling berhubungan.

Menurut Pressman (2011, p. 134) Jacobson memberikan beberapa pertanyaan yang harus dapat dijawab oleh *use case diagram*

1. Siapa yang menjadi aktor utama dan aktor tambahan?
2. Apa tujuan dari aktor?
3. Apa kondisi yang harus dipenuhi sebelum cerita dimulai?
4. Apa tugas utama dan fungsi utama yang akan dilakukan oleh aktor?
5. Apa pengecualian yang mungkin dipertimbangkan dari cerita yang dideskripsikan?
6. Apa bisa ada variasi dari aksi yang dilakukan aktor?
7. Apa informasi dari sistem dapat dicapai, diproduksi, atau diubah?
8. Apakah aktor akan diinformasikan oleh sistem tentang perubahan lingkungan eksternal?
9. Apa informasi yang diinginkan oleh aktor dari sistem?

10. Apakah aktor berharap diinformasikan tentang perubahan yang tidak terduga?



Gambar 2.3 Contoh Use Case Diagram

Hubungan yang terdapat dalam sebuah *use case diagram* adalah (Whitten & Bentley, 2007: 248-250):

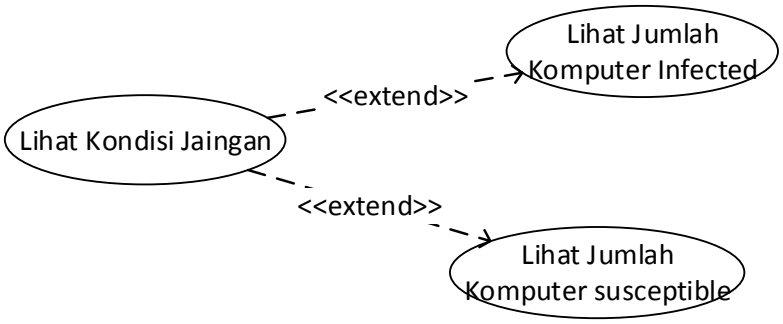
1. *Association*

   Association yang dilambangkan dengan simbol garis dengan, atau tanpa tanda panah melambangkan interaksi antara *use case* dengan *aktor* atau pengguna. Hubungan *Association* bisa dibagi menjadi dua, yaitu *bidirectional association* dan *unidirectional association*. *Bidirectional association* adalah *association* yang bersifat dua arah, dilambangkan dengan sebuah garis lurus sedangkan *unidirectional association* adalah *association* yang bersifat satu arah dan dilambangkan oleh sebuah garis lurus dengan tanda panah.

Gambar 2.4 Contoh *association* pada *use case diagram*

2. *Extends*

   Extends akan digunakan ketika sebuah *use case* cukup rumit dan terdapat beberapa tahap di dalamnya sehingga sulit dimengerti. *Extends* akan menghasilkan use case baru yang mewakilkan suatu fungsi tertentu dari *use case* awal yang nantinya disebut sebagai *extended use case*.

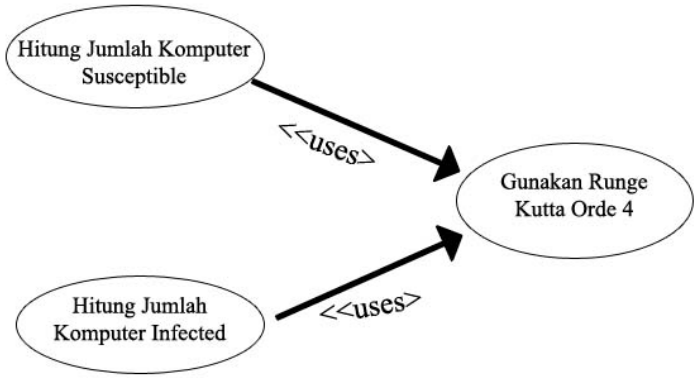


Gambar 2.5 Contoh *Extended Use Case*

3. *Includes / Uses*

   Includes / uses berguna ketika terdapat dua atau lebih *use case* yang melakukan langkah yang sama. Langkah tersebut dipisahkan menjadi sebuah *use case* terpisah yang disebut

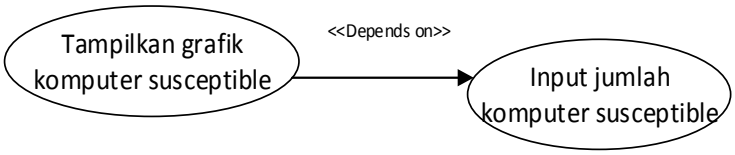
abstract use case. Tujuan dari *abstract use case* adalah untuk mengurangi redundansi pada *use case*.



Gambar 2.6 Contoh *Includes / Uses* pada *Use Case Diagram*

4. *Depends On*

   Depends On adalah hubungan keterkaitan antara sebuah *use case* di mana sebuah *use case* baru bisa dijalankan ketika *use case* yang lainnya sudah dilakukan.

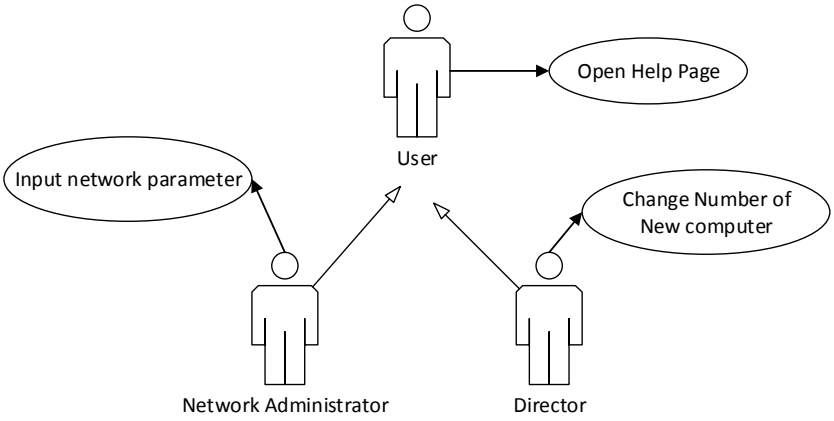


Gambar 2.7 Contoh *Depends On* dalam *Use Case Diagram*

5. *Inheritance*

Inheritance adalah suatu keadaan di mana dua atau lebih aktor melakukan langkah yang sama. *Inheritance* akan membuat

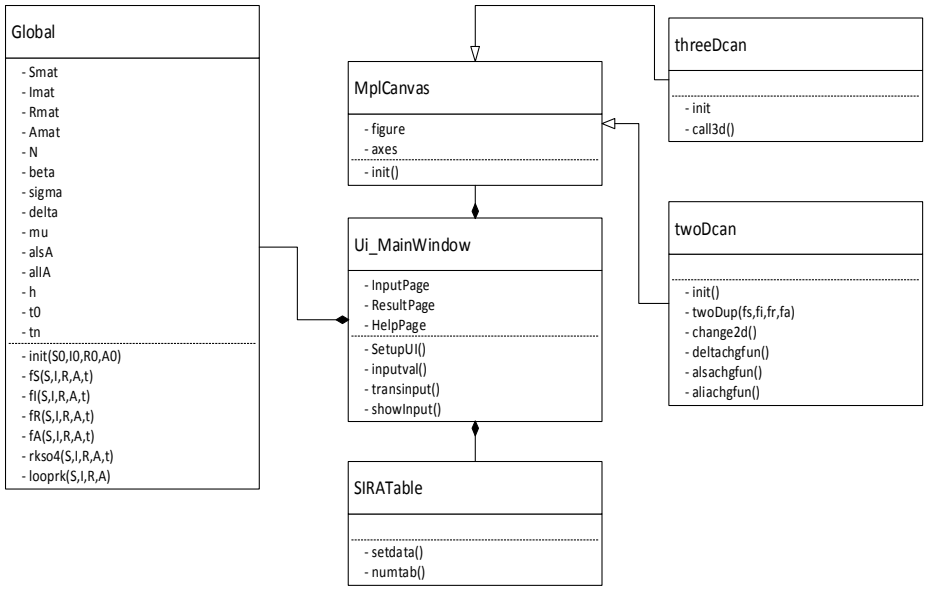
sebuah aktor abstrak yang baru dengan tujuan untuk menyederhanakan *use case*.



Gambar 2.8 Contoh *Inheritance* Dalam *Use Case Diagram*

2. 1. 6. Class Diagram

Class diagram adalah sebuah diagram yang menggambarkan struktur objek statis dari suatu sistem. Diagram ini menunjukkan kelas – kelas dari objek yang membangun sebuah sistem dan juga memaparkan hubungan antarkelas tersebut (Whitten & Bentley, 2007, p. 400).



Gambar 2.9 Contoh *Class Diagram*

Dalam *class diagram* terdapat berbagai notasi, yaitu:

1. Class

Class adalah pedoman dasar untuk merepresentasikan objek (Pressman, 2011, p. 174).

Class terdiri atas 3 bagian, yaitu:

a. *Class name*: Nama dari kelas yang terdapat pada bagian atas dari *class*.

b. *Class attribute*: Atribut dari kelas yang terdapat pada bagian tengah class.

c. *Class behavior*: tingkah laku / fungsi dari kelas yang terdapat pada bagian bawah kelas.

2. Relationship

Relasi yang ada antarkelas. Relasi antarkelas terbagi atas 4 macam, yaitu:

a. *Association*

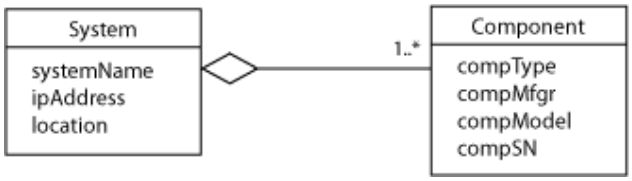
Association (asosiasi) adalah keterhubungan antara satu kelas dengan kelas lainnya. Hubungan antar kelas dilengkapi dengan simbol – simbol yang dituliskan pada asosiasi

Meaning	Symbol	Description	Example
Consist of	=	Introduces the definition of data item	Customer=
And	+	Joins components of the definition in sequence	Customer=name+address
One or more	{ }	Attributes may be repeated; any restrictions on number of repetitions are shown by a subscript	Customer = name+address+$\{phone\}_2$
Zero or more	( )	Attribute is optional	Customer = name+address+$\{phone\}_2$+(email)
Alternatives	[ ]	Selection is indicated by enclosing the alternatives in square brackets[]	Name=[initial | firstname]+surname
Either..or	|	Alternatives for selection in[ ] are separated by a vertical bar	
Specific value	" "	Indicates specific value	"individual", "wholesale"
..	Comment	Comments are enclosed between asterisks	Customer = name+address+$\{phone\}_2$+(email) + ["individual" | "wholesale"] *wholesale customers are entitled to special discounts*

Tabel 2.1 Tabel notasi asosiasi pada class diagram

b. *Aggregation*

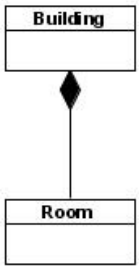
Aggregation adalah suatau relasi yang lebih kuat dari *association*. *Aggregation* dapat diartikan bahwa suatu *class* merupakan bagian dari class lain namun bersifat tidak wajib. (Whitten & Bentley, 2007, p. 379)



Gambar 2.10 Contoh *Aggregation*

c. *Composition*

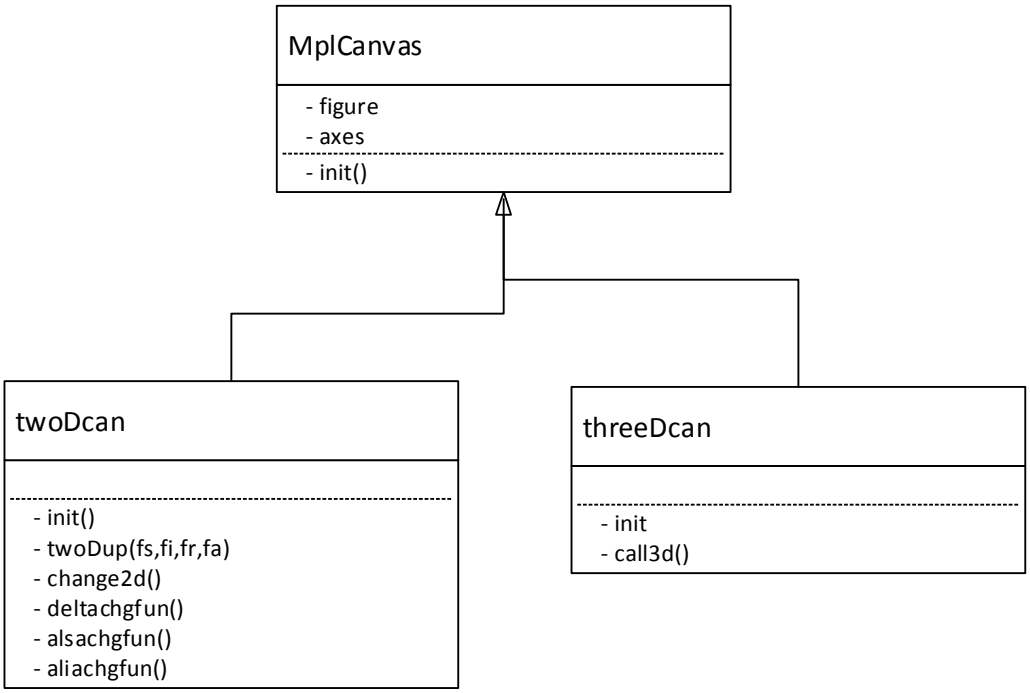
Composition diartikan bahwa suatu *class* merupakan bagian wajib dari *class* lain.



Gambar 2.11 Contoh *Composition*

d. *Generalization*

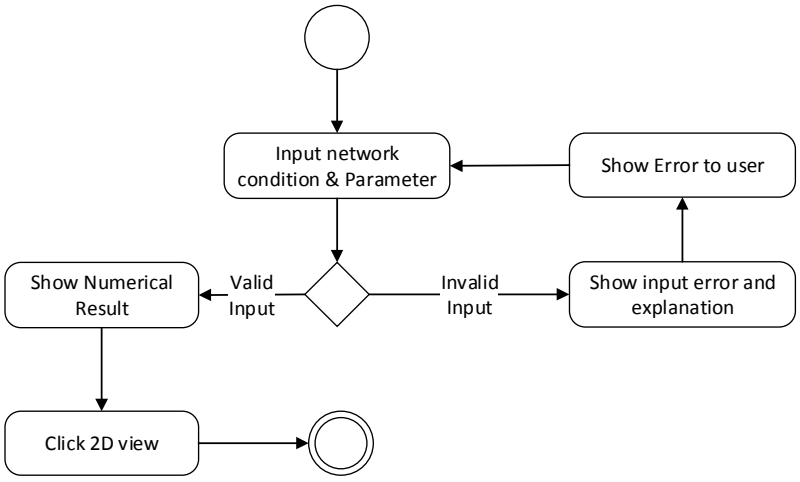
Generalization memungkinkan suatu class mewarisi *attribute* dan *behavior* dari kelas lainnya. *Generalization* digambarkan dengan garis yang memiliki tanda panah tertutup kosong pada salah satu ujungnya mengarah ke *base class* (Whitten & Bentley, 2007, p. 379).



Gambar 2.12 Contoh *Generalization*

2. 1. 7. *Activity Diagram*

Activity diagram adalah diagram yang menggambarkan rangkaian aliran aktivitas baik proses bisnis, langkah – langkah *use case,* dan logika perilaku dari objek. Diagram ini digunakan untuk memodelkan *action* yang akan dilakukan ketika operasi dieksekusi dan hasil dari *action* tersebut (Whitten & Bentley, 2007, p. 390-391)



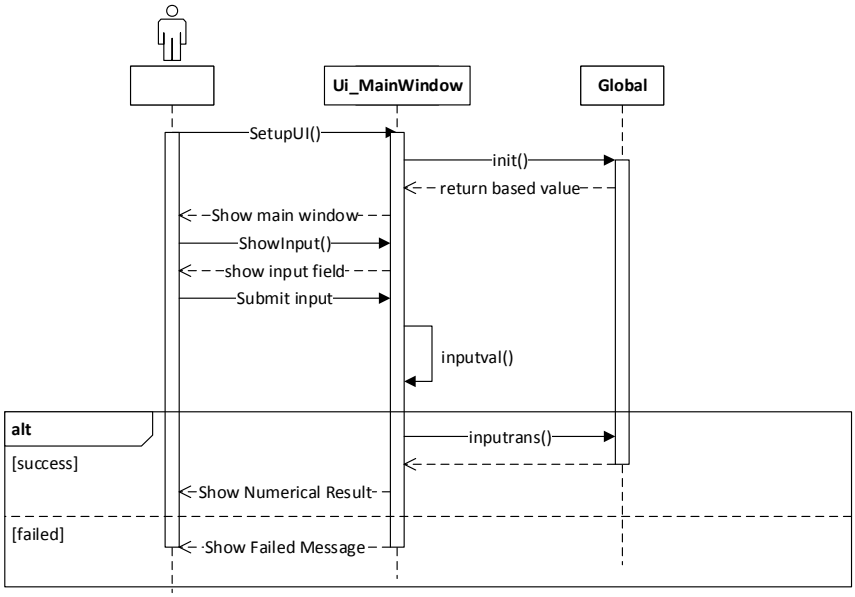
Gambar 2.13 Contoh *Activity Diagram*

Activity diagram memiliki beberapa notasi untuk mengilustrasikan diagram, yaitu:

1. *Initial node:* Lingkaran hitam padat yang merepresentasikan awal proses.
2. *Actions*: Elips yang merepresentasikan langkah / aktivitas.
3. *Flow*: Panah yang mengindikasikan perkembangan sepanjang *actions.*
4. *Decision*: Bentuk permata dengan satu *flow* masuk dan dua atau lebih *flow* keluar. Aliran keluar ditandai untuk mengidentifikasi kondisi.
5. *Merge*: Bentuk permata dengan dua atau lebih *flow* masuk dan satu *flow* keluar. Notasi ini mengkombinasikan *flow* yang sebelumnya dipisahkan menjadi satu *flow*.
6. *Fork*: Balok hitam panjang dengan satu *flow* masuk dan dua atau lebih *flow* keluar. Tindakan paralel pada *flow* paralel dibawah *fork* dapat terjadi dalam urutan apapun atau bersamaan.
7. *Join*: Kotak hitam dengan dua atau lebih *flow* yang masuk dan satu *flow* keluar. Tindakan yang masuk ke dalam *join* harus selesai seluruhnya sebelum proses dilanjutkan.
8. *Activity final*: Lingkaran padat yang merepresentasikan akhir dari proses.

2. 1. 8. *Sequence Diagram*

Sequence diagram adalah diagram UML yang menggambarkan logika dari sebuah *use case* dengan cara menggambarkan bagaimana objek berinteraksi satu sama lain. Diagram ini mengilustrasikan bagaimana pesan dikirim dan diterima antara objek dan dalam urutan apa (Whitten & Bentley, 2007, p. 659).



Gambar 2.14 Contoh *sequence diagram*

Dalam *sequence diagram* terdapat beberapa notasi yang berfungsi untuk mengilustrasikan diagram, yaitu (Whitten & Bentley, 2007, p. 660):

1. *Actor*

   Actor yang berinteraksi dengan objek kelas *interface*.

2. *Interface class*

   Interface class adalah kotak yang mengidentifikasi kode kelas dari tampilan antarmuka.

3. *Controller class*

Setiap *use case* akan memiliki satu atau lebih *controller class*, digambarkan dengan notasi yang sama dengan *interface class.*

4. *Entity classes*

   Entitiy classes adalah kotak tambahan untuk setiap kesatuan yang membutuhkan kolaborasi dari tahap – tahap *sequence*.

5. *Messages*

   Messages dinyatakan dengan garis horizontal utuh yang memanggil *behavior* dari class yang ditunjuk.

6. *Activation bars*

   Activation bars adalah kotak yang berfungsi untuk mengidentifikasi waktu keberadaan dari setiap objek.

7. *Return messages.*

   Return messages adalah jawaban atas pesan yang disampaikan oleh suatu objek.

8. *Self-call*

   Self-call mengizinkan sebuah objek mengirimkan pesan ke objek itu sendiri.

9. *Frame*

   Frame digunakan untuk menandakan area pada diagram yang mengalami pengulangan, seleksi, atau sebuah ketentuan.

2. 1. 9. Python

Python adalah bahasa pemrograman yang bersifat *general-purpose* dan *high level programming language*. Python mendukung berbagai paradigma dalam *programming* termasuk *object-oriented*,

imperative, dan juga *procedural programming*. *Source code* dari python dapat menghasilkan *stand-alone executable program* bagi berbagai sistem operasi yang populer sehingga dapat di jalankan dalam berbagai sistem operasi tanpa perlu meng-*install interpreter* python dalam komputer yang ingin menggunakannya.

2. 1. 10. Python IDLE

Python IDLE (*Integrated Developer Environment*) merupakan salah satu editor yang sudah terintegrasi dengan intepreter bahasa pemrograman python yang umumnya digunakan oleh python *software developer*. IDLE mempermudah *developer* untuk mengembangkan aplikasi yang berbasis python untuk dapat langsung melakukan *trial and error* dari *source code* yang dibuatnya

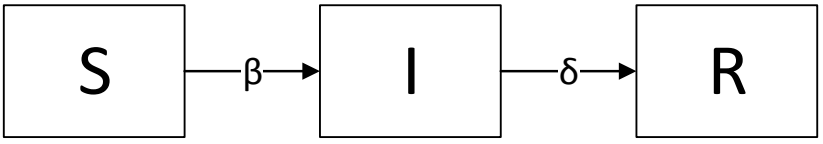
2. 1. 11. PyQt

PyQt adalah salah satu *toolkit* dari Qt yang terikat dengan python. Qt sendiri adalah *toolkit* untuk membuat GUI (*Graphical User Interface*) yang dapat digunakan oleh berbagai bahasa pemrograman. PyQt adalah *software* gratis dan biasanya telah diimplementasikan ke dalam instalasi python sebagai *plug-in*.

2. 2. Teori Khusus

2.2.1. Model Komplementer SIR

Model Komplementer SIR pertama kali ditemukan oleh Willia Ogilvy Kermack dan juga Anderson Gray McKendrick pada tahun 1927 untuk memodelkan kelompok orang sehat dan terkena penyakit. Model tersebut menyatakan tentang pembagian sebuah populasi ke dalam tiga kelompok yaitu S, I, dan R. S(t) menyatakan kelompok *susceptible* (individu yang belum terkena virus tetapi dapat terkena) pada waktu t. I(t) menyatakan kelompok *infected* (individu yang telah terkena virus dan dapat menyebarkan virus ke individu lain), R(t) menyatakan kelompok *removed* (individu yang telah terinfeksi tetapi virus telah diangkat darinya baik karena imunisasi ataupun meninggal), kelompok ini tidak dapat terinfeksi lagi ataupun menyebarkan infeksi kepada individu lain. Pada model SIR aliran perpindahan anggota tiap kelompok dapat dilihat pada gambar 2.15.



Gambar 2.15 Bagan SIR

Kermack dan McKendrick menurukan persamaan diferensial model SIR sebagai berikut:

$$\frac{dS}{dt} = -\frac{\beta SI}{N}$$

$$\frac{dI}{dt} = \frac{\beta SI}{N} \quad \gamma I$$

$$\frac{dR}{dt} = \gamma I \tag{2.1}$$

Dengan jumlah populasi tetap yaitu:

$$N - S(t) + I(t) + R(t)$$

Asumsi yang digunakan pada model ini adalah:

1. Individu antar populasi memiliki kemungkinan terkena virus yang sama, yaitu β.
2. Rata – rata periode infeksi dari virus adalah $\frac{1}{\gamma}$, dengan kata lain γ adalah rata – rata tingkat kematian / penyembuhan.

2.2.2. *Eigenvector* dan *Eigenvalue*

Menurut Boyce, W. E. & DiPrima, R. C (2001, p.357). Misalkan $\boldsymbol{v}$ adalah sebuah vektor tak nol, maka $\boldsymbol{A}$ (sebuah matriks mxm) yang memenuhi

$$\boldsymbol{Av} = \lambda v \tag{2.2}$$

dan λ adalah eigenvalue dari $\boldsymbol{A}$ yang berkorespondensi dengan $\boldsymbol{v}$.

Nilai dari λ dapat ditentukan dengan menggunakan persamaan berikut:

$$\det(\boldsymbol{A} - \lambda \boldsymbol{I}) = 0 \tag{2.3}$$

2.2.3. Matriks Jacobian

Menurut Boyce, W. E. & DiPrima, R. C (2001, p.102). Dalam kalkulus vektor, matriks jacobian adalah matriks yang berisi turunan parsial dari sebuah fungsi F yang dinyatakan dengan F: $\boldsymbol{R^m \rightarrow R^n}$ (fungsi yang menerima x ∈ $\mathbf{R}^n$ dan outputnya x ∈ $\mathbf{R}^m$). Jacobian matrix **J** dari F adalah sebuah matriks dengan ukuran mxn, yaitu:

$$\boldsymbol{J} = \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial F_m}{\partial x_1} & \cdots & \frac{\partial F_m}{\partial x_n} \end{bmatrix} \tag{2.4}$$

2.2.4. Determinan Matriks Persegi dengan kofaktor

Misalkan A adalah sebuah matriks mxm

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mm} \end{bmatrix} \tag{2.5}$$

Untuk menentukan determinan dari matriks harus menentukan suatu kolom untuk diekspansi agar bisa mendapatkan determinan. Misalkan kolom yang dipilih adalah kolom ke x.

Setelah itu kita perlu menghitung kofaktor untuk setiap anggota kolom tersebut.

$$\boldsymbol{Kofaktor : A_{ix} = (-1)^{i+x} \det(A_{ix})} \tag{2.6}$$

$$\boldsymbol{Determinan : Det\ (A) = \sum_{j=0}^{m} a_{jx} * A_{jx}} \tag{2.7}$$

2.2.5. Kestabilan Persamaan Diferensial

Menurut Boyce, W. E. & DiPrima, R. C (2001, p.357). Persamaan diferensial adalah persamaan matematika untuk fungsi satu variabel atau lebih yang menghubungkan nilai fungsi itu sendiri dan turunannya dalam berbagai orde.

Misalkan diketahui fungsi F, maka sebuah persamaan differensial dapat memiliki titik equilibrium (titik kesetimbangan) saat $\frac{dF}{dt} = 0$. Titik equilibrium tersebut dapat digolongkan menjadi dua yaitu yang stabil dan tidak stabil. Stabil atau tidaknya sebuah titik equilibrium dapat ditentukan dengan kriteria berikut:

1. Titik equilibrium x_e dikatakan stabil jika diambil x(t) yang berada di sekitar x_e, nilainya akan semakin mendekati x_e.
2. Titik equilibrium x_e dikatakan tidak stabil jika diambil x(t) yang berada di sekitar x_e, nilainya akan semakin menjauhi x_e.

Matriks jacobian dapat digunakan untuk menentukan kestabilan suatu titik equilibrium $x_e = (x_1, x_2, \dots, x_3)$ dengan cara mensubstitusi nilai x_e ke dalam J yang dilambangkan dengan $J|_{x_1, x_2, \dots, x_n}$ lalu menghitung nilai eigen(λ) dari dari hasil tersebut.

1. Jika bagian *real* dari setiap $\lambda < 0$ maka x_e stabil.
2. Jika ada bagian real dari λ ada yang > 0 maka x_e tidak stabil.
3. Jika bagian *real* terbesar dari $\lambda = 0$ maka x_e tidak dapat ditentukan stabil atau tidak.

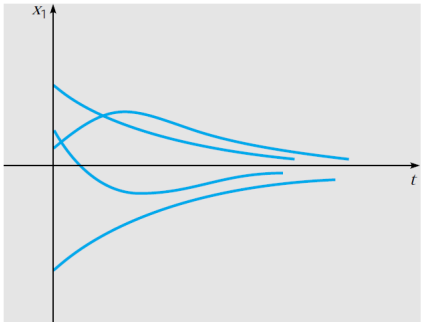
Kestabilan dari suatu sistem juga dapat digolongkan menjadi 3 jenis, yaitu:

1. Tidak stabil

Sebuah sistem persamaan tidak stabil adalah kondisi yang terjadi bila sampai kapanpun sistem tersebut berlangsung, nilai dari masing – masing variabelnya tidak membentuk pola atau menuju ke satu titik.

2. Stabil Asimtotik

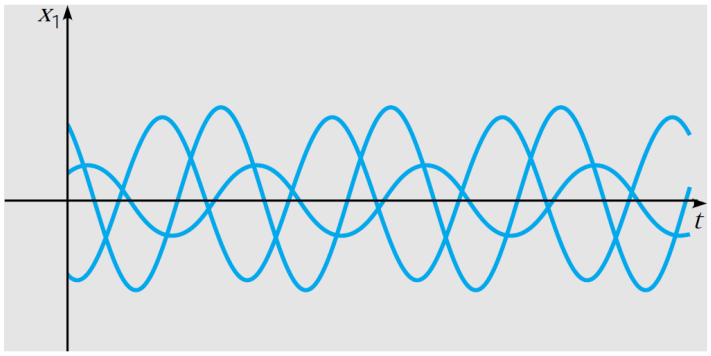
Sebuah sistem persamaan stabil asimtotik bila kondisi yang terjadi adalah masing – masing variabel pada sebuah sistem akan mencapai sebuah nilai pada waktu tertentu dan nilai tersebut tidak berubah. Nilai – nilai dari masing – masing variabel tersebut membentuk sebuah barisan yang konvergen.



Gambar 2.16 Contoh Ilustrasi Stabil Asimtotik

3. Stabil

Sebuah sistem persamaan stabil bila kondisi yang terjadi adalah masing – masing variabel pada sistem persamaan tersebut berubah membentuk pola tertentu dan selalu seperti untuk waktu seterusnya.



Gambar 2.17 Contoh Ilustrasi Stabil

2.2.6. Teori Bifurkasi

Menurut Yuri A. Kuznetsov(1998, p.1), bifurkasi adalah perubahan kualitatif pada sebuah sistem dinamik dikarenakan perubahan parameter dari sistem dinamik tersebut. Bifurkasi terbagi atas dua jenis, yaitu:

- Bifurkasi global: Terjadi ketika sebuah himpunan invarian yang relatif besar dari suatu sistem dinamik bertabrakan satu sama lain atau bertabrakan dengan titik equilibrium.
- Bifurkasi lokal: Bifurkasi yang terjadi ketika perubahan parameter menyebabkan kestabilan dari titik equilibrium berubah.

Dalam sistem dinamik bifurkasi lokal dapat dianalisis dengan menggunakan matriks jacobian dan *eigenvalue*. Misalkan sebuah sistem dinamik dideskripsikan dengan $\dot{x} = f(x, \lambda)$.

2.2.7. Metode Runge Kutta

Menurut Erwin Kreyszig(2006, p.892), metode runge-kutta orde 4 adalah salah satu metode numerik yang dapat digunakan untuk mengintegralkan suatu persamaan. Metode ini melakukan pehitungan dengan cara melakukan pengulangan berkali – kali hingga galat mencapai batas toleransi yang ditentukan. Bila kita misalkan syarat awal dari sebuah

sistem persamaan differensial adalah: $\frac{dy}{dt} = f(t, y)$ dan $y(t_0)=y_0$. Kita dapat menemukan solusi dari persamaan tersebut dengan cara menentukan terlebih dahulu langkah dari runge kutta yaitu h>0. Untuk menghitung nilai y dan t pada saat ke n+1 dapat diselesaikan dengan rumus di bawah ini:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$t_{n+1} = t_n + h$$

$$k_1 = f(t_n, y_n)$$

$$k_2 = f\left(t_n + \frac{h}{2}, y_n + \frac{1}{2}k_1 h\right)$$

$$k_3 = f\left(t_n + \frac{h}{2}, y_n + \frac{1}{2}k_2 h\right)$$

$$k_4 = f(t_n + h, y_n + k_3 h) \qquad (2.8)$$

2.2.8. Kriteria Routh-Hurwitz

Berdasarkan pemaparan oleh Gerald Teschl(2012, p.71), kriteria Routh-Hurwitz adalah sebuah kriteria penting yang memenuhi semua kondisi dari akar karakteristik dari polinomial. Misalkan diberikan polinomial sebagai berikut,

$$P(\lambda) = \lambda^n + a_1\lambda^{n-1} + \cdots + a_{n-a}\lambda + a_n \qquad (2.9)$$

Di mana koefisien a_i adalah bilangan *real* konstan dan $i = 1,2,\dots,n$. Definisikan n buah matriks Hurwitz menggunakan koefisien a_i dari karakteristik polinomial:

$H1 = (a_1)$, $H2 = \begin{pmatrix} a_1 & 1 \\ a_3 & a_2 \end{pmatrix}$, $H3 = \begin{pmatrix} a_1 & 1 & 0 \\ a_3 & a_2 & a_1 \\ a_5 & a_4 & a_3 \end{pmatrix}$, dan

$$H_n = \begin{pmatrix} a_1 & 1 & 0 & 0 & \dots & 0 \\ a_3 & a_2 & a_1 & 1 & \dots & 0 \\ a_5 & a_4 & a_3 & a_2 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \dots & \vdots \\ 0 & 0 & 0 & 0 & \dots & a_n \end{pmatrix}$$

Di mana $a_j = 0$ jika $j > n$. Seluruh akar polinomial $P(\lambda)$ bernilai negatif, atau memiliki nilai negatif pada bagian *real*-nya jika determinan dari seluruh matriks Hurwitz bernilai positif.