# BAB II

# LANDASAN TEORI

## Definisi Umum

### Sistem

Sistem adalah suatu kesatuan usaha yang terdiri dari bagian-bagian yang berkaitan satu sama lain yang berusaha mencapai suatu tujuan dalam suatu lingkungan kompleks. Pengertian tersebut mencerminkan adanya beberapa bagian dan hubungan antar bagian, ini menunjukkan kompleksitas dari sistem yang meliputi kerja sama antara bagian yang interpenden satu sama lain (Fatta, 2009).

### Informasi

Informasi adalah data yang sudah dibentuk ke dalam sebuah formulir bentuk yang dapat bermanfaat dan dapat digunakan untuk manusia (Hartono, 2004).

### Manajemen

Manajemen adalah serangkaian aktivitas (termasuk perencanaan dan pengambilan keputusan, pengorganisasian, kepemimpinan, dan pengendalian) yang diarahkan pada sumber-sumber daya organisasi (manusia, finansial, fisik, dan informasi) dengan maksud untuk mencapai tujuan organisasi secara efisien dan efektif (Griffin, 2004).

### PERANCANGAN SISTEM APLIKASI

Perancangan Sistem Aplikasi adalah sebuah sistem informasi yang selain melakukan pengolahan transaksi yang sangat berguna untuk kepentingan organisasi, juga banyak memberikan dukungan informasi dan pengolahan untuk fungsi manajemen dalam pengambilan keputusan (Chr. Jimmy Lbn. Gaol, 2008).

## Definisi Internet

Internet merupakan contoh jaringan yang menghubungkan jutaan komputer yang terbesar di seluruh dunia dan tidak terkait pada suatu organisasi, dalam hal ini, jaringan tersusun atas berbagai jenis komputer dan sistem operasi, supaya dapat berhubungan dengan internet. Seorang pemakai dapat mengakses komputer yang telah terkoneksi ke internet yang telah berlangganan pada sebuah ISP (*Internet Service Provider*). ISP adalah organisasi yang bergerak dalam bidang penyediaan jasa ke internet (Kadir, 2003).

## Sejarah Internet

Sejarah internet dimulai pada Agustus 1962 dan penciptaan internet pertama kali dikemukakan oleh J.C.R Licklider dari MIT *Massachutts Institute of Technology*. Konsep awal dinamakan “*Galactic Network*”. Ia mengemukakan tentang jaringan global yang memungkinkan orang dapat mengakses data dan program dari mana saja. Oktober 1962 beliau mengepalai program penelitian komputer di ARPA yang merupakan bagian dari Departmenet Pertahanan Amerika Serikat. Pada 1965 peneliti dari MIT bernama Lawrence G. Roberts sering juga disebut Larry Roberts dan Thomas Merill melakukan koneksi komputer TX-2 di MIT dengan komputer Q-32 di California menggunakan jalur telpon berkecepatan rendah untuk menciptakan jaringan berskala luas untuk pertama kalinya. Pada tahun 1966 Larry Roberts mengembangkan konsep jaringan komputer kemudian beliau merencanakan jaringan yang disebut *Arpanet* yang dipublikasikan pada tahun 1967. Pada tahun 1969 *Arpanet* telah melibatkan empat buah komputer yang terkoneksi. Komputer pertama berada di *University of California Los Angelos*, komputer ke dua berada di *Stanford Research Institute*, komputer ketiga berada di *University of California Barbara* dan komputer ke empat berada di *University Utah*. Pada tahun 1971 jumlah komputer yang terhubung ke*Arpanet* mencapai 14 buah. Pada tahun ini pulalah protokol *Telnet* dan FTP berhasil dibangun. Pada tahun 1972 larry Roberts dan Bob Kahn mengenalkan *Arpanet* pada konferensi ICCC yang diselenggarakan di *Washington*. Pada tahun 1972 Ray Tomliinson menulis program yang memungkinkan surat elektronik dikirimkan ke jaringan *Arpanet.* Beliaulah yang merancang konversi “*user@host*.” Pada tahun ini pula *Arpanet* menggunakan NCP untuk menstransfer data. Pada tahun yang sama *Arpa* berubah nama menjadi *Darpa*. Tambahan huruf D berasal dari kata *Defense*. Pada tahun ini *Arpanet* melakukan koneksi international yang pertama dengan *University College of London dan Royal Establishment* di Norwegia. Pada tahun 1978 *Unix to Copy Protocol* ditemukan di Labolatorium Bell. Program ini berguna untuk melakukan file transfer.Pada tahun 1979 news group yang diberi nama *Usenet* beroperasi dengan dasar UUCP. Penciptanya adalah Tom Truscott dan Jim Ellis (kedua mahasiswa di *Duke University*) dan Steven Bellovin (dari *University North Carolina*). Pemakai dari seluruh dunia bergabung ke grup diskusi ini membicarakan masalah jaringan, politik, agama dan berbagai topik lainnya,Pada tahun 1982 DCAatau *Defense Communication Agency* dan *Darpa* membentuk protokol yang disebut TCP/IP untuk *Arpanet*. Selanjutnya, Departemen Pertahanan Amerika Serikat menyatakan TCP/IP sebagai sebuah standar. Saat itulah internet didefinisikan sebagai sekumpulan jaringan yang terhubung yang menggunakan TCP/IP sebagai protokol. Pada tahun 1983 John Postel, Paul Mockapetris dan Craig Partidge mengembangkan *Domain Name System* (DNS) dan mengusulkan sistem pengamatan berbentuk *user@host.cdomain*. Pada tahun 1984 DNS diperkenalkan di internet dengan menyebutkan nama-nama jenis domain seperti *.gov, .mil, .org, .net* dan *.com*. Pada tahun 1986 TCP/IP mulai tersedia pada *Workstaiton* dan PC (www.artikeltik.com).

## Konsep Perancangan

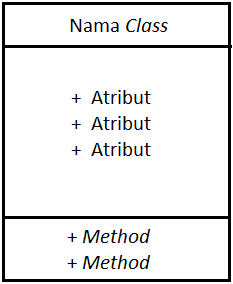
1. Unified Modelling Language (UML )
2. Definisi UML

Pada pengembangan teknologi perangkat lunak, diperlukan adanya bahasa yang digunakan untuk memodelkan perangkat lunak yang akan dibuat dan perlu adanya standarisasi agar orang di berbagai Negara dapat mengerti pemodelan perangkat lunak. Seperti yang kita ketahui bahwa menyatukan banyak kepala untuk menceritakan sebuah ide dengan tujuan untuk memahami hal yang sama tidaklah mudah, oleh karena itu diperlukan sebuah bahasa pemodelan perangkat lunak yang dapat dimengerti oleh banyak orang.

Pada perkembangan teknik pemrograman berorientasi objek, muncullah sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang dibangun dengan menggunakan teknik pemrograman berorientasi objek, yaitu *Unified Modeling Language* (UML). UML muncul karena adanya kebutuhan pemodelan visual untuk menspesifikasikan, menggambarkan, dan dokumentasi dari sistem perangkat lunak. UML merupakan bahasa visual untuk pemodelan dan komunikasi mengenai sebuah *system* dengan menggunakan diagram dan teks-teks pendukung. UML hanya berfungsi untuk melakukan pemodelan. Jadi penggunaan UML tidak terbatas kepada metodologi tertentu, meskipun pada kenyataannya UML paling banyak digunakan pada metodologi berorientasi objek. Seperti yang kita ketahui banyak hal di dunia sistem informasi yang tidak dapat dibakukan, semua tergantung kebutuhan, lingkungan dan konteksnya. Begitu juga dengan perkembangan penggunaan UML bergantung pada level abstraksi penggunannya. Jadi belum tentu pandangan yang berbeda dalam penggunaan UML adalah suatu yang salah, tapi perlu ditelaah dimanakah UML digunakan dan hal apa yang ingin divisualkan. Secara analogi jika dengan bahasa yang kita gunakan sehari-hari, belum tentu penyampaian bahasa dengan puisi adalah hal yang salah. Sistem informasi bukanlah ilmu pasti, maka jika ada banyak perbedaan dan interpretasi di dalam bidang sistem informasi merupakan hal yang sangat wajar (Rosa dan Shalahudin, 2013).

#### *Class Diagram*

Diagram kelas atau *class diagram* menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem.



Gambar 2. 1*Class Diagram*

* + 1. *Class*

*Class* adalah merupakan suatu cetakan untuk menciptakan suatu *instant* dari *object. Class* juga merupakan grup suatu *object*  dengan kemiripan atribut, *behaviour* dan relasi ke *object* lain.

* + 1. Atribut

Atribut adalah *instance* dari *class.* Jika *class* secara umum mempresentasikan secara umum sebuah *object*, sebuah *instance*  adalah representasi nyata dari *class* itu sendiri.

* + 1. *Method*

*Method* adalah merupakan suatu proses operasi berupa fungsi-fungsi yang dapat dikerjakan oleh suatu *object*. *Method* didefinisikan pada *class*akan tetapi dipanggil melalui *object*.

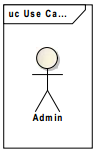
Diagram kelas dibuat agar pembuat program atau *programmer* membuat kelas-kelas sesuai rancangan di dalam diagram kelas agar antara dokumentasi perancangan dan perangkat lunak sinkron (Rosa dan Shalahudin, 2013).

#### Use Case Diagram

*Use case* atau diagram *use case* merupakan pemodelan untuk kelakukan (*behavior*) sistem informasi yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. secara kasar, *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi itu.

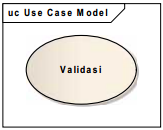
Syarat penamaan pada *use case* adalah nama didefinisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada *use case* yaitu pendefinisian apa yang disebut aktor dan *use case* (Rosa dan Shalahudin, 2013).

1. *Actor* merupakan orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang.



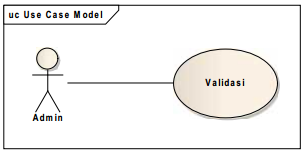
Gambar 2. 2*Actor*

1. *Use case* merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor.



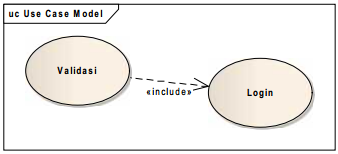
Gambar 2. 3*Use Case*

1. *Assosiation* merupakan komunikasi antara *actor* dan *use case* yang berpartisipasi pada *use case diagram*



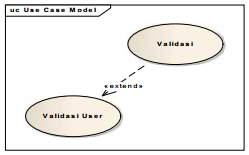
Gambar 2. 4*Assosiation*

1. *Include* relasi *use case* master dibutuhkan untuk memasukkan data ke *use case* transaksi. Karena data yang dimasukkan ke dalam *use case* transaksi berdasarkan data *use case* master.



Gambar 2. 5*Include*

1. *Extends* Relasi *use case* tambahan ke sebuah *use case* di mana *use case* yang ditambahkan dapat berdiri sendiri walau tanpa *use case* validasi dapat berdiri sendiri walau tanpa *use case* validasi*.*

**

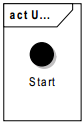
Gambar 2. 6 *Extends*

#### Activity Diagram

Diagram aktivitas atau *activity diagram* menggambarkan *workflow* (aliran kerja) atau aktifitas dari sebuah sistem atau proses bisnis atau menu yang ada pada perangkat lunak. Yang perlu diperhatikan di sini adalah bahwa diagram aktivitas menggambarkan aktivitas sistem bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem. *Activity diagram* memiliki beberapa komponen, yaitu:

* + 1. *Initial*

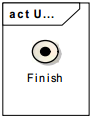
*Start point* (*initial node*), menggambarkan permulaan dari sebuah sistem yang akan dikerjakan, biasanya diletakkan pada pojok kiri.



Gambar 2. 7*Initial*

* + 1. *End Point*

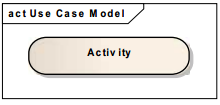
*End Point* (*activity final node*), menggambarkan akhir dari sebuah sistem pada *activity diagram.*



Gambar 2. 8*End Point*

* + 1. *Activity*

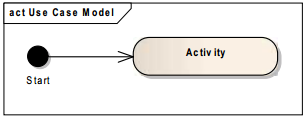
*Activity* menggambarkan suatu tindakan, perilaku atau proses yang terdapat pada lingkungan sistem.



Gambar 2. 9*Activity*

* + 1. *Black hole activities*

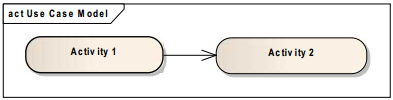
*Black hole activities* yaitu menggambarkan aktifitas yang menerima masukan dan keluaran.

**

Gambar 2. 10*Black Hole Activities*

* + 1. *Miracle Activities*

Miracle activities yaitu menggambarkan tidak adanya masukan tetapi ada keluaran dan dipakai pada waktu *start point*.

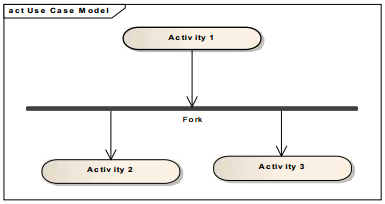


Gambar 2. 11*Miracle Activity*

* + 1. *Parallel Activities*

*Parallel activities* yaitu *activity* yang berjalan secara bersamaan terdiri dari:

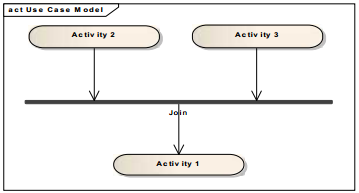
* + - 1. *Fork* (percabangan),mempunyai 1 transisi masuk dan 2 atau lebih transisi keluar.



Gambar 2. 12*Fork*

* + - 1. *Join* (penggabungan), mempunyai 2 atau lebih transisi masuk dan

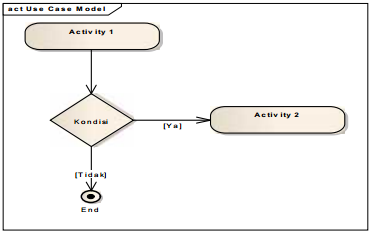
hanya 1 transisi keluar.



Gambar 2. 13*Join*

* + 1. *Decision Point*

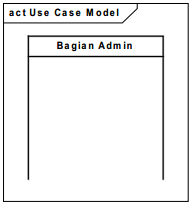
*Decision points,* tidak ada keterangan (pertanyaan) pada tengah belah ketupat seperti pada *flowchart* dan harus mempunyai *guards*( kunci ).



Gambar 2. 14*Decision*

* + 1. *Swimlane*

*Swimlane,* yaitu sebuah cara untuk mengelompokkan *activity* berdasarkan *actor.*

**

Gambar 2. 15 *Swimlane*

Dengan aktivitas juga banyak digunakan untuk mendefinisikan hal-hal berikut:

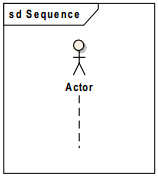
1. Rancangan proses bisnis di mana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem yang didefinisikan.
2. Urutan atau pengelompokkan tampilan dari sistem / *user interface* di mana setiap aktivitas dianggap memiliki sebuah rancangan antarmuka tampilan.
3. Rancangan pengujian dimana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya.
4. Rancangan menu yang ditampilkan pada perangkat lunak.

#### *Sequence Diagram*

Diagram sekuen menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambar diagram sekuen maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu. Membuat diagram sekuen juga dibutuhkan untuk melihat skenario yang ada pada *use case. Sequence diagram* mempunyai beberapa simbol, yaitu:

1. *Actor*

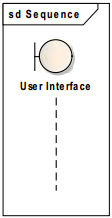
Orang atau sistem lain yang berinteraksi dengan sistem atau menggunakan sistem.



Gambar 2. 16 *Actor* pada *Sequence diagram*

1. *Boundary*

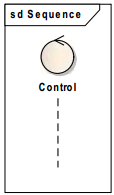
Suatu *Boundary* adalah suatu kelas yang meniru beberapa model *boundary* sistem yang secara khas yaitu merupakan alat penghubung seseorang dengan *user interface. Boundary* dapat diartikan sebagai antarmuka bagi pengguna, sebagai perangkat keras dan panel-panel yang mengendalikan keadaan sistem.



Gambar 2. 17*Boundary*

1. *Control*

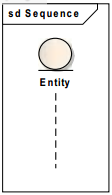
Suatu *control* adalah meniru suatu kelas yang menggambarkan suatu pengendalian manajer atau kesatuan. *Control* mengorganisir dan menjadwalkan aktivitas elemen-elemen.

**

Gambar 2. 18*Control*

1. *Entity*

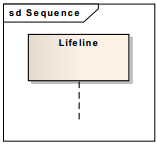
*Entity* adalah menggambarkan suatu objek dari data yang terdapat dalam sistem.

**

Gambar 2. 19 *Entity*

1. *Lifeline*

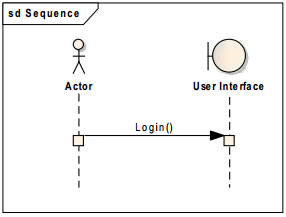
*Lifeline* adalah elemen individu di dalam suatu interaksi (dengan kata lain *lifeline* tidak bisa mempunyai keberagaman). *Lifeline* menggambarkan hubungan suatu elemen yang berbeda*.*

**

Gambar 2. 20 *Lifeline*

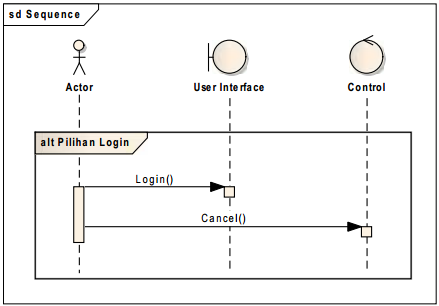
1. *Message*

Pesan menandai adanya suatu alur informasi atau transisi kendali antara elemen-elemen.



Gambar 2. 21 *Message*

1. *Fragment*

Suatu *fragment* mencerminkan suatu potongan atau potongan interaksi yang dikendalikan oleh suatu operator interaksi.

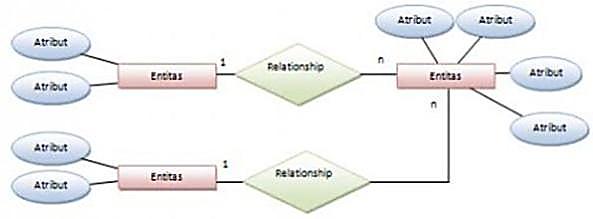
Gambar 2. 22 *Fragment*

Banyaknya diagram sekuen yang harus digambar adalah minimal sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram sekuen sehingga semakin banyak *use case* yang didefinisikan maka diagram sekuen yang harus dibuat juga semakin banyak (Rosa dan Shalahudin, 2013)

## Basis Data

1. Entity Relationship Diagram ( ERD)

ERD adalah gambar atau diagram yang menunjukkan informasi dibuat, disimpan dan digunakan dalam sistem bisnis (Fatta, 2007).



Gambar 2. 23 ERD

Dalam pembentukan ERD terdapat tiga komponen yang akan di bentuk yaitu:

1. Entitas

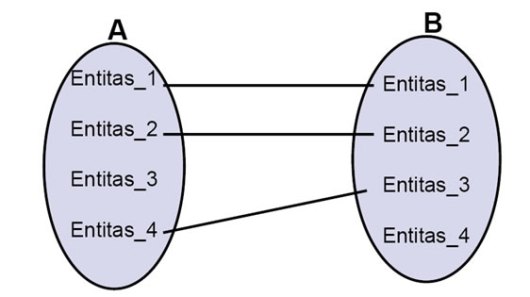
Suatu objek yang dapat di bedakan dari lainnya yg dapat di wujudkan dalam basis data. Pengertian lainnya menurut Brandy dan Loonam (2010), entitas adalah objek yang menarik dibidang organisasi yang dimodelkan.

1. Relasi

Hubungan antara dua jenis entitas dan direpresentasikan sebagai garis lurus yang menghubungkan dua entitas. Relasi terdiri dari:

1. *One to one* (1:1)

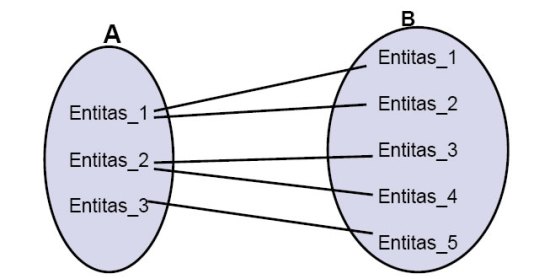
Setiap anggota entitas A hanya boleh berhubungan dengan satu entitas B, begitu pula sebaliknya.



Gambar 2. 24 *Relasi One to one*

1. *One to many* (1:M)

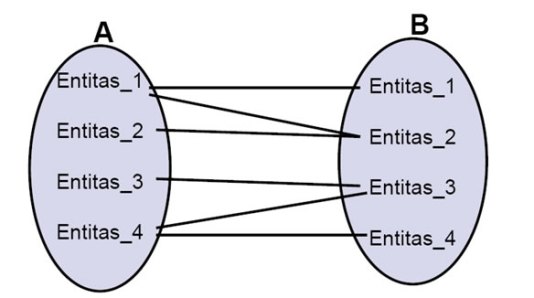
Setiap anggota entitas A Dapat berhubungan lebih dari satu anggota entitas B tetapi tidak sebaliknya.



Gambar 2. 25 Relasi *One to many*

1. *Many to many* (M:M)

Setiap entitas A dapat berhubungan dengan banyak himpunan entitas B dan demikian pula sebaliknya.



Gambar 2. 26 Relasi *Many to many*

1. Atribut

Atribut memberikan informasi lebih rinci tentang jenis entitas. Atribut memiliki struktur internal berupa tipe data. Jenis-jenis atribut:

1. Atribut *Key*

Atribut *key* adalah satu atau gabungan dari beberapa atribut yang dapat membedakan semua baris data (*Row*/*Record)* dalam tabel secara unik. Dikatakan unik jika pada atribut yang dijadikan *key* tidak boleh ada baris data dengan nilai yang sama.

1. Atribut *Simple*

Atribut yang bernilai atomic, tidak dapat di pecah / dipilah lagi.

1. Atribut *Multivalue*

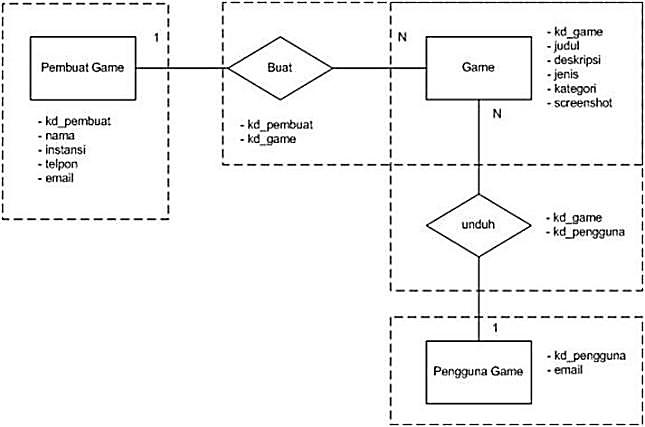
Nilai dari suatu atribut yang mempunyai lebih dari satu (*Multivalue*) nilai dari atribut yang bersangkutan.

1. Atribut *Composite*

Atribut *composite* adalah suatu atribut yang terdiri dari beberapa atribut yang lebih kecil yang mempunyai arti tertentu yang masih bisa dipecah lagi atau mempunyai sub atribut.

### Transformasi ERD ke LRS

Transformasi diagram ERD ke LRS merupakan suatu kegiatan untuk membentuk data-data dari diagram hubungan etitas ke suatu LRS. Transformasi ERD ke LRS adalah sebagai berikut: Penggabungan *entity* dengan Jadwal Pada relasi Dapat merupakan penggabungan antara *entity* Calon dan Jadwal, relasi tersebut akan dikonversikan ke LRS dan di gabungkan ke *entity* Jadwal karena *entity* jadwal memiliki kardinalitas relasi many (Latifah & Fithri, 2012).

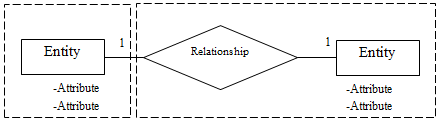


Gambar 2. 27 Transformasi ERD ke LRS

Aturan dalam transformasi adalah:

1. Transformasi 1 : 1

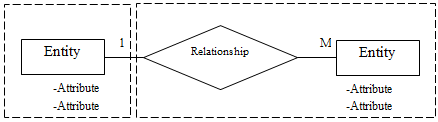
Ke arah *week entity* dan ke *entity*  yang membutuhkan referensi atau *entity* dengan jumlah atribut yang lebih sedikit.



Gambar 2. 28 Transformasi 1 : 1

1. Transformasi 1 : m

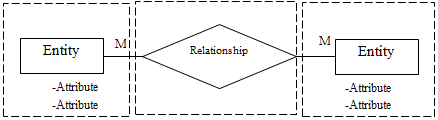
Tidak perlu melihat jumlah jumlah atribut yang lebih sedikit dan selalu digabung ke arah *many.*



Gambar 2. 29 Transformasi 1 : m

1. Transformasi m : m

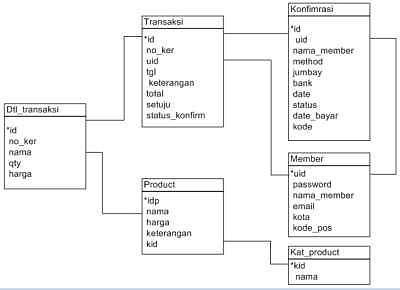
*Relationship* berdiri sendiri. Tiap entity menjadi sebuah *logical record structure. Relationship* juga menjadi sebuah *logical record structure.*



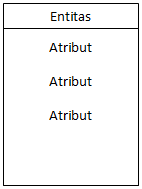
Gambar 2. 30 Trasformasi m : m

### *Logical Record Structure* ( *LRS*)

*Logical Record Structure* dibentuk dengan nomor dari tipe *record*. Beberapa tipe *record* digambarkan oleh kotak empat persegi panjang dan dengan nama yang unik. Perbedaan LRS dengan E-R diagram adalah nama tipe *record* berada diluar kotak *field* tipe *record* ditempatkan. *Logical Record Structure* terdiri dari *link-link* diantara tipe *record*. Link ini menunjukkan arah dari satu tipe *record* lainnya. Banyak *link* dari LRS yang diberi tanda *field-field* yang kelihatan pada kedua *link* tipe *record*. Penggambaran LRS mulai dengan menggunakan model yang dimengerti. Dua metode yang dapat digunakan, dimulai dengan hubungan kedua model yang dapat dikonverensikan ke LRS, metode yang lain dimulai dengan ER-diagram dan langsung dikonversikan ke LRS (Tabrani, 2014).



Gambar 2. 31 LRS



Gambar 2. 32 Kotak Entitas LRS

Dalam kaitannya dengan konversi ke *Logical Record Structure,* untuk perubahan yang terjadi adalah mengikuti aturan-aturan berikut:

1. Setiap entitas diubah kebentuk kotak dengan nama entitas, berada di luar kotak dan atribut berada di dalam kotak.
2. Sebuah *relationship* kadang disatukan dalam sebuah kotak bersama entitas, kadang sebuah kotak bersama-sama dengan entitas, kadang disatukan dalam sebuah kotak sendiri.

### Tipe Data

Tipe data adalah sejenis data yang memiliki batasan tempat dan karakteristik yang sesuai dengan interprestasi data itu sendiri dan bisa diolah oleh sebuah komputer agar dapat memenuhi kebutuhan pemrograman komputer. Tipe data juga ialah termasuk tipe data primitif, yaitu *character, floating point number* dan *interger* (*string*). Sekumpulan dari tipe data primitif yang termasuk sejenis, dan juga dapat disatukan dalam bentuk blok yang biasa disebut *array*.

Dan setiap *variable* yang ada di dalam kode program, sebaiknya anda tentukan sendiri dengan pasti pada tipe datanya. Ketepatan dalam pemilihan tipe data pada variable baisanya sangat menentukan pemakaian sumberdaya komputer (terutama pada memori komputer itu sendiri). Salah satu tugas yang tidak kalah pentingnya ialah seorang programmer harus memilih tipe data yang sesuai agar menghasilkan program yang baik atau biasa juga diartikan efisien dan berkinerja tinggi.

1. Tipe data *primitive* (Sederhana)

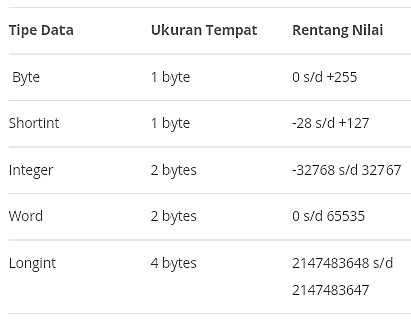
Adalah tipe data yang hanya mampu menyimpan satu nilai tiap satu variabelnya. Tipe data *primitive* merupakan tipe data dasar yang sering dipakai oleh program. Contoh tipe data *primitive* adalah tipe numerik (*integer* dan *real*), tipe data karakter/*char*, tipe data *boolean*.

1. *Numeric*

Tipe data *numeric* digunakan pada variabel atau konstanta untuk menyimpan nilai dalam bentuk bilangan atau angka. Semua bahasa pemrograman menyediakan tipe data *numeric*, hanya saja berbeda dalam jenis *numeric* yang diakomodasi. Tipe data ini terbagi atas *integer*, dan *real.*

* + 1. *Integer*

Merupakan tipe data berupa bilangan bulat, terbagi atas beberapa kategori seperti terlihat dalam tabel 2.1. Tabel 2.1 menunjukkan jenis data, ukuran dalam memori dan rentang nilainya.

Tabel 2.1

Tabel 2. 1 *Integer*

* + 1. *Real*

Konstanta bertipe real adalah bilangan yang berisi titik desimal atau jenis bilangan pecahan. Dalam Pascal paling sedikit harus ada satu digit sebelum dan sesudah titik desimal, tidak boleh ada koma dan nilainya bisa positif atau negatif. Dapat dituliskan secara biasa atau model *scientific*. Contoh bilangan *real*: 34.265 -3.55 0.0 35.997E+11, di mana E merupakan simbol perpangkatan 10. Jadi 452.13 mempunyai nilai sama dengan 4.5213e2.

Tabel 2.2 Bilangan *Real*

Tabel 2. 2 Bilangan *Real*

1. Karakter

Nilai-nilai yang termasuk karakter adalah:

* + Karakter huruf : ‘a’..’z’,’A’..’Z’
  + Karakter angka : ‘0’..’9’
  + Karakter tanda baca : titik, koma, titik koma, titik dua dan sebagainya
  + Karakter khusus : $, %, #, @ dan sebagainya.

1. Boolean

Merupakan tipe data logika, yang berisi dua kemungkinan nilai: *True* (benar) atau *False* (salah).

1. Tipe data *Composite*

*Composite Data Types* atau Tipe Data Komposit merupakan tipe data yang dapat menampung banyak nilai.

* + - 1. *Array*

*Array* atau sering disebut sebagai larik, adalah tipe data yang sudah terstruktur dengan baik, meskipun masih sederhana *Array* mampu menyimpan sejumlah data dengan tipe yang sama (homogen) dalam sebuah variabel. Sebagai ilustrasi, *array* mampu menampung banyak data namun dengan satu tipe data yang sama, misalnya integer saja. Setiap lokasi data array diberi nomor *indexs* yang berfungsi sebagai alamat dari data tersebut.

* + - 1. *Record* atau *struct*

Seperti halnya a*rray, record* atau *struct* juga termasuk tipe data komposit. *Record* dikenal dalam bahasa *pascal/pelphi* sedangkan *struct* dikenal dalam bahasa C++. Berbeda dengan *array,* tipe data *record* mampu menampung banyak data dengan tipe data berbeda-beda (heterogen). Misalnya, satu bagian integer, satu bagian lagi *character*, dan bagian lainnya *boolean*. Biasanya *record* digunakan untuk menampung data suatu obyek. Misalnya, siswa memiliki nama, alamat, usia, tempat lahir, dan tanggal lahir. Nama akan menggunakan tipe data string, alamat bertipe data *string,* usia bertipe data *single* (*numeric*), tempat lahir bertipe data *string*, dan tanggal lahir bertipe data *date*.

* + - 1. *Image*

*Image*, atau gambar, atau citra, merupakan tipe data grafik. Misalnya grafik perkembangan jumlah siswa SMK, foto keluarga kita, video perjalanan, dan lain-lain. Pada bahasa-bahasa pemrograman modern terutama yang berbasis *visual*, tipe data ini telah didukung dengan sangat baik

* + - 1. *Date Time*

Nilai data untuk tanggal (*date*) dan waktu (*time*) secara internal disimpan dalam format yang spesifik. Variabel atau konstanta yang dideklarasikan dengan tipe data *Date* dapat digunakan untuk menyimpan, baik tanggal maupun jam. Tipe data ini masuk dalam kelompok tipe data *composite*, karena merupakan bentukan dari beberapa tipe data.

* + - 1. *Subrange*

Tipe data subrange merupakan tipe data bilangan yang mempunyai jangkauan nilai tertentu sesuai dengan yang ditetapkan programmer. Biasanya, tipe data ini mempunyai nilai batas minimum dan nilai batas maksimum.

* + - 1. Enumerasi

Tipe data ini merupakan tipe data yang mempunyai elemen-elemen yang harus disebut satu persatu, dan bernilai konstanta integer sesuai dengan urutannya. Nilai konstanta *integer* elemen ini diwakili oleh suatu nama variable yang ditulis di dalam kurung.

### Normalisasi

Normalisasi adalah proses mendesain struktur database dan teknik analisis data yang terorganisasikan atribut data dengan cara mengelompokkan sehingga terbentuk entitas yang non-redundant, stabil, serta fleksibel sehingga menghasilkan sebuah tabel normal. Difungsikan untuk memastikan bahwa *database* yang dibuat berkualitas sangat baik. Tahap normalisasi dimulai dari tahap paling ringan (1NF) sampai tingkat 3-NF, karena sudah cukup memadai untuk menghasilkan tabel-tabel yang berkualitas baik. Tahapan dalam Normalisasi adalah:

1. Bentuk tidak normal

Bentuk ini iyalah kumpulan data yang akan direkam, tidak ada keharusan mengikukti format tertentu, bisa saja data tidak lengkap atau terduplikasi(rangkap).

1. 1-NF (Bentuk normal pertama )

Ciri-cirinya adalah Setiap data di bentuk dalam *flat file* dan Tidak ada *set atribut* yang berulang–ulang. Membentuk sebuah tabel menjadi *Un-Normalized*, dengan mencantumkan data yang ada pada semua *field*. Membentuk normal pertama dengan *flat table* yang memisahkan data pada *field-field* yang tepat bernilai atomik, dan yang melengkapi atribut/ *field* yg ada.

1. 2-NF (Bentuk normal kedua)

Bentuk normal kedua ini iyalah didasarinya atas konsep *full functional dependency* (ketergantungan fungsional sepenuhnya) yang dapat didefinisikan.

1. 3-NF (Bentuk normal ketiga)

Walaupun relasi 2-NF memiliki redudansi yang lebih sedikit dari relasi 1-NF, namun relasi tersebut masih mendapat kemungkinan mengalami kendala bila terjadi *anomaly* peremajaan (*update*) terhadap relasi tersebut (Kadir, 2009).

## *Software* Pendukung

1. SQL SERVER

Menurut (Kuniyo dan Kusrini, 2007) dalam bukunya yang berjudul Membangun Sistem Informasi Akuntansi dengan Visual Basic & SQL Server, mendefinisikan bahwa: SQL Server adalah perangkat lunak relation database management system (RDBMS) yang didesain untuk melakukan proses manipulasi database berukuran besar dengan berbagai fasilitas.”

### C SHARP

C # (tanda “#” dibaca “Sharp”) merupakan bahasa pemrograman baru yang diciptakan Microsoft secara khusus sebagai salah satu bahasa pemrograman dalam teknologi .Net. Sebagai bahasa baru. C# tidak berevolusi dari bahasa C# versi bukan teknologi .Net. Dengan demikian, C # dapat memaksimalkan kemampuannya tanpa khawatir dengan masalah kompatibilitas dengan versi - versi sebelumnya. Keharusan sebuah perangkat lunak untuk tetap dapat kompatibel dengan versi - versi sebelumnya sebagaimana yang terjadi pada Visual Basic (VB) maupun C++ biasanya menghambat optimalitas kemampuan dari perangkat lunak tersebut.

Sejak diluncurkan pada tahun 2000, C# dengan cepat merebut hati programmer C++ bahkan VB. Dengan tata Cara penulisan yang mirip C ++ dan user interface yang mirip VB 6.0 menurut Wikipedia, sebuah ensiklopedia gratis di internet pengguna C# .Net pada saat ini dapat dikatakan sudah melebihi jumlah pengguna VB.Net. Sementara itu, jumlah pengguna bahasa pemrograman lain masih berada di bawah jumlah pengguna VB.Net. Masih menurut Wikipedia, jumlah buku C # yang terjual pun berada di kisaran 2 hingga 3 kali lebih banyak dari jumlah buku VB yang terjual.

Dari informasi ini dapat disimpulkan bahwa C# merupakan bahasa pernrograman baru yang sedang berkembang dan dapat diterima dengan baik oleh kebanyakan programmer dan kalangan industri. Di Microsoft sendiri, C# merupakan bahasa pemrograman yang digunakan untuk membuat perangkat lunak yang berteknologi .Net. Dengan demikian, dapat diperkirakan bahwa C# akan menjadi bahasa pemrograman yang akan banyak digunakan di masa-masa mendatang.

Jika program aplikasi yang dibuat di masa mendatang akan banyak dibuat dengan C#, maka alangkah baiknya jika pada saat itu Anda sudah memahami dan menguasai C# dengan baik. Buku ini merupakan buku pertama dari sekumpulan buku C# yang akan membahas secara tuntas semua kemampuan C#. Melalui buku ini Anda akan belajar tentang konsep dasar pembuatan program berbasis Windows. Dengan menguasai konsep dasar ini, Anda dapat membuat berbagai program-program lain yang juga berbasis Windows.

Istilah .Net (baca: dot Net) mungkin cukup sering Anda dengar belakangan ini. Bahkan bukannya tidak mungkin, saat buku ini ada di tangan Anda, istilah tersebut sudah sedemikian menjamurnya sehingga menggerakkan Anda untuk mencari informasi tentang apa sebenarya .Net tersebut. Secara singkat, .Net adalah teknologi yang dikembangkan oleh perusahaan perangkat lunak Microsoft untuk memfasilitasi pembuatan program yang dapat dijalankan pada berbagai jenis komputer, sistem operasi, hingga ke peralatan-peralatan lain seperti handphone dan sebagainya. Dengan demikian, jika Anda bermaksud untuk membuat program yang dapat dijalankan melalui internet, jaringan komputer, handphone, dan lain-lain, maka sangatlah tepat jika Anda membuatnya melalui teknologi .Net ini.

Teknologi .Net dapat juga digunakan bagi Anda yang ingin membuat program untuk komputer pribadi yang dijalankan secara individual. Dibandingkan dengan perangkat lunak pembuat program yang telah ada sebelumnya seperti Ms.Visual Studio 6.0 (Ms. merupakan singkatan yang umum digunakan untuk perusahaan Microsoft). Perangkat lunak pembuat program berbasis .Net mempunyai lebih banyak fasilitas dan perintah - perintah yang akan mempermudah dan mempercepat Anda dalam pembuatan program aplikasi. Selain itu, perangkat lunak pembuat program berbasis Net diciptakan dengan dasar pemrograman berorientasi objek sehingga program aplikasi akan menjadi lebih mudah dibuat, dipelihara, dan dikembangkan. Beberapa kelebihan lain yang dapat Anda peroleh melalui teknologi .Net ini adalah:

1. Kemudahan untuk rnenggabung beberapa program yang dibuat dengan berbagai bahasa pernrograman berteknologi .Net
2. Kemudahan untuk berpindah antarbahasa pemrograman.
3. Peningkatan keandalan program dalam menangani error
4. Penanganan masalah memori secara otomatis dan sebagainya.

Beberapa perangkat lunak pembuat program yang menggunakan teknologi .Net adalah Visual C# .Net (tanda # dibaca “sharp”), Visual Basic Net, Visual C++ .Net, dan Visual J# .Net. Dalam versi yang dikeluarkan pada tahun 2002, 2003, dan 2005, Microsoft menggabungkan Visual C# .Net, Visual Basic .Net, Visual C++ .Net, dan Visual J# .Net menjadi satu-kesatuan perangkat lunak yang disebut dengan Ms.Visual Studio .Net. Dengan demikian, jika Anda membeli clan menginstal Ms.Visual Studio .Net pada komputer Anda, maka Anda akan dapat membuat program dengan Visual C# Net, Visual Basic .Net, Visual C++ .Net, atau Visual J# .Net.

Karena alasan tertentu. pada tahun 2005 Microsoft juga mengeluarkan Visual Studio .Net versi Express Edition yang dapat diambil secara gratis melalui internet. Pada Visual Studio .Net versi Express Edition masing - masing perangkat lunak, yaitu Visual C# .Net, Visual Basic .Net, Visual C++ .Net, Visual J# .Net, dan Visual Web Developer (untuk pembuatan program ke Web) tidak lagi dikemas dalam satu-kesatuan perangkat lunak Visual Studio .Net, tetapi per masing-masing perangkat lunak. Dengan demikian, jika Anda ingin menggunakan Visual C# .Net, misalnya, maka Anda harus men-download dan menginstal perangkat lunak itu saja. Demikian juga jika Anda ingin menggunakan perangkat lunak berbasis teknologi .Net lainnya.

Untuk menjalankan program aplikasi yang dibuat dengan teknologi .Net, Anda perlu menginstal suatu komponen yang disebut .Net Framework. Pada saat buku ini ditulis, .Net Framework yang paling baru adalah .Net Framework versi 3.5. Komponen ini bahkan sudah disertakan bersama Windows Vista sehingga Anda tidak perlu menginstalnya secara terpisah jika sistem operasi komputer Anda sudah menggunakan Windows Vista.

Spesiﬁkasi .Net Framework telah distandarkan melalui suatu badan intemasional. Vendor lain yang ingin membuat .Net Framework untuk sistem operasi lain harus mengikuti standar yang telah ditetapkan tersebut. Dengan demikian, jika misalnya .Net Framework untuk sistem operasi X telah diciptakan, maka semua program yang dibuat dengan teknologi .Net akan. dapat dijalankan secara langsung pada komputer yang menggunakan sistem operasi X tersebut tanpa harus melakukan perubahan pada program. Jadi. dapat disimpulkan bahwa pembuatan program melalui teknologi .Net akan membuat program Anda menjadi lebih ﬂeksibel karena program tersebut dapat dijalankan pada berbagai perangkat keras dan sistem operasi selama .Net Framework telah terpasang pada perangkat keras tersebut.

### Microsoft Visual Studio .Net

Visual Studio .Net merupakan suatu perangkat lunak yang dapat memudahkan programmer dalam membuat program-program yang berteknologi .Net. Melalui Visual Studio .Net, programmer dapat mengetikkan program pada editor yang tersedia, menggunakan fasilitas untuk mencari kesalahan program, menciptakan user interface seperti pembuatan menu, form, dan sebagainya secara interaktif, membentuk (build) program menjadi ﬁle .exe, hingga menjalankan program yang telah dibuat. Semua fasilitas yang disediakan Visual Studio tersebut terangkum menjadi satu-kesatuan aplikasi yang disebut dengan istilah *Visual Studio Integrated Development Environment* (VS-IDE). Secara sederhana, istilah Visual Studio IDE dapat diartikan sebagai lingkungan atau tempat pembuatan program yang terintegrasi pada Visual Studio.

Pada saat ini ada empat bahasa pemrograman yang didukung oleh Visual Studio .Net. Keempat bahasa tersebut adalah C#, C++, Basic, dan Java. Pada bab ini kita akan melihat alasan mengapa Anda sebaiknya tahu dan Dapat menggunakan bahasa Visual C# dibandingkan dengan bahasa lain juga didukung oleh Visual Studio .Net. Selain itu, kita juga akan meliat beberapa bagian penting dari antarmuka (user-interface) Visual Studio .Net.

Visual Studio .Net pertama kali diluncurkan pada tahun 2002 bersama dengan .Net Framework versi 1.0. Setelah itu, Visual Studio .Net diperbarui lagi pada tahun 2003 dengan .Net Framework versi 1.1. dan tahun 2005 dengan .Net Framework versi 2.0. Pada saat buku ini ditulis microsoft sedang mempersiapkan peluncuran Visual Studio 2008 dengan .NET framework versi 3.5. Dalam semua Visual Studio .Net tersebut.

Terdapat empat bahasa pemrograman yang didukung olch Microsoft. Keempat bahasa pemrograman tersebut adalah C#, C++, Basic, dan Java (Hartanto, 2008).

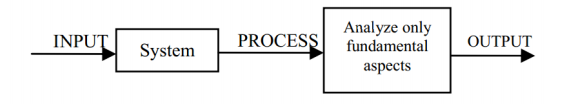
## Metode Pengujian

Pengujian perangkat lunak adalah sangat diperlukan dalam suatu sistem informasi, dimana dengan melakukan suatu pengujian akan ditemukan kesalahan atau error yang muncul dari system perangkat lunak tersebut.Dengan demikian tentunya seorang programmer akan bisa mengetahui dan apa yang harus dikerjakan selanjutnya. Pentingnya pengujian perangkat lunak dan implikasinya yang mengacu pada kualitas perangkat lunak tidak dapat terlalu ditekan karena melibatkan sederetan aktivitas produksi dimana peluang terjadinya kesalahan manusia sangat besar dan arena ketidakmampuan manusia untuk melakukan dan berkomunikasi dengan sempurna maka pengembangan perangkat lunak diiringi dengan aktivitas jaminan kualitas (Rouf, 2009).

Sebelum melakukan pengujian perlu dipersiapkan *Test CaseTest Case* terlebih dahulu agar diperoleh terlebih dahulu agar diperoleh kemungkinan tertinggi dalam menemukan kemungkinan tertinggi dalam menemukan kesalahan dengan waktu dan usaha yang kesalahan dengan waktu dan usaha yang minimum. Desain *Test Case* dapat dilakukan minimum. Desain *Test Case* dapat dilakukan melalui berbagai teknik pengujian diantaranya adalah:

1. *Black Box*

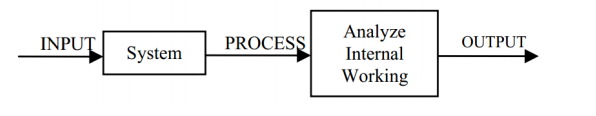
pengujian untuk mengetahui apakah semua fungsi perangkat lunak telah berjalan semestinya sesuai dengan kebutuhan fungsional yang telah didefinsikan. Metode *Black Box* memungkinkan perekayasa perangkat lunak mendapatkan serangkaian kondisi input yang sepenuhnya menggunakan semua persyaratan fungsional untuk suatu program. *Black Box* dapat menemukan kesalahan dalam kategori berikut:

1. Fungsi-fungsi yang tidak benar atau hilang
2. Kesalahan *interface*
3. Kesalahan dalam strutur data atau akses basis data eksternal
4. Inisialisasi dan kesalahan terminasi
5. Validitas fungsional
6. Kesensitifan sistem terhadap nilai *input* tertentu
7. Batasan dari suatu data

Gambar 2. 33 Sistem kerja *Black Box*

1. *White Box*

Pengujian untuk memperlihatkan cara kerja dari produk secara rinci sesuai dengan spesifikasinya. Metode pengujian dengan menggunakan struktur kontrol program untuk untuk memperoleh kasus uji. Dengan menggunakan *white box* akan didapatkan kasus uji seperti berikut:

1. Menjamin seluruh jalur independen didalam modul yang dieksekusi sekurang-kurangnya sekali
2. Menguji semua keputusan logikal
3. Menguji seluruh *Loop* yang sesuai dengan batasannya
4. Menguji seluruh struktur data internal yang menjamin validitas

Gambar 2. 34 Sistem kerja *White Box*