BAB II

LANDASAN TEORI

2.1 Chatbot

Chatbot merupakan program komputer yang didesain untuk berinteraksi dengan pengguna yang dapat menstimulasi percakapan dengan orang. Kemampuan berinteraksi tersebut didapatkan dengan menetapkan beberapa peraturan bahkan *Artificial Intelligence* (AI) (Burns, 2017).

Chatbot menyediakan layanan yang konsisten selama 24 jam / 7 hari seminggu. Selain itu, *chatbot* dapat ditingkatkan sebagai sumber daya, jasa, atau pengganti staf, dan menyediakan antarmuka yang *user-friendly*. Aplikasi ini menggantikan sistem navigasi yang rumit dan mengurangi pencarian panjang berdasarkan hasil pencarian (Allison, 2012).

2.2 AIML

AIML atau *Artificial Intelligence Mark-up Language* memungkinkan pengguna untuk memasukkan pengetahuan ke dalam *chatbot*. AIML dibuat oleh *Alicebot free software community* dan Dr. Richard S. Wallace (2001) selama tahun 1995 sampai tahun 2000. Pada awalnya, AIML diadaptasi dari tata bahasa non-XML yang kemudian menjadi dasar untuk Alicebot pertama yaitu A.L.I.C.E (*Artificial Linguistic Internet Entity Computer*). AIML menggambarkan *class* dari objek data yang disebut objek AIML dan secara parsial menggambarkan perilaku program komputer yang memprosesnya. Objek AIML terdiri dari unit yang disebut

topic dan *category* yang berisi data, baik terurai maupun tidak terurai. Data yang diurai terdiri dari karakter-karakter, beberapa di antaranya membentuk karakter data dan di antaranya membentuk elemen AIML. Elemen AIML merangkum pengetahuan *stimulus-response* yang terkandung dalam dokumen. Data karakter dalam elemen ini terkadang diurai oleh penerjemah AIML dan terkadang meninggalkan data yang tidak terurai untuk diproses kemudian oleh sebuah *responder* (Wallace, 2001).

Menurut penjelasan Dr. Richard S. Wallace (2001), unit dasar pengetahuan dalam AIML disebut sebagai *category*. Setiap *category* terdiri dari *input* pertanyaan, *output* jawaban, dan konteks opsional. Pertanyaan atau stimulus disebut *pattern*, sedangkan jawaban atau respon disebut *template*. Ada dua jenis konteks opsional yang disebut "*that*" dan "*topic*". Pola bahasa AIML cukup sederhana dan hanya terdiri dari kata-kata, spasi, dan simbol-simbol *wildcard* '_', '#', '^' dan '*'. Kata-kata dapat terdiri dari huruf dan angka, tetapi tidak boleh ada karakter lain. Kata-kata tersebut dipisahkan oleh satu spasi dan karakter *wildcard* berfungsi sama dengan kata-kata.

Pada versi pertama AIML, hanya satu karakter *wildcard* yang diperbolehkan dalam setiap *pattern*. Standar AIML 1.01 memungkinkan adanya beberapa *wildcard* pada setiap pola, tetapi bahasanya dirancang sesederhana mungkin bahkan lebih sederhana dibandingkan dengan *regular expression*. *Template* merupakan respon dari AIML dalam bentuk yang paling sederhana dan hanya terdiri dari teks biasa (Wallace, 2001).

Tag AIML mengubah jawabannya menjadi program komputer mini yang dapat menyimpan data, mengaktifkan program lain, memberikan tanggapan

bersyarat, dan secara rekursif memanggil *pattern matcher* untuk memasukkan tanggapan dari *category* lainnya (Wallace, 2001).

Wallace (2001) menjelaskan bahwa AIML memiliki bagian konteks opsional yang terdiri dari dua kategori, yaitu *tag* <that> dan <topic>. *Tag* <that> muncul di dalam *category* dan *pattern*-nya harus sesuai dengan kata-kata terakhir *bot*. Yang penting jika mengajukan pertanyaan sebagai berikut.

```
<category>
        <pattern>KNOCK KNOCK</pattern>
        <template>Who is there?</template>
</category>
<category>
        <pattern>*</pattern>
        <that>WHO IS THERE</that>
        <template><person/> who?</template>
</category>
<category>
        <pattern>*</pattern>
        <that>* WHO</that>
        <template>Ha ha very funny, <get name="name"/>.</template>
</category>
```

Hasil yang akan muncul adalah sebagai berikut.

```
User    : Knock knock.
Bot     : Who's there?
User    : Banana.
Bot     : Banana who?
User    : Knock knock.
Bot     : Who's there?
User    : Banana.
Bot     : Banana who?
User    : Knock knock.
Bot     : Who's there?
User    : Orange.
Bot     : Orange who?
User    : Orange you glad I didn't say banana.
Bot     : Ha ha very funny, Nancy.
```

Interpreter AIML menyimpan *pattern input*, *pattern* "*that*", dan *pattern* "*topic*" dalam satu jalur, yaitu "INPUT <that> THAT <topic> TOPIC". Ketika nilai dari <that> dan <topic> tidak ditentukan, program akan secara implisit menetapkan nilai-nilai yang sesuai dengan *pattern* "*that*" dan "*topic*" sebagai *wildcard* '*' (Wallace, 2001).

Bagian pertama yang akan dicocokkan adalah *input*. Jika lebih dari satu kategori memiliki *pattern input* yang sama, program akan dapat membedakan dengan melihat nilai *tag* <that>. Jika dua atau lebih kategori memiliki *tag* <pattern> dan *tag* <that> yang sama, langkah terakhir dari program adalah untuk memilih jawaban berdasarkan *tag* <topic>. Struktur tersebut menunjukkan aturan desain, sebagai berikut (Wallace, 2001).

a. Hanya gunakan *tag* <that> jika terdapat dua *category* dengan *tag* <pattern> yang sama.
b. Hanya gunakan *tag* <topic> jika terdapat dua *category* dengan *tag* <pattern> dan *tag* <that> yang sama.

Tag <topic> muncul di luar *category* dan mengumpulkan suatu grup dari *category* bersama-sama, akan tetapi dapat juga diatur dalam *template*. Salah satu keuntungan dari penggunaan *tag* <topic> adalah untuk menciptakan *subject-dependent "pickup-lines"* yang akan memisahkan *category* sesuai dengan *topic*, sebagai berikut (Wallace, 2001).

```
<topic name="CARS">
<category>
    <pattern>*</pattern>
    <template>
    <random>
    <li>What's your favorite car?</li>
    <li>What kind of car do you drive?</li>
    <li>Do you get a lot of parking tickets?</li>
```

```
        <li>My favorite car is one with a driver.</li>
        </random>
        </template>
</category>
```

Dr. Richard S. Wallace (2001) juga mengatakan bahwa AIML tidak sama persis dengan *database* sederhana yang berisi pertanyaan dan jawaban. Pola pencocokan "*query*" pada bahasa jauh lebih sederhana dibandingkan dengan SQL. Tetapi *template category* dapat berisi *tag* <srai>, sehingga *output* tidak hanya bergantung pada satu *category* yang cocok, tetapi pada *category* lainnya yang secara berulang dicapai melalui <srai>.

Wallace (2001) menjelaskan bahwa AIML mengimplementasikan pengulangan dengan operator <srai>. Tidak ada kesepakatan mengenai makna dari akronim. Perselisihan tentang akronim menggambarkan bermacam-macam aplikasi untuk <srai> dalam AIML sebagai berikut.

a. *Symbolic reduction*

Symbolic reduction mengacu pada proses penyederhanaan bentuk gramatikal yang kompleks menjadi lebih sederhana. Biasanya, *atomic pattern* dalam *category* menyimpan pengetahuan *bot* yang dinyatakan dalam istilah sesederhana mungkin, misalnya *pattern* "WHO IS SOCRATES" lebih sederhana dibanding dengan *pattern* "DO YOU KNOW WHO SOCRATES IS" ketika menyimpan informasi biografi tentang Socrates.

Banyak bentuk yang lebih kompleks direduksi menjadi bentuk yang lebih sederhana menggunakan *category* AIML yang dirancang untuk *symbolic reduction* sebagai berikut.

```
<category>
    <pattern>DO YOU KNOW WHO * IS</pattern>
    <template><srai>WHO IS <star/></srai></template>
</category>
```

Apapun *input* yang cocok dengan *pattern* ini, porsi yang terikat pada *wildcard* '*' dapat dimasukkan ke dalam balasan dengan *markup* <star/>. *Category* ini mengurangi *input* dari bentuk "Do you know who X is?" menjadi "Who is X?".

b. *Divide and conquer*

Banyak kalimat individu yang dapat dikurangi menjadi dua atau lebih subkalimat dan jawabannya dibentuk dengan menggabungkan balasan masing-masing. Misalnya jika sebuah kalimat yang diawali dengan kata "Ya" memiliki lebih dari satu kata, dapat diperlakukan sebagai subkalimat "Ya" ditambah dengan apapun yang mengikutinya.

```
<category>
    <pattern>YES *</pattern>
    <template><srai>YES</srai> <sr/></template>
</category>
```

Markup <sr/> merupakan kependekan dari <srai><star/></srai>.

c. *Synonyms*

Standar AIML versi 1.01 tidak mengizinkan ada lebih dari satu *pattern* dalam setiap *category*. *Synonym* mungkin merupakan aplikasi yang paling umum dari <srai>. Banyak cara untuk mengurangi hal yang sama menjadi satu *category*, salah satunya dengan menggunakan *tag* <srai> untuk memanggil *pattern* yang sudah ada seperti contoh berikut.

```
<category>
    <pattern>HELLO</pattern>
    <template>Hi there!</template>
</category>
```

```
<category>
        <pattern>HI</pattern>
        <template><srai>HELLO</srai></template>
</category>
<category>
        <pattern>HI THERE</pattern>
        <template><srai>HELLO</srai></template>
</category>
<category>
        <pattern>HOWDY</pattern>
        <template><srai>HELLO</srai></template>
</category>
<category>
        <pattern>HOLA</pattern>
        <template><srai>HELLO</srai></template>
</category>
```

d. *Spelling and grammar correction*

Hal yang menjadi kesalahan ejaan umum dari pengguna adalah penggunaan "your" padahal yang dimaksud adalah "you're" atau "you are". Akan tetapi, tidak semua kata "your" harus diubah menjadi "you're". Sejumlah kecil konteks tata bahasa biasanya diperlukan untuk mengatasi kesalahan ini. Berikut merupakan *bot* yang mengoreksi *input* dari pengguna dan bertindak sebagai tutor bahasa.

```
<category>
        <pattern>YOUR A *</pattern>
        <template>I think you mean "you're" or "you are" not "your."
                <srai>YOU ARE A <star/></srai>
        </template>
</category>
```

e. *Keywords*

Penulisan *template* AIML yang akan aktif disesuaikan dengan munculnya *keyword* dalam kalimat *input* seperti ilustrasi berikut.

```
<category>
        <pattern>MOTHER</pattern>
        <template> Tell me more about your family. </template>
</category>
```

```
<category>
        <pattern>_ MOTHER</pattern>
        <template><srai>MOTHER</srai></template>
</category>
<category>
        <pattern>MOTHER _</pattern>
        <template><srai>MOTHER</srai></template>
</category>
<category>
        <pattern>_ MOTHER *</pattern>
        <template><srai>MOTHER</srai></template>
</category>
```

Kategori pertama mendeteksi *keyword* saat muncul sendirian dan memberikan respon umum. Kategori kedua mendeteksi *keyword* saat muncul sebagai akhiran kalimat. Kategori ketiga mendeteksi *keyword* sebagai awalan kalimat. Kategori terakhir mendeteksi *keyword* sebagai *infix*. Masing-masing dari ketiga kategori terakhir menggunakan <srai> untuk menghubungkan ke jawaban kategori pertama, sehingga semua kategori menghasilkan jawaban yang sama, tetapi perlu ditulis dan disimpan hanya sekali.

f. *Conditionals*

Cabang kondisional dalam AIML dapat ditulis hanya menggunakan *tag* <srai> dengan mempertimbangkan tiga kategori sebagai berikut.

```
<category>
        <pattern>WHO IS HE</pattern>
        <template><srai>WHOISHE <get name="he"/></srai></template>
</category>
<category>
        <pattern>WHOISHE *</pattern>
        <template>He is <get name="he"/>.</template>
</category>
<category>
        <pattern>WHOISHE UNKNOWN</pattern>
        <template>I don’t know who he is.</template>
</category>
```

Apabila predikat “he” diinisialisasi menjadi “unknown”, *category* akan menjalankan cabang bersyarat, tergantung pada apakah “he” telah ditetapkan. AIML juga menyediakan fungsi setara melalui *tag* <condition>.

Menuru Wallace (2001), bahaya penggunaan *tag* <srai> adalah memungkinkan *botmaster* untuk membuat *infinite loops*. Meskipun ada beberapa resiko untuk *programmer* pemula, penggunaan *tag* <srai> lebih sederhana dibandingkan dengan *tag-tag* yang berulang.

2.3 Confix-stripping

Algoritma *Confix-stripping stemmer* adalah algoritma yang digunakan untuk melakukan proses *stemming* terhadap kata-kata berimbuhan (Adriani dkk, 2007). Algoritma ini mempunyai aturan imbuhan sendiri dengan model sebagai berikut.

[[[AW +]AW +]AW +] Kata-Dasar [[+AK][+KK][+P] ...(2.1)

AW : Awalan AK : Akhiran
KK : Kata ganti kepunyaan P : Partikel

Stemming berarti proses pemetaan dan penguraian berbagai bentuk (*variants*) dari suatu kata menjadi bentuk kata dasarnya (*stem*) (Tala, 2003). Tujuan dari proses *stemming* adalah menghilangkan imbuhan-imbuhan baik itu berupa *prefix*, *suffix*, maupun *confix* yang ada pada setiap kata. Jika imbuhan tersebut tidak dihilangkan, setiap satu kata akan disimpan dengan berbagai macam bentuk yang berbeda sesuai dengan imbuhan yang melekatinya sehingga hal tersebut akan menambah beban *database*. Karena Bahasa Indonesia mempunyai aturan morfologi, maka proses *stemming* harus berdasarkan aturan morfologi Bahasa Indonesia.

Langkah-langkah algoritma *Confix-stripping* menurut Adriani dkk (2007) adalah sebagai berikut.

1) Kata yang belum di-*stemming* dibandingkan ke dalam *database* kamus kata dasar. Jika sesuai, kata tersebut diasumsikan sebagai kata dasar dan algoritma berhenti. Jika kata tidak sesuai dengan kata dalam kamus, lanjutkan ke langkah 2.
2) Jika kata di-*input* memiliki pasangan awalan-akhiran "be-lah","be-an","me-i","di-i","pe-i", atau "te-i", langkah *stemming* selanjutnya adalah langkah 5, 3, 4, 5, dan 6, tetapi jika kata yang di-*input* tidak memiliki pasangan awalan-akhiran tersebut, langkah *stemming* berjalan normal yaitu 3, 4, 5, 6.
3) Hilangkan partikel dan kata ganti kepunyaan. Pertama hilangkan partikel ("-lah", "-kah", "-tah", "-pun"). Setelah itu hilangkan juga kata ganti kepunyaan ("-ku", "-mu", atau "-nya"). Contoh: kata "bajumulah", proses *stemming* pertama menjadi "bajumu" dan proses *stemming* kedua menjadi "baju". Jika kata "baju" ada di dalam kamus, algoritma akan berhenti sesuai dengan model imbuhan sebagai berikut.

[[[AW+]AW+]AW+] Kata Dasar [+AK] ...(2.2)

4) Hilangkan juga akhiran ("-i","-an", dan "-kan"), sesuai dengan model imbuhan sebagai berikut.

[[[AW+]AW+]AW+] Kata Dasar ...(2.3)

Contoh : kata "membelikan" di-*stemming* menjadi "membeli", jika tidak ada dalam *database* kata dasar, akan dilakukan proses penghilangan awalan.

5) Penghilangan awalan ("be-", "di-", "ke-", "me-", "pe-", "se-", dan "te-") mengikuti langkah-langkah berikut.

a) Algoritma akan berhenti sesuai aturan sebagai berikut.

(1) Awalan diidentifikasi bentuk sepasang imbuhan yang tidak diperbolehkan dengan akhiran (berdasarkan Tabel 2.1) yang dihapus pada langkah 3.

(2) Diidentifikasi awalan yang sekarang identik dengan awalan yang telah dihapus sebelumnya.

(3) Kata tersebut sudah tidak memiliki awalan.

b) Identifikasi jenis awalan dan peluruhannya bila diperlukan. Jenis awalan ditentukan dengan aturan sebagai berikut.

(1) Jika awalan dari kata adalah “di-“, “ke-“, atau “se“, awalan dapat langsung dihilangkan.

(2) Hapus awalan “te-“, “be-“, “me-“, atau “pe-“ yang menggunakan aturan peluruhan yang dijelaskan pada Tabel 2.2. Sebagai contoh kata “menangkap”, setelah menghilangkan awalan “me-“ maka kata yang didapat adalah “nangkap”. Karena kata “nangkap” tidak ditemukan dalam *database* kata dasar, maka karakter “n” diganti dengan karakter “t” sehingga dihasilkan kata “tangkap” dan kata “tangkap” merupakan kata yang sesuai dengan kata yang ada di *database* kata dasar, maka algoritma berhenti.

6) Jika semua langkah gagal, kata yang diuji pada algoritma ini dianggap sebagai kata dasar (Kurniawan dkk, 2012).

Tabel 2.1 Kombinasi *Prefix* dan *Suffix* yang Tidak Diperbolehkan (Adriani dkk, 2007)

Awalan (*Prefix*)	**Akhiran (*Suffix*)**
be-	-i
di-	-an
ke-	-i -an
me-	-an
se-	-i -kan
te-	-an

Tabel 2.2 Aturan Peluruhan Kata Dasar (Adriani dkk, 2007)

Aturan	**Bentuk Awalan**	**Peluruhan**
1	berV...	ber-V… | be-rV…
2	Belajar...	bel-ajar
3	beC$_1$erC$_2$…	be-C$_1$erC$_2$…dimana C$_1$!={'r'|'l'}
4	terV…	ter-V…|te-rV…
5	terCer…	ter-Cer…dimana C!='r'
6	teC$_1$erC$_2$	te-C$_1$erC$_2$…dimana C$_1$!='r'
7	me{l|r|w|y}V…	me-{l|r|w|y}V…
8	mem{b|f|v}…	mem-{b|f|v}…
9	mempe…	mem-pe…
10	mem{rV|V}…	me-m{rV|V}…|me-p{rV|V}…
11	men{c|d|j|z}…	men-{c|d|j|z}…
12	menV…	me-nV…|me-tV…
13	meng{g|h|q|k }…	meng-{g|h|q|k}…
14	mengV…	meng-V…|meng-kV…
15	mengeC	menge-C
16	menyV…	me-ny…| meny-sV…
17	mempV…	mem-pV…
18	pe{w|y}V…	pe-{w|y}V…
19	perV…	per-V…|pe-rV…
20	pem{b|f|v}…	pem-{b|f|v}…
21	pem{rV|V}…	pe-m{rV|V}…|pe-p{rV|V}
22	pen{c|d|j|z}…	pen-{c|d|j|z}…
23	penV…	pe-nV…|pe-tV…
24	peng{g|h|q}	peng-{g|h|q}
25	pengV	peng-V|peng-kV
26	penyV…	pe-nya|peny-sV
27	pelV..	pe-lV…; kecuali untuk kata "pelajar" menjadi "ajar"
28	peCP	pe-CP…dimana C!={r|w|y|l|m|n} dan P!='er'
29	perCerV	Per-CerV… dimana C!={r|w|y|l|m|n}

2.4 Objek Wisata Belitung

Menurut Dinas Pariwisata Belitung (2016), objek wisata di Belitung terbagi ke dalam tiga kategori, yaitu wisata alam, wisata alam buatan, dan wisata budaya. Objek wisata alam berupa laut, pantai, gunung, danau, dan flora, sedangkan objek wisata alam buatan berupa kawasan lindung, cagar alam, dan pemandangan alam. Objek wisata budaya berupa upacara kelahiran, tari-tarian tradisional, pakaian adat, bangunan bersejarah, peninggalan tradisional, festival budaya, pertunjukkan tradisional, adat istiadat lokal, dan museum.

Ada 56 data objek wisata di Belitung yang diperoleh dari Dinas Pariwisata Belitung pada tahun 2016. Berikut daftar objek wisata beserta kategorinya.

Tabel 2.3 Daftar Objek Wisata Belitung

No	Objek Wisata	Kategori
1	Pantai Tanjung Pendam	Wisata Alam
2	Pulau Kalemua	
3	Pantai Pulau Bayan	
4	Pantai Pegantungan	
5	Pantai Tanjung Kelayang	
6	Pantai Bebilai	
7	Pulau Batu Berlayar	
8	Pulau Babi/Kepayang	
9	Pulau Burung	
10	Pulau Lengkuas	
11	Pantai Bukit Berahu	
12	Pantai Secupak	
13	Pantai Mabai	
14	Pantai Tanjung Tinggi	
15	Pantai Pendaunan	
16	Pantai Penyaeran	
17	Pantai Siantu	
18	Pantai Teluk Gembira	
19	Pantai Penyabong	
20	Bukit Batu Baginde	
21	Pantai Tanjung Kiras	
22	Pantai Pasir Panjang	
23	Gurok Beraye	
24	Batu Mentas	
25	Pantai Marina	

Tabel 2.3 Daftar Objek Wisata Belitung (Lanjutan)

No	Objek Wisata	Kategori
26	Pulau Batu Dinding	Wisata Alam
27	Pantai Batu Lubang	
28	Pantai Awan Mendung	
29	Danau Biru Kulong Murai	
30	Pesona Indomarine	
31	Gue Nek Santen	
32	Air Terjun Gunong Kubing	
33	Kawasan Pulau Lima Selat Nasik	
34	Bukit Peramun	
35	Pulau Leebong	
36	Pantai Batu Bedil	
37	Jurak Insum Kepala Kawai	
38	Pulau Seliu	
39	Taman Satwa	Wisata Alam Buatan
40	Pemandian Dayang Sri Pinai	
41	Taman Hiburan Kulong Keramik	
42	Kulong Kaolin Murai	
43	Tirta Merundang Indah	
44	Pemandian Suci Indah	
45	Kampung Orange	
46	Museum Tanjungpandan	Wisata Budaya
47	Museum Badau	
48	Makam Raja Badau	
49	Makam Kota Tanah Cerucuk	
50	Makam Syech Abubakar	
51	Makam K.A. Rahad	
52	Mesjid Tua Sijuk	
53	Klenteng Sijuk	
54	Relief Perjuangan Rakyat Belitung	
55	Rumah Adat Belitung	
56	Toapekong Marga Ho	

2.5 Sampling Insidental

Menurut Sugiyono (2010), *sampling* insidental adalah teknik penentuan sampel berdasarkan kebetulan, yaitu siapa saja yang secara kebetulan bertemu dengan peneliti dapat digunakan sebagai sampel, bila dipandang orang yang kebetulan ditemui itu cocok sebagai sumber data. Jumlah sampel yang dibutuhkan

mengikuti perkataan Roscoe yang dikutip Uma Sekaran (2006), bahwa ukuran sampel lebih dari 30 dan kurang dari 500 adalah tepat untuk kebanyakan penelitian.

2.6 Skala Likert

Skala Likert menurut Sugiyono (2010) merupakan skala yang digunakan untuk mengukur sikap, pendapat, dan persepsi dari individu atau kelompok tentang fenomena sosial. Fenomena sosial ini disebut variabel penelitian yang telah ditetapkan secara spesifik oleh peneliti. Menurut Darmadi (2011), rumus yang digunakan untuk menghitung total nilai Skala Likert adalah sebagai berikut.

$$t = T \times Pn \quad ...(2.5)$$

T: Total responden Pn: Nilai Skala Likert
t: Total Nilai Skala Likert

Untuk mendapatkan hasil interpretasi nilai Skala Likert, nilai tertinggi dan nilai terendah untuk *item* penilaian harus terlebih dahulu diketahui. Nilai tertinggi dan nilai terendah dapat dihitung menggunakan Rumus 2.5, yaitu dengan mengalikan total responden dengan nilai tertinggi ataupun terendah Skala Likert. Pada Skala Likert Lima Tingkat, nilai Skala Likert tertinggi adalah 5, sedangkan nilai Skala Likert terendah adalah 1 dengan interval persentase sesuai dengan Tabel 2.4.

Tabel 2.4 Interval Persentase Skala Likert

No	Keterangan	Interval
1	Sangat Tidak Setuju	0% – 19,99%
2	Tidak Setuju	20% – 39,99%
3	Netral	40% – 59,99%
4	Setuju	60% – 79,99%
5	Sangat Setuju	80% – 100%

Untuk menghitung persentase nilai Skala Likert, rumus interpretasinya adalah sebagai berikut.

$$P = \frac{t}{Y} \times 100 \quad \text{...(2.6)}$$

P: Persentase nilai
t: Total Nilai Skala Likert
Y: Total Nilai Tertinggi Skala Likert

2.7 Technology Acceptance Model (TAM)

Technology Acceptance Model yang telah dikembangkan oleh Davis (1989) adalah salah satu model penelitian yang paling populer dan telah dipelajari secara luas dan diverifikasi oleh berbagai studi yang menguji perilaku penerimaan teknologi individual dalam konstruksi sistem informasi yang berbeda. Pada model TAM, ada dua faktor dalam perilaku penggunaan komputer, yaitu *perceived usefulness* dan *perceive ease of use*. Davis mendefinisikan *perceived usefulness* sebagai calon pengguna. *Perceive ease of use* (EOU) dapat didefinisikan sebagai tingkat dimana calon pengguna mengharapkan sistem target bebas dari usaha. Menurut TAM, kemudahan penggunaan dan kegunaan yang dirasakan merupakan faktor penentu penggunaan sistem yang paling penting.

Masing-masing faktor dinilai menggunakan enam poin pernyataan yang telah ditentukan oleh Davis melalui beberapa proses. Pada awalnya terdapat 14 poin untuk penilaian masing-masing faktor, lalu Davis melakukan wawancara dengan 15 orang pengguna komputer berpengalaman untuk menentukan apakah poin-poin tersebut sesuai untuk mengukur *perceive ease of use* dan *perceive usefulness*. Dari wawancara tersebut didapatkan 10 poin pada masing-masing faktor. Setelah itu, Davis melakukan tes *reliability* dan *validity* terhadap 10 poin tersebut dengan

melakukan studi lapangan kepada 112 karyawan yang bekerja di IBM Toronto, Kanada. Studi lapangan dilakukan dengan memberikan skala 1 sampai 7 Skala Likert untuk setiap poin pernyataan. Dari hasil dari studi lapangan, didapatkan 6 poin untuk setiap faktor sesuai dengan Tabel 2.5 dan Tabel 2.6 sebagai berikut.

Tabel 2.5 Butir Pengukuran Perceived Usefulness

No	Butir Pengukuran Perceived Usefulness
1	Menggunakan 'Aplikasi' dalam pekerjaan saya memungkinkan saya mengerjakan pekerjaan dengan lebih cepat.
2	Menggunakan 'Aplikasi' akan meningkatkan kinerja pekerjaan saya.
3	Menggunakan 'Aplikasi' dalam pekerjaan akan meningkatkan produktivitas saya.
4	Menggunakan 'Aplikasi' akan meningkatkan efektivitas dalam pekerjaan saya.
5	Menggunakan 'Aplikasi' akan mempermudah dalam melakukan pekerjaan saya.
6	'Aplikasi' berguna dalam pekerjaan saya.

Tabel 2.6 Butir Pengukuran Perceived Ease of Use

No	Butir Pengukuran Perceived Ease of Use
1	Belajar menggunakan 'Aplikasi' mudah bagi saya.
2	Saya merasa mudah untuk mengoperasikan 'Aplikasi'.
3	Interaksi dengan 'Aplikasi' jelas dan mudah dimengerti.
4	Saya merasa fleksibel untuk berinteraksi dengan 'Aplikasi'.
5	Saya merasa mudah untuk menjadi ahli dalam menggunakan 'Aplikasi'.
6	'Aplikasi' mudah digunakan.