BAB II
LANDASAN TEORI

2.1 Tinjauan Pustaka

Parno, dkk (2011), menjelaskan bahwa dari penelitian yang berjudul aplikasi *mobile* kamus istilah psikologi berbasis *android* dapat disimpulkan bahwa merancang sebuah aplikasi *mobile* kamus istilah psikologi berbasis *android 2.2* yang merupakan aplikasi kamus pada *smartphone* dibuat dengan menggunakan salah satu paket *java* dan XML (*Extensible Markup Language)*. Kamus istilah psikologi tersebut mempunyai fungsi untuk mencari serta menginput kata istilah psikologi. Kamus tersebut bermanfaat bagi pengguna agar mudah mencari dan mengupdate kata istilah psikologi sesuai kebutuhan.

Argakusumah dan Hansun (2014), melalui penelitian yang berjudul implementasi algoritma *boyer-moore* pada aplikasi kamus kedokteran berbasis *android* berdasarkan hasil implementasi dan uji coba yang dilakukan sebelumnya, simpulan dari penelitian ini adalah implementasi algoritma *boyer-moore* dalam pencarian kata pada aplikasi kamus kedokteran berhasil dilakukan dirancang dengan menggunakan *software java* dan *SQLite*. Simpulan ini dirumuskan berdasarkan hasil uji coba, dimana pencarian kata dengan algoritma *boyer-moore* berhasil dilakukan dengan persentase sebesar 100% dari responden menyatakan bahwa proses pencarian istilah pada aplikasi kamus kedokteran dapat memberikan penjelasan (hasil) yang sesuai dengan yang diharapkan.

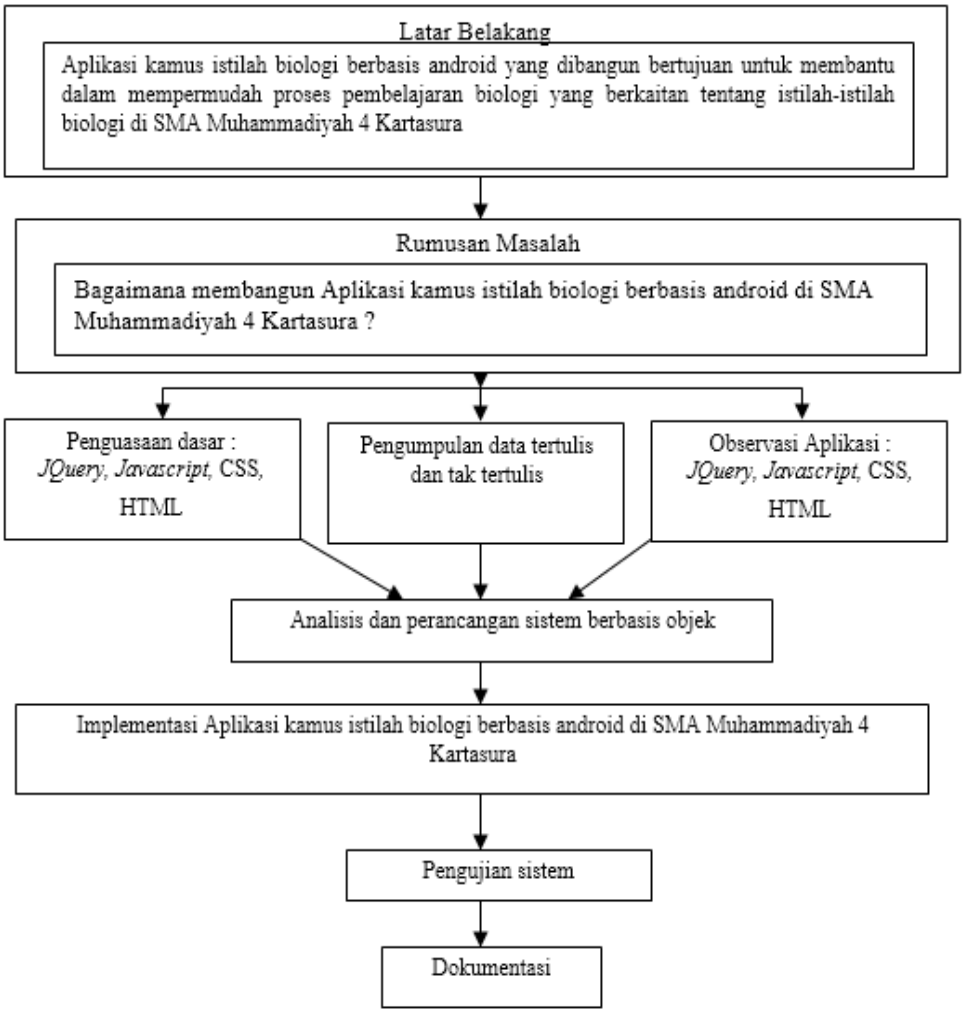
Marpaung (2013), menjelaskan bahwa dari penelitian yang berjudul implementasi algoritma *string matching* pada kamus istilah-istilah kedokteran berbasis *android* dapat disimpulkan bahwa penggunaan kamus sangat diperlukan namun tidak mempersulit pengguna saat menggunakannya dan dapat mempermudah pemakai dalam menterjemahkan suatu bahasa tanpa harus membawa kamus yang berbentuk buku yang memiliki ketebalan dan bobot yang cukup berat. Untuk itu dibutuhkan sebuah aplikasi dimana aplikasi tersebut berupa kamus *mobile*, yang dapat dipasang pada perangkat *mobile* berbasis *android*. Dan untuk mempermudah pencarian kata dalam kamus *mobile* tersebut,

aplikasi ini dirancang dengan menggunakan algoritma *string matching* yang digunakan adalah algoritma *brute force* dapat diterapkan dalam perancangan aplikasi kamus istilah kedokteran sehingga dapat memudahkan pengguna untuk mencari kata yang ingin diterjemahkan. Aplikasi kamus istilah kedokteran telah selesai dirancang dengan menggunakan *software eclipse galileo* sebagai editor, *Software Development Kit* (SDK) sebagai *platform* dan telah dapat dijalankan pada perangkat *mobile* dengan sistem operasi Android 2.2 froyo (*frozen yoghurt*).

Dari ketiga penelitian dapat disimpulkan bawah ada perbedaan masing-masing aplikasi yaitu yang pertama dari penelitian aplikasi *mobile* kamus istilah psikologi berbasis *android* dapat mencari terjemahan kata tanpa harus membawa buku, mempermudah dalam proses pencarian kata, user dapat merubah dengan menambah dan menghapus kata sesuai dengan yang dibutuhkan. Kedua penelitian implementasi algoritma *boyer-moore* pada aplikasi kamus kedokteran berbasis *android* dari segi pencarian kata dengan menggunakan algoritma *boyer-moore* yaitu salah satu algoritma untuk mencari suatu *string* di dalam teks. Algoritma *boyer-moore* melakukan perbandingan dimulai dari kanan ke kiri, tetapi pergeseran *window* tetap dari kiri ke kanan. Jika terjadi kecocokkan maka dilakukan perbandingan karakter teks dan karakter pola yang sebelumnya, yaitu dengan sama-sama mengurangi indeks teks dan pola masing-masing sebanyak satu. Dengan mengunakan algoritma ini, secara rata-rata proses pencarian akan menjadi lebih cepat jika dibandingkan dengan algoritma lainnya. Alasan melakukan pencocokkan dari kanan (posisi terakhir *string* yang dicari). Ketiga penelitian implementasi algoritma *string matching* pada kamus istilah-istilah kedokteran berbasis *android* dari segi pencarian juga dengan menggunakan algoritma *brute force* untuk proses pencarian istilah dimana algoritma *brute force* berguna bagi para pemakai yang menginginkan cara cepat untuk mencari kata dalam kamus istilah yang ada dalam *handphone* dengan sistem operasi *android*. algoritma *brute force* yaitu algoritma yang digunakan untuk mencocokan *pattern* dengan semua teks antara 0 dan n-m untuk menemukan keberadaan *pattern* teks. Algoritma *brute force* memecahkan masalah dengan sangat sederhana, langsung, dan jelas.

2.2 Kerangka Pemikiran

Kerangka pemikiran disajikan pada Gambar 2.1



Gambar 2.1 Diagram Kerangka Pemikiran

2.3 Landasan Teori

2.3.1 Aplikasi

Aplikasi berasal dari kata *application* yang artinya penerapan, lamaran, penggunaan. Secara istilah aplikasi adalah program siap pakai yang direka untuk melaksanakan suatu fungsi bagi pengguna atau aplikasi yang lain dan dapat digunakan oleh sasaran yang dituju (Ahmad, 2013:10 dikutip oleh Susilo, 2015). Adapun karakteristik aplikasi adalah sebagai berikut:

1. Aplikasi merupakan elemen sistem logik dan bukan elemen sistem fisik seperti perangkat keras.

2. Elemen aplikasi bisa rusak.
3. Elemen aplikasi bisa direkayasa atau dikembangkan dan bukan dibuat di pabrik, seperti halnya perangkat keras.
4. Aplikasi tidak bisa dirakit atau disusun.

2.3.2 Kamus

Kamus adalah buku acuan yang memuat kata dan ungkapan, biasanya disusun menurut abjad beserta penjelasan tentang makna dan pemakaiannya (Kamus Besar Bahasa Indonesia). Kamus disusun sesuai dengan abjad dari A-Z dengan tujuan untuk memudahkan pengguna kamus dalam mencari istilah yang diinginkannya dengan mudah dan akurat.

2.3.3 *Android*

Android merupakan sebuah sistem operasi pada *smartphone* yang bersifat terbuka dan berbasis pada sistem operasi *linux*. *Android* bisa digunakan oleh setiap orang yang ingin menggunakannya pada perangkat mereka. *Android* menyediakan *platform* terbuka bagi para pengembang untuk menciptakan aplikasi mereka sendiri yang akan digunakan untuk bermacam peranti bergerak. Awalnya, *Google Inc* membeli *Android Inc* pendatang baru yang membuat peranti lunak untuk ponsel. Kemudian untuk mengembangkan *android,* dibentuklah *open handset alliance,* konsorsium dari 34 perusahaan peranti keras, peranti lunak, dan telekomunikasi, termasuk *google, HTC, intel, motorola, qualcomm, T-Mobile, dan nvidia*. pada saat perilisan perdana *android*, 5 november 2007, *android* bersama *open handset alliance* menyatakan mendukung pengembangan standar terbuka pada perangkat seluler. Di lain pihak Google merilis kode–kode *android* di bawah lisensi *apache,* sebuah lisensi perangkat lunak dan standar terbuka perangkat seluler. Adapaun ikon dari sistem operasi *android* ini sering disebut juga dengan sebutan *robot ijo* (Fadlullah, 2012:25 dikutip oleh Susilo, 2015).

2.3.4 *JQuery*

Jquery adalah *javascript library,* kumpulan kode atau fungsi *javascript* siap pakai, sehingga memudahkan dan mempercepat kita dalam membuat kode *javascript*. *JQuery mobile* merupakan cara aman agar tetap bisa bertahan di dunia *mobile programming,* karena dukungan *multi-platform* mulai dari *android, iPhone*

Operating System (iOS) sampai *blackberry*, tidak hanya untuk perangkat *mobile* tapi bisa untuk membuat responsive *website* serta digunakan oleh 57% *website* di seluruh dunia dengan kecepatan pertumbuhan yang stabil dan selalu meningkat tanpa menunjukkan pengurangan (Gliser,2013 dikutip oleh Ajie, 2014).

2.3.5 *Javascript*

Javascript diperkenalkan pertama kali oleh *Netscape* pada tahun 1995. Pada awalnya bahasa yang sekarng disebut *javascript* ini dulunya dinamai *livescript* yang berfungsi sebagai bahasa sederhana untuk *browser netscape navigator* 2 yang sangat popular pada saat itu. Kemudian sejalan dengan sedang giatnya kerjasama antara *Netscape* dan *SUN* (pengembang bahasa pemrograman *java*) pada masa itu, maka *netscape* memberikan nama "*javascript*" kepada bahasa tersebut pada tanggal 4 Desember 1995. Pada saat yang bersamaan *Microsoft* sendiri mencoba untuk mengadaptasikan teknologi ini yang mereka sebut sebagai "*JScript*" di *browser* milik mereka yaitu internet explorer 3. *Javascript* sendiri merupakan modifikasi dari bahasa pemrograman C++ dengan pola penulisan yang lebih sederhana dari bahasa pemrograman C++.

Javascript adalah bahasa pemrograman yang sederhana karena bahasa ini tidak dapat digunakan untuk membuat aplikasi ataupun *applet*. Dengan *javascript* kita dapat dengan mudah membuat sebuah halaman *web* yang interaktif. Program *javascript* dituliskan pada file HTML *(*.htm*.html*). (Hardjono, 2006:4 dikutip oleh Nandari dan Sukadi, 2014).

2.3.6 *Cascanding Style Sheet* (CSS)

Cascanding Style Sheet (CSS) adalah sebuah dokumen yang berisi aturan yang digunakan untuk memisahkan isi dengan *layout* dalam halaman-halaman *web* yang dibuat. CSS memperkenalkan "*template*" yang berupa *style* untuk dibuat dalam mengizinkan penulisan kode yang lebih mudah dari halaman-halaman *web* yang dirancang (Budi, 2008 dikutip oleh Ishak dan Simin, 2016). CSS bekerja sebagai pelengkap pada HTML (*Hyper Text Markup Language*) dalam memformat dokumen *web* atau untuk mempercantik tampilan *web*. Penulisan kode CSS disisipkan pada *tag* HTML. Penulisan kode CSS dapat

langsung pada dokumen HTML atau disimpan dalam dokumen tersendiri kemudian dipanggil untuk digunakan.

2.3.7 HTML (*Hypertext Markup Language*)

HTML merupakan singkatan dari *Hyper Text Markup Language*, digunakan untuk membangun suatu halaman *web*. Dengan adanya HTML mulai dari teks, gambar, suara, serta *link* dapat digabungkan menjadi satu HTML sebenarnya sama sekali bukan merupakan bahasa pemrograman, karena seperti tercermin dari namanya, HTML adalah sebuah bahasa *mark up* (penandaan) terhadap sebuah dokumen teks yaitu dengan tanda "<..> dan </..>" (Arief, 2011:23 dikutip oleh Muhtar, 2014).

Sebuah file HTML merupakan file teks biasa yang mengandung *tag-tag* HTML. Oleh karena itu, HTML dapat dibuat dengan menggunakan teks editor yang bersifat sederhana seperti notepad pada *windows*. Dapat juga dibuat dengan menggunakan HTML editor yang bersifat visual seperti *frontpage*, *hotmetal*, *netscape composer*, dan lain-lainnya.

2.3.8 UML (*Unified Modeling Language)*

UML (*Unified Modeling Language*) adalah 'bahasa' pemodelan untuk sistem atau perangkat lunak yang berparadigma 'berorientasi objek' (Nugroho, 2010). Pemodelan (*modeling*) sesungguhnya digunakan untuk penyederhanaan permasalahan-permasalahan yang kompleks demikian rupa sehingga lebih mudah dipelajari dan dipahami.

2.3.8.1 *Use Case Diagram*

Use case adalah deskripsi fungsi yang disediakan oleh sistem dalam bentuk teks sebagai dokumentasi dari *use case symbol* namun dapat juga dilakukan dalam *activity diagrams*. *Use case* digambarkan hanya dilihat dari luar oleh *actor* (keadaan lingkungan sistem yang dilihat *user*) dan bukan bagaimana fungsi yang ada dalam sistem (Kusumo, 2004:3). Simbol-simbol yang digunakan pada *use case diagram* disajikan pada Tabel 2.1.

Tabel 2.1. Simbol-simbol *Use Case Diagram*

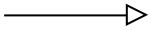
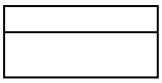
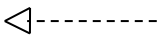
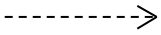
No	Gambar	Nama	Keterangan
1.		*Aktor*	Idealization orang eksternal, proses, atau hal yang berinteraksi dengan sistem,subsistem atau kelas.
2.		*Use case*	Sebuah *use case* menggambar-kan interaksi dengan*actor* sebagai urutan pe san antarasistem dan aktor satu atau lebih.
3.		*System Boundary*	Menspesifikasikan paket yang menampilkan sistem secara terbatas.
4.		*Generalization*	Hubungan antara *use case* umum dan *use case* yang lebih spesifik yang mewarisi dan menambahkan fitur.
5.		*Comunication Association*	Jalur komunikasi antara *actor* dan *use case* yang berpartisipasi didalam
6.		*Extend*	Penyisipan perilaku tambahan kedalam basis *use case* yang tidak tahu tentang hal itu.
7.		*Include*	Penyisipan perilaku tambahan kedalam basis *use case* yang secara eksplisit menggambarkan penyisipan.

2.3.8.2 *Class Diagram*

Class diagram menggambarkan struktur statis *class* di dalam sistem. *Class* merepresentasikan sesuatu yang ditangani oleh sistem. *Class* dapat berhubungan dengan yang lain melalui berbagai cara: *associated* (terhubung satu sama lain), *dependent* (satu *class* tergantung/menggunakan *class* yang lain), *specialed* (satu *class* merupakan spesialisasi dari *class* lainya), atau *package* (grup bersama sebagai satu unit). Sebuah sistem biasanya mempunyai beberapa *class diagram*

(Kusumo, 2004:3). Simbol-simbol yang digunakan pada *class diagram* disajikan pada Tabel 2.2.

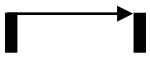
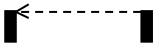
Tabel 2.2. Simbol-simbol *Class Diagram*

No	Gambar	Nama	Keterangan
1.		*Generalization*	Hubungan dimana objek anak (*descendent*) berbagai perilaku dan struktur data dari objek yang ada diatas objek induk (*ancestor*).
2.		*Nary Association*	Upaya untuk menghindari asosiasi dengan lebih dari dua objek.
3.		*Class*	Himpunan dari objek–objek yang terbagi atribut serta operasi yang sama.
4.		*Collaboration*	Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu aktor.
5.		*Realization*	Operasi yang benar-benar dilakukan oleh suatu objek.
6		*Dependency*	Hubungan yang terjadi pada suatu elemen mandiri (*independent*) akan mempengaruhi elemen yang bergantung pada elemen yan tidak mandiri.
7		*Association*	Untuk menghubungkan objek satu dengan objek yang lainnya.

2.3.8.3 *Sequence Diagram*

Sequence diagram menggambarkan kolaborasi dinamis antara sejumlah objek. Kegunaan untuk menunjukan rangkaian pesan yang dikirim antara objek juga interaksi antara objek, sesuatu yang terjadi pada titik tertentu dalam eksekusi sistem (Kusumo, 2004:4). Simbol-simbol yang digunakan pada *sequence diagram* disajikan pada Tabel 2.3.

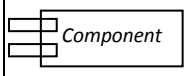
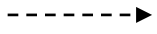
Tabel 2.3. Simbol-simbol *Sequence Diagram*

No	Gambar	Nama	Keterangan
1.		*LifeLine*	Objek *entity*, antarmuka yang saling berinteraksi.
2.		*Message*	*Message* ditampilkan sebagai anak panah dari lifeline dari satu objek ke objek yang lain.
3.		*Return*	Spesifikasi dari komunikasi antar objek yang memuat informasi-informasi tentang aktifitas yang terjadi.

2.3.8.4 *Component Diagram*

Component diagram menggambarkan struktur fisik kode dari komponen. Komponen dapat berupa *sourcecode*, komponent biner, atau *executable component*. Sebuah komponen berisi informasi tentang *logic class* atau *class* yang diimplementasikan sehingga membuat pemetaan dari *logical view* ke *component view* (Kusumo, 2004:4). Simbol-simbol yang digunakan pada component *diagram* disajikan pada Tabel 2.4

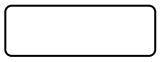
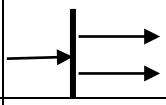
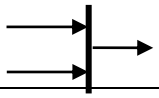
Tabel 2.4. Simbol-simbol *Component Diagram*

No	Gambar	Nama	Keterangan
1.		*Component*	Sebuah komponen melambangkan sebuah entitas *software* dalam sebuah sistem.
2.		*Dependency*	Sebuah *dependency* digunakan untuk menotasikan relasi antara dua komponen.

2.3.8.5 *Activity Diagram*

Activity diagram menggambarkan rangkaian aliran aktifitas, digunakan untuk mendeskripsikan aktifitas yang dibentuk dalam suatu operasi sehingga dapat juga digunakan untuk aktifitas lainnya seperti *use case* atau interaksi (Kusumo, 2004:4). Simbol-simbol yang digunakan pada *use activity diagram* disajikan pada Tabel 2.5.

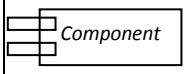
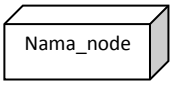
Tabel 2.5. Simbol-simbol *Activity Diagram*

No	Gambar	Nama	Keterangan
1.		*Actifity State*	Aktivitas yang mewakili pelaksanaan dalam pernyataan dalam prosedur atau pelaksanaan kegiatan dalam alur kerja.
2.		*Branch/Merge*	Asosiasi percabangan dimana jika ada pilihan aktivitas lebih dari satu.
3.		*Initial State*	Status awal aktivitas sistem, sebuah diagram aktivitas memiliki sebuah status awal.
4.		*Final State*	Status akhir yang dilakukan sistem.
5.		*Fork*	Percabangan yang menunjukkan aliran pada *activity diagram.*
6.		*Join*	Penggabungan yang menjadi arah aliran pada *activity diagram.*

2.3.8.6 *Deployment Diagram*

Deployment Diagram menggambarkan arsitektur fisik dari perangkat keras dan perangkat lunak sistem, menunjukkan hubungan komputer dengan perangkat (*nodes*) satu sama lain dan jenis hubungannya. Di dalam *nodes*, *executeable component* dan *object* yang dialokasikan untuk memperlihatkan unit perangkat lunak yang dieksekusi oleh *node* tertentu dan ketergantungan komponen (Kusumo, 2004:4). Simbol-simbol yang digunakan pada component *diagram* disajikan pada Tabel 2.6.

Tabel 2.6. Simbol-simbol *Deployment Diagram*

No	Gambar	Nama	Keterangan
1.		*Component*	Komponen-komponen yang ada diletak-kan didalam *node*.
2.		*Node*	*Node* menggambarkam bagian-bagian *hardware* dalam sebuah sistem.
3.		*Association*	Sebuah *association* digambarkan sebagai sebuah garis yang menghubungkan dua *node* yang mengindikasikan jalur komunikasi antara element-elemen *hardware*.

2.3.9 Metode Pengujian *Five View*

Pengujian *five view* adalah pengujian yang sifatnya deskriptif dimana *software* yang diuji dinilai melalui lima sudut pandang atau kategori yang berbeda melalui penilaian dari *expertise* atau orang yang berpengalaman dari masing-masing sudut pandang (Naik dan Tripathy, 2008). Lima sudut pandang tersebut adalah sebagai berikut:

2.3.9.1 *Transcendental View*

Kualitas menurut pandangan ini ialah sesuatu yang dapat dikenali melalui pengalaman tapi tidak dapat digambarkan. Objek atau *software* yang bagus itu menonjol dan dapat dengan mudah dikenali.

2.3.9.2 *User View*

Kualitas menyangkut sejauh mana produk memenuhi kebutuhan dan harapan pengguna dan apakah suatu produk cocok untuk digunakan. Pendapat ini bersifat sangat *personal*. Sebuah produk berkualitas baik jika memuaskan sebagian besar pengguna. Hal ini berguna untuk mengidentifikasi fitur dari produk yang pengguna anggap penting. Pandangan ini dapat mencakup banyak unsure subjek, seperti kegunaan, kendala, dan keefisiensasian.

2.3.9.3 *Manufacturing View*

Pandangan ini berkaitan dengan faktor dalam industri manufaktur, apakah produk memenuhi persyaratan atau tidak. Setiap penyimpangan dari persyaratan yang dinilai mengurangi kualitas produk. Konsep proses memainkan peran kunci. Produk yang dibuat harus orisinil sehingga biaya berkurang, misal biaya pembangunan dan biaya pemeliharaan.

Kesesuaian dengan persyaratan dan spesifikasi menyebabkan keseragaman dalam produk tapi beberapa berbendapat bahwa keseragaman tersebut tidak menjamin kualitas. Kualitas produk dapat secara bertahap ditingkatkan dengan memperbaiki proses.

2.3.9.4 *Product View*

Dalam hal ini, kualitas dipandang dengan karakteristik produk yang melekat. Karakteristik bawaan produk yaitu kualitas internal menentukan kualitas eksternalnya. Jika sebuah produk diproduksi dengan sifat internal (misalnya bahan dan tindakan) yang baik, maka produk akan memiliki sifat eksternal atau *output* yang baik dan dapat dieksplorasi hubungan antara sifat internal dan kualitas eksternal.

2.3.9.5 *Value-based View*

Berapa banyak pengguna bersedia membayar untuk tingkat kualitas tertentu. Kualitas tidak berarti jika produk memenuhi nilai ekonomi. Pandangan berbasis nilai antara biaya dan kualitas.