## BAB III

**TINJAUAN PUSTAKA**

**3.1. Penelitian Sebelumnya**

a. Menurut (Handayani, 2021) di dalam jurnal “*Sistem Informasi Manajemen Aplikasi Rumah Kost Dengan Menggunakan Rapid Application Development Berbasis Android Dan Sms Gateway”* di Rumah Kost yang berada diwilayah Kota Bekasi belum memiliki sistem informasi manajemen mengenai ketersediaan, lokasi, fasilitas serta belum adanya sistem pembayaran secara online. Untuk mengatasi permasalahan tersebut penulis membangun sistem informasi manajemen berbasis android dan sms *gateway* dengan menggunakan metode Rapid Application Development, aplikasi dibangun menggunakan bahasa pemrograman *Java, Android Studio, Sublime* dan *database MySql.*

b. Menurut (Rivaldi, 2021) pada jurnal “*Sistem Manajemen Penyewaan Kamar Kos Berbasis Web Web-Based Boarding House Rental Management System”* pada persewaan kamar kos sebagian masih dilakukan dengan sistem manual. Sistem manual ini mempunyai beberapa kendala, diantaranya pengecekan kamar kosong, pencatatan data penyewa baru, pencatatan dan pelaporan keuangan relatif lebih lama, lebih sulit, dan kurang akurat. Untuk mengatasi permasalahan tersebut penulis membangun sistem dalam penelitian ini dengan Analisis dan perancangan sistem dilakukan menggunakan UML, sistem dibangun menggunakan bahasa pemrograman PHP dan MySQL sebagai server database, serta mengintegrasikan google maps API sebagai media pemetaan lokasi.

c. Menurut (Al Ghiffari et al., 2020) di dalam jurnal “*Perancangan Sistem Informasi Manajemen Pengelolaan Hunian Sewa Berbasis Website Pada Kos-kosan Pondok Salma*” Sistem manajemen yang diterapkan pada hunian sewa ini masih dengan sistem konvensional, seperti pembayaran dan laporan keluhan masih melalui penjaga hunian sewa tersebut. Sistem

yang diterapkan sekarang dianggap kurang efektif untuk mengolah segala kegiatan hubungan antara penghuni hunian sewa dan penjaga hunian sewa ataupun pemilik dari hunian sewa tersebut. Untuk menanggapi permasalahan tersebut, dilakukan perancangan sistem informasi manajemen untuk pengelolaan hunian sewa berbasis website menggunakan metode Agile Development. Perancangan sistem juga dilakukan dengan menggunakan Unified Modelling Language.

* 1. **Pengertian Sistem**

Secara etimologi, Sistem berasal dari bahasa Latin (systcma) dan bahasa Yunani (sustcma) yang berarti suatu kesatuan yang terdiri komponen atau elemen yang dihubungkan bersama untuk memudahkan aliran informasi, materi atau energi untuk mencapai tujuan (A. S. Nugroho, 2017).

(Jogiyanto, 2018) mendefinisikan sistem sebagai gabungan dari berbagai elemen yang berhubungan dan berinteraksi untuk mencapai tujuan tertentu. system ini menggambarkan kejadian-kejadian dan kesatuan objek nyata, misalnya tempat, benda, dan orang-orang yang benar-benar ada dan nyata. Sehingga secara umum sistem diartikan sebagai suatu kumpulan objek atau unsur-unsur atau bagian-bagian yang memiliki arti berbeda-beda yang saling memiliki hubungan, saling berkerjasama dan saling mempengaruhi satu sama lain serta memiliki keterikatan pada rencana yang sama dalam mencapai suatu tujuan tertentu pada lingkungan yang kompleks.

* 1. **Pengertian Managemen**

Managemen disebut sebagai seni untuk merealisasikan pekerjaan melalui orang lain dalam kaitannya dengan organisasi bahwa manager dalam mencapai tujuan organisasi dengan cara pengaturan orang lain (pegawai). Manajemen mencakup fungsi perencanaan, pengorganisasian, penyusunan personalia, pengarahan dan pengawasan (I Heryanto, 2016).

Menurut George R. Terry (Ahmad, 2018) manajemen merupakan proses yang khas yang terdiri dari tindakan-tindakan perencanaan, pengorganisasian, penggerakkan, dan pengawasan untuk mencapai tujuan yang telah ditetapkan memalui pemanfaatan sumber daya manusia dan sumber daya lainnya Definisi lain Stoner AF (Ahmad, 2018) menyatakan bahwa manajemen merupakan proses perencanaan, pengorganisasian dan pengawasan antar anggota organisasi dengan menggunakan seluruh sumber.

* 1. **Pengertian Sistem Managemen**

Definisi Sistem Managemen menurut (Ahmad, 2018) sistem managemen adalah jaringan prosedur pengolahan data yang dikembangkan dalam organisasi dan disatukan apabila di pandang perlu, dengan maksud memberikan data kepada manajemen setiap waktu diperlukan, baik data yang bersifat intern maupun yang bersifat ekstern, untuk dasar pengambilan keputusan dalam rangka mencapai tujuan organisasi.

* 1. **Pengertian Kost**

Indekos atau kos/kost adalah sebuah jasa yang menawarkan sebuah kamar atau tempat untuk ditinggali dengan sejumlah pembayaran tertentu untuk setiap periode tertentu (umumnya pembayaran per bulan). Kata ini diserap dari frasa bahasa Belanda "in de kost". Definisi "in de kost" sebenarnya adalah "makan di dalam", tetapi dapat pula berarti "tinggal dan ikut makan" di dalam rumah tempat menumpang tinggal. wikipedia.org. (2021, 06 Desember)

* 1. ***Framework Laravel***

1. Pengertian *Framework*

Menurut (Supono, 2016), *Framework* adalah sekumpulan dari fungsi-fungsi atau prosedur dan *class-class* untuk bertujuan tertentu yang sudah siap digunakan sehingga bisa lebih mempermudah dan mempercepat pekerjaan seorang *programmer*, tanpa harus membuat fungsi atau *class* dari awal.

Menurut (Rahmawati, 2017), Framework merupakan struktur konseptual dasar yang digunakan untuk memecahkan sebuah permasalahan atau isu kompleks.

1. Isi dari framework berupa fungsi, plugin dan konsep untuk membentuk suatu sistem tertentu.
2. Dengan menggunakan framework, aplikasi dapat tersusun dan terstruktur dengan rapi.
3. Dalam frameworktersedia struktur aplikasi yang baik, standard coding, best practice, design patterndan common function.

Maksud dari standard coding adalah sebuah standar yang harus diikuti programmer untuk menulis kode. Kemudian untuk best practice yaitu kumpulan actionyang telah teruji oleh para expert. Sedangkan maksud dari design patternadalah teknik - teknik yang menjadi best practice. Lalu untuk common function berarti fungsi - fungsi atau library yang telah umum digunakan dalam pengembangan sistem.

1. Pengertian *Laravel*

Laravel adalah framework bahasa pemrograman Hypertext Preprocessor (PHP) yang diciptakan oleh Taylor Otwell yang dirilis pada 9 Juni 2011. Laravel adalah sebuah framework PHP yang dirilis di bawah lisensi perusahaan MIT. Dibangun dengan konsep MVC (Model View Controller). Laravel adalah pengembanangan website berbasis MVP yang ditulis dengan bahasa pemrograman PHP yang dirancang untuk meningkatkan kualitas perangkat lunak dan berfungsi untuk meningkatkan pengalaman bekerja dengan aplikasi yang menyediakan sintaks yang ekspresif, jelas dan menghemat waktu (Hermanto, 2019).

* 1. **Pengertian *PHP***

Menurut Supono (2016), *PHP* ( *PHP* : *Hypertext preprocessor)* adalah suatu bahasa pemrograman yang digunakan untuk menterjemahkan baris kode program menjadi kode mesin yang dapat dimengerti oleh komputer yang bersifat *server-side* yang dapat ditambah ke dalam *HTML.*

Menurut (Tim, 2016), PHP memiliki beberapa kelebihan jika dibandingkan dengan bahasa pemrograman weblainnya, antara lain yaitu :

1. Bahasa pemrograman PHP adalah sebuah bahasa script yang tidak melakukan sebuah kompilasi dalam penggunaannya.
2. Banyak web server yang mendukung PHP. Mulai dari Apache, IIS, Lighttpd, hingga Xitami dengan konfigurasi yang relatif mudah. Ada juga paket web serveryang memudahkan melakukan instalasi sekali klik, seperti XAMPP yang tersedia untuk berbagai sistem operasi.
3. PHP lebih mudah dalam sisi pengembangan karena banyaknya milis, group, facebook, dan developeryang siap membantu dalam pengembangan.
4. Dalam sisi pemahaman, PHP adalah bahasa scriptingyang paling mudah dipahami karena memiliki referensi yang banyak.
5. PHP adalah bahasa pemrograman open source yang dapat digunakan diberbagai mesin (Linux, Unix, Macintosh, Windows) dan dapat dijalankan secara runtimemelalui console, serta dapat menjalankan perintah - perintah sistem.

Pada awalnya PHP merupakan kependekan dari *Personal Home Page*. PHP pertama kali dibua oleh Rasmus Lerdorf pada tahun 1995. Pada waktu itu PHP masih bernama *Form Interpreted* (FI), yang wujudnya masih berupa sekumpulan skrip yang digunakan untuk mengolah data formulir dari web. Selanjutnya Rasmus merilis kode sumber untuk umum dan menamakannya PHP/FI. Dengan perilisan kode sumber ini menjadi sumber terbuka, maka banyak pemrogram yang tertarik untuk ikut mengembangkan PHP. PHP memiliki 8 (delapan) tipe data, yaitu : (Abdulloh, 2018)

1. *Integer*

Tipe ini meliputi semua bilangan bulat dengan range -2,147,483,648 sampai +2,147,483,647 pada *platform* 32bit. PHP juga akan mengkonversi secara otomatis bila suatu bilangan berada diluar *range* tersebut ke dalam tipe data *floating point*. Tipe ini juga dapat dinyatakan dalam bentuk oktal (berbasis 8), desimal (berbasis 10), heksadesimal (berbasis 16).

1. *Floating Point*

Tipe ini biasa digunakan dalam bilangan pecahan namun bisa juga bilangan desimal. Tipe ini memiliki range 1.7E-308 sampai 1.7E+308. Dapat dinyatakan dalam bentuk bilangan desimal atau dalam bentuk pangkat.

1. *String*

Tipe data string dinyatakan dengan mengapitnya menggunakan tanda petik tunggal (‘ ‘) atau tanda petik ganda (” “). Perbedaan dari penggunaan keduanya adalah dengan tanda petik tunggal kita tidak dapat menggunakan variable dan *escape sequence handling* bersama dalam suatu kalimat.

1. Boolean

Tipe data boolean digunakan untuk menyimpan nilai *true* atau *false.* Biasanya tipe data ini mayoritas digunakan untuk melakukan pengecekan kondisi pada php.

1. Null

Tipe data yang tidak memuat apapun. Setiap Variabel yang diset menjadi Tipe Data NULL ini akan menjadikan Variabel tersebut kosong.

1. *Array*

Tipe ini dapat mengandung satu atau lebih data juga dapat diindeks berdasarkan numerik atau string. Ia juga mendukung multiarray dimensi dan membolehkan semua datanya berbeda tipe data.

1. Object

Tipe data object dapat berupa bilangan, variable atau fungsi. Object dibuat dengan tujuan agar para programmer terbiasa dengan OOP, meski fasilitas ini masih minim.

1. *Resource*

Tipe Data Spesial yang satu ini di khususkan untuk menyimpan resource, sumber atau alamat. Variabel tersebut hanya dapat diciptakan oleh suatu fungsi khusus yang mengembalikan nilai berupa resource seperti penggunaan fungsi fopen, opendir, mysql\_connect, mysql\_query dan semacamnya.

Beberapa kelebihan PHP dari bahasa pemrograman *web*, antara lain :

1. Bahasa pemrograman PHP adalah sebuah bahasa *script* yang tidak melakukan sebuah kompilasi dalam penggunaannya.
2. *Web* Server yang mendukung PHP dapat ditemukan dimana-mana, mulai dari apache, IIS, Lighttpd, hingga Xitami dengan konfigurasi yang relatif mudah.
3. Dalam sisi pengembangan lebih mudah, karena banyaknya milis-milis dan developer yang siap membantu dalam pengembangan.
4. Dalam sisi pemahaman, PHP adalah bahasa *scripting* yang paling mudah karena memiliki referensi yang banyak.
5. PHP adalah bahasa *open source* yang dapat digunakan di berbagai mesin (Linux, Unix, Macintosh, Windows) dan dapat dijalankan secara *runtime* melalui *console* serta juga dapat menjalankan perintah-perintah sistem.
   1. ***MySQL***

MySQL adalah sebuah implementasi dari sistem manajemen basis data relasional (RDBMS) yang didistribusikan secara gratis dibawah lisensi GPL (*General Public License*). Setiap pengguna dapat secara bebas menggunakan MySQL, namun dengan batasan perangkat lunak tersebut tidak boleh dijadikan produk turunan yang bersifat komersil. MySQL sebenarnya merupakan turunan salah satu konsep utama dalam basis data yang telah ada sebelumnya SQL (*Structured Query Language*). SQL adalah sebuah konsep pengoperasian basis data, terutama untuk pemilihan atau seleksi dan pemasukan data, yang memungkinkan pengoperasian data dikerjakan dengan mudah secara otomatis. MySQL memiliki beberapa keistimewaan, antara lain: (Nugroho, 2019)

1. *Portabilitas.*

MySQL dapat berjalan stabil pada berbagai sistem operasi seperti Windows, Linux, FreeBSD, Max Os X Server, Solaris, Amiga, dan masih banyak lagi.

1. Perangkat lunak sumber terbuka.

MySQL didistribusikan sebagai perangkat lunak sumber terbuka, dibawah lisensi GPL sehingga dapat digunakan secara gratis.

1. *Multi-user*.

MySQL dapat digunakan oleh beberapa pengguna dalam waktu yang bersamaan tanpa mengalami masalah.

1. *Performance-tuning*.

MySQL memiliki kecepatan yang menakjubkan dalam menangani *query* sederhana, dengan kata lain dapat memproses lebih banyak SQL persatuan waktu.

1. Ragam tipe data.

MySQL memiliki ragam tipe data yang sangat kaya, seperti signed/unsigned integer, float, double, char, text, date, timestamp, dan lain-lain.

1. Perintah dan fungsi.

MySQL memiliki operator dan fungsi secara penuh yang mendukung perintah *select* dan *where* dalam perintah (*query*).

1. Keamanan.

MySQL memiliki beberapa lapisan keamanan seperti level *subnetmask*, nama *host*, dan izin akses *user* dengan sistem perizinan yang mendetail serta sandi terenskripsi.

1. Skalabilitas dan pembatasan.

MySQL mampu menangani basis data secara besar, dengan jumlah rekaman (*records*) lebih dari 50 juta dan 60 ribu tabel serta 5 milyar baris. Selain itu batas indeks yang dapat ditampung mencapai 32 indeks pada tiap tabelnya.

1. Konektivitas.

MySQL dapat melakukan koneksi dengan klien menggunakan protokol TCP/IP, UNIX soket (UNIX), atau Named Pipes (NT).

1. Lokalisasi.

MySQL dapat mendeteksi pesan kesalahan pada klien dengan menggunakan lebih dari dua puluh bahasa. Meskipun demikian, Bahasa Indonesia belum termasuk di dalamnya.

1. Antar muka.

MySQL memiliki antar muka (*interface*) terhadap berbagai aplikasi dan bahasa pemrograman dengan menggunakan fungsi API (*Application Programming Interface*).

1. Klien dan peralatan

MySQL dilengkapi dengan berbagai peralatan (*tool*) yang dapat digunakan untuk administrasi basis data, dan pada setiap peralatan yang ada disertakan petunjuk *online*.

1. Struktur tabel

MySQL memiliki struktur tabel yang lebih fleksibel dalam menangani ALTER TABLE, dibandingkan dengan basis data lainnya semacam PostgreSQL ataupun Oracle.

* 1. ***XAMPP***

Menurut (Riyanto, 2014), *XAMPP* adalah sebuah *software* yang berfungsi untuk menjalankan *website* berbasis *PHP* dan menggunakan pengolah data *MySQL* di komputer secara local. *XAMPP* berperan sebagai *website* *server* pada komputer. *XAMPP* juga dapat disebut sebuah CPanel *server* *virtual*, yang dapat membantu anda melakukan preview sehingga dapat memodifikasi *website* tanpa harus online atau ter akses dengan internet.

*Software* *XAMPP* bersifat *open sources* yang dapat diperoleh secara gratis dari situs [www.apachefriends.org](http://www.apachefriends.org). *XAMPP* adalah perangkat lunak yang mendukung banyak sistem operasi dan merupakan komplikasi dari beberapa program. Fungsinya adalah sebagai *server* yang berdiri sendiri dan terdiri atas *Apache, MySQL*, dan bahasa pemrograman *PHP*.

Menurut (Purbadian, 2016), berpendapat bahwa XAMPP merupakan suatu software yang bersifat open source yang merupakan pengembangan dari LAMP (Linux, Apache, MySQL, PHP dan Perl).

Berdasarkan pengertian diatas dapat disimpulkan bahwa Xampp merupakan tool pembantu pengembangan paket perangkat lunak berbasis open source yang menggabungkan Apache web server, MySQL, PHP dan beberapa modul lainnya didalam satu paket aplikasi.

* 1. ***Unified Modeling Language* (*UML*)**

Pada perkembangan teknik pemograman berorientasi objek, muncullah sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang dibangun dengan menggunakan teknik pemograman berorientasi objek yaitu *Unified Modelling Language*(UML). UML muncul karena adanya kebutuhan pemodelan visual untuk menspesifikasikan, menggambarkan, membangun dan dokumentasi dari sistem perangkat lunak. UML merupakan bahasa visual untuk pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram dan teks-teks pendukung. (Muslihudin, 2016)

UML hanya berfungsi sebagai untuk melakukan pemodelan. Jadi penggunaan UML tidak terbatas pada metodologi tertentu, meskipun pada kenyataannya UML paling banyak digunakan pada metodologi berorientasi objek.

Beberapa jenis *UML* yang digunakan untuk merancang sistem diantaranya, yaitu :

1. ***Use case Diagram***

*Use case diagram* menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah “apa” yang diperbuat sistem, dan bukan “bagaimana”. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya login ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. (Muslihudin, 2016)

*Use case diagram* dapat sangat membantu bila sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem. *Sebuah* *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal. Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*. Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain. Berikut adalah simbol - simbol *Use Case Diagram* pada Tabel 3.1.

Tabel 3.1 Simbol *Use case Diagram*

(Muslihudin, 2016)

|  |  |  |
| --- | --- | --- |
| **SIMBOL** | **NAMA** | **KETERANGAN** |
|  | *Actor* | Menspesifikasikan himpuan peran yang pengguna mainkan ketika berinteraksi dengan *use case*. |
|  | *Dependency* | Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri *(independent)* akan mempengaruhi elemen yang bergantung padanya elemen yang tidak mandiri (*independent*). |
|  | *Generalization* | Hubungan dimana objek anak (*descendent*) berbagi perilaku dan struktur data dari objek yang ada di atasnya objek induk (*ancestor*). |
|  | *Include* | Menspesifikasikan bahwa *use case* sumber secara *eksplisit*. |
|  | *Extend* | Menspesifikasikan bahwa *use case* target memperluas perilaku dari *use case* sumber pada suatu titik yang diberikan. |
|  | *Association* | Apa yang menghubungkan antara objek satu dengan objek lainnya. |
|  | *System* | Menspesifikasikan paket yang menampilkan sistem secara terbatas. |
|  | *Use Case* | Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu aktor |

Lanjutan Tabel 3.1. Simbol *Use Case Diagram*

(Muslihudin, 2016)

|  |  |  |
| --- | --- | --- |
| **SIMBOL** | **NAMA** | **KETERANGAN** |
|  | *Collaboration* | Interaksi aturan-aturan dan elemen lain yang bekerja sama untuk menyediakan prilaku yang lebih besar dari jumlah dan elemen-elemennya (sinergi). |
|  | *Note* | Elemen fisik yang eksis saat aplikasi dijalankan dan mencerminkan suatu sumber daya komputasi |

1. ***Activity Diagram***

*Activity diagram* menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi. (Muslihudin, 2016)

*Activity diagram* merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan behaviour internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas. Sama seperti *state*, standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan behaviour pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal. Berikut adalah simbol - simbol *Activity Diagram* pada Tabel 3.2.

Tabel 3.2 Simbol *Activity Diagram*

(Muslihudin, 2016)

|  |  |  |
| --- | --- | --- |
| **SIMBOL** | **NAMA** | **KETERANGAN** |
|  | State | Kondisi yang mungkin dialami oleh suatu obyek. |
|  | Note | Note digunakan untuk memberikan keterangan ataukomentar |
|  | Aktivitas | Perilaku obyek yang dilakukan saat obyek berada dalam *state* tertentu. |
|  | Start State | *Start state* digunakan untuk memulai diagram *statechart.* |
|  | End State | End start digunakan untuk mengakhiri diagram. |
|  | Decision | *Decision* digunakan sebagai pilihan untuk pengambilan keputusan. |
|  | Penggabungan / Join | Digunakan untuk split dan join.pada saat diagram akanmembagi 2, bar ini akanditambahkan. dan sebelum diagram digabung menjadi satu, sebagai join. |
|  | Asosiasi *(association)* | Apa yang menghubungkan antara objek satu dengan objek yang lainnya. |

1. ***Class Diagram***

*Class* adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi). *Class diagram* menggambarkan struktur dan deskripsi *class, package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. *Class* memiliki tiga area pokok : (Muslihudin, 2016)

* 1. Nama (dan stereotype)
  2. Atribut
  3. Metoda

*Class* dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class.* Dengan demikian *interface* mendukung resolusi metoda pada saat *run-time*. Hubunganantar *class* yaitu:

* 1. Asosiasi

Hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain. Panah *navigability* menunjukkan arah *query* antar class.

* 1. Agregasi

Hubungan yang menyatakan bagian (“terdiri atas..”).

* 1. Pewarisan

Hubungan hirarkis antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metoda *class* asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.

* 1. Hubungan Dinamis

Rangkaian pesan (*message*) yang di-*passing* dari satu *class* kepada *class* lain.

Tabel 3.3 Simbol *Class Diagram*

(Muslihudin, 2016)

|  |  |  |
| --- | --- | --- |
| **SIMBOL** | **NAMA** | **KETERANGAN** |
|  | *Generalization* | Hubungan dimana objek anak *(descendent)* berbagi perilaku dan struktur data dari objek yang ada di atasnya objek induk (*ancestor*). |
|  | *Nary Association* | Upaya untuk menghindari asosiasi dengan > 2 objek. |
|  | *Class* | Himpunan dari objek-objek yang berbagi atribut serta operasi yang sama. |
|  | *Collaboration* | Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu aktor |
|  | *Realization* | Operasi yang benar-benar dilakukan oleh suatu objek. |
|  | *Dependency* | Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri *(independent)* akan mempegaruhi elemen yang bergantung padanya elemen yang tidak mandiri |
|  | *Association* | Apa yang menghubungkan antara objek satu dengan objek lainnya |

1. ***Sequence Diagram***

*Sequence diagram* menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). *Sequence diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan. (Muslihudin, 2016)

Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal. *Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, *message* akan dipetakan menjadi operasi/metoda dari *class*. *Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah *message*. Berikut adalah simbol - simbol *Sequence Diagram* pada Tabel 4.

Tabel 3.4 Simbol *Sequence Diagram*

(Muslihudin, 2016)

|  |  |  |
| --- | --- | --- |
| **SIMBOL** | **NAMA** | **KETERANGAN** |
|  | *LifeLine* | Menggambarkan objek *entity*, antarmuka yang saling berinteraksi. |
|  | *Message* | Menggambarkan pengiriman pesan |
|  | *Actor* | Menggambarkan orang yang sedang berinteraksi dengan sistem |

Lanjutan Tabel 3.4 Simbol *Sequence Diagram*

(Muslihudin, 2016)

|  |  |  |
| --- | --- | --- |
| **SIMBOL** | **NAMA** | **KETERANGAN** |
|  | *Entity Class* | Menggambarkan hubungan kegiatan yang akan dilakukan |
|  | *Boundary Class* | Menggambarkan sebuah penggambaran dari form |
|  | *Control Class* | Menggambarkan penghubung antara *boundary* dengan tabel |

* 1. **Pengujian Sistem**

Teknik pengujian adalah cara atau teknik untuk menguji perangkat lunak, mempunyai mekanisme untuk menentukan data uji yang dapat menguji perangkat lunak secara lengkap dan mempunyai kemungkinan tinggi untuk menentukan kesalahan. Pengujian perangkat lunak adalah elemen kritis dari jaminan kualitas perangkat lunak dan mempresentasikan kajian pokok dari spesifikasi, desain dan pengkodean. (Pressman, 2015)

1. **Dasar-Dasar Pengujian Perangkat Lunak**

Pengembangan perangkat lunak sesuai dengan sifat dasar mereka adalah manusia pembangun. Pengujian mengharuskan pengembang membuang pemikiran yang sebelumnya mengenai “kebenaran” perangkat lunak yang baru saja dikembangkan dan mengatasi konflik minat yang terjadi pada saat kesalahan ditemukan. (Pressman, 2015)

1. Sasaran Pengujian
2. Pengujian adalah proses eksekusi suatu program dengan maksud menemukan kesalahan.
3. *Test case* yang baik adalah *test case* yang memiliki probabilitas tinggi untuk menemukan kesalahan yang belum pernah ditemukan sebelumnya.
4. Pengujian yang sukses adalah pengujian yang mengungkapkan semua kesalahan yang belum pernah ditemukan sebelumnya.
5. Prinsip Pengujian
6. Semua pengujian harus dapat ditelusuri sampai ke gejala terakhir.
7. Pengujian harus direncanakan lama sebelum pengujian itu dimulai.
8. Prinsip Pareto berlaku untuk pengujian perangkat lunak.
9. Pengujian harus mulai "dari yang kecil" dan berkembang ke pengujian "yang besar".
10. Pengujian yang mendalam tidak mungkin.
11. Paling efektif, pengujian dilakukan oleh pihak ketiga yang independen.
12. Karakteristik yang membawa perangkat lunak dapat diuji:
13. Operabilitas “semakin baik dia bekerja semakin efisien dian dapat diuji”.
14. Observabilitas “apa yang dilihat adalah apa yang diuji”.
15. Kontrolabilitas “semakin baik kita dapat mengontrol perangkat lunak semakin banyak pengujian yang dapat diotomatisasi dan dioptimalkan”.
16. Dekomposabilitas “dengan mengontrol ruang lingkup pengujian kita dapat lebih cepat mengisolasi masalah dan melakukan pengujian kembali”.
17. Kesederhanaan “semakin sedikit yang diuji semakin cepat pengujian”.
18. Stabilitas “semakin sedikit perubahan semakin sedikit gangguan pengujian”.
19. Kemampuan yang dapat dipahami “semakin banyak informasi yang dimiliki semakin detail pengujiannya”.
20. Atribut-atribut pengujian yang baik :
21. Memiliki probabilitas yang tinggi menemukan kesalahan.
22. Tidak redundan.
23. Harusnya “jenis terbaik”.
24. Tidak boleh terlalu sederhana atau terlalu kompleks.
25. **Pengujian *Black-Box***

*Black-box testing* adalah pengujian yang dilakukan hanya mengamati hasil eksekusi melalui data uji dan memeriksa fungsional dari perangkat lunak. Jadi dianalogikan seperti melihat suatu kotak hitam, hanya bisa melihat penampilan luarnya saja tanpa mengetahui apa sesungguhnya yang terjadi dalam proses detilnya (hanya mengetahui *input* dan *output*). Pengujian *black-box* berfokus pada persyaratan fungsional perangkat lunak. Pengujian ini memungkinkan analis sistem memperoleh kumpulan kondisi *input* yang akan mengerjakan seluruh keperluan fungsional program. Tujuan metode ini mencari kesalahan pada: (Pressman, 2015)

1. Fungsi yang salah atau hilang.
2. Kesalahan pada *interface*.
3. Kesalahan pada struktur data atau akses *database*.
4. Kesalahan performansi.
5. Kesalahan inisialisasi dan tujuan akhir.

Ujicoba *black-box* diaplikasikan dibeberapa tahapan berikutnya. Karena ujicoba *black-box* dengan sengaja mengabaikan struktur kontrol, sehingga perhatiannya difokuskan pada informasi domain. Ujicoba didesain untuk dapat menjawab pertanyaan- pertanyaan berikut: (Pressman, 2015)

Bagaimana validitas fungsionalnya diuji?

Jenis input seperti apa yang akan menghasilkan kasus uji yang baik ?

Apakah sistem secara khusus sensitif terhadap nilai input tertentu ?

Bagaimana batasan-batasan kelas data diisolasi?

Berapa rasio data dan jumlah data yang dapat ditoleransi oleh sistem?

Apa akibat yang akan timbul dari kombinasi spesifik data pada operasi sistem?

Dengan mengaplikasikan ujicoba *black-box*, diharapkan dapat menghasilkan sekumpulan kasus uji yang memenuhi kriteria berikut : (Pressman, 2015)

1. Kasus uji yang berkurang, jika jumlahnya lebih dari 1, maka jumlah dari uji kasus tambahan harus didesain untuk mencapai ujicoba yang cukup beralasan.
2. Kasus uji yang memberitahukan sesuatu tentang keberadaan atau tidaknya suatu jenis kesalahan, daripada kesalahan yang terhubung hanya dengan suatu ujicoba yang spesifik.
3. **Pengujian *White Box***

Pengujian *white box* biasa disebut dengan *glass box* adalah metode desain *test case* yang menggunakan struktur kontrol desain prosedural untuk memperoleh sebuah *test case* (Pressman, 2015). Tujuan dari penggunaan pengujian *white box* yaitu menguji semua *statement* pada program. Metode pengujian *white box* dapat menjamin:

1. Semua jalur (*path*) yang independen / terpisah dapat di tes setidaknya satu kali tes.
2. Semua logika keputusan dapat dites dengan jalur yang salah dan atau jalur yang benar.
3. Semua *loop* dapat dites terhadap batasannya dan ikatan operasionalnya.
4. Semua struktur internal data dapat dites untuk memastikan validitasnya.

Pada dasarnya sekarang ini,banyak para penguji perangkat lunak jarang melakukan pengujian dengan menggunakan *white box testing*, tetapi langsung melakukan pengujian pada tampilan antarmuka sistem. Namun demikian pengujian tidak bisa hanya dilakukan untuk menguji pada tampilan sistem, tetapi juga perlu dilakukan pengujian pada struktur dan kontrol logika pada kode program. Pengujian dengan metode *white box testing* dilakukan pada suatu perangkat lunak dengan tujuan untuk menemukan kesalahan-kesalahan yang mungkin tidak ditemukan ketika melakukan pengujian pada tampilan antarmuka sistem atau yang biasa disebut dengan metode *black box testing*. Pengujian *white box* diantaranya:

1. Notasi Diagram Alir

Notasi yang digunakan untuk menggambarkan jalur eksekusi adalah notasi diagram alir atau grafik program, yang menggunakan notasi lingkaran (simpul atau node) dan anak panah (*link* atau *edge*). Notasi ini menggambarkan aliran control logika yang digunakan dalam suatu bahasa pemrograman. Setiap representasi rancangan procedural dapat diterjemahkan kedalam *flowgraph*.

1. Kompleksitas Siklomatis

Kompleksitas siklomatis adalah metode pengukuran perangkat lunak yang memberikan pengukuran kuantitatif terhadap kompleksitas logika sebuah program. Dalam mengukur sebuah kompleksitasi pada sebuah program maka dilakukan perhitungan *cyclomatic complexity* (cc), nilai yang didapat berdasarkan perhitungan tersebut untuk menentukan jumlah jalur *independent* dalam himpunan *path*, serta akan memberi nilai batas atas bagi jumlah pengujian yang harus dilakukan, untuk memastikan bahwa semua pernyataan telah dieksekusi setidaknya satu kali. jalur independen adalah jalur yang terdapat dalam program yang mengintroduksi sedikitnya satu rangkaian pernyataan proses atau kondisi baru. Kompleksitas siklomatis dapat dicari dengan salah satu dari 3 cara sebagai berikut:

1. Jumlah region dari grafik alur mengacu kepada kompleksitas siklomatis.
2. Kompleksitas siklomatis dilambangkan dengan V(G). Rumus penghitungan kompleksitas siklomatis adalah:

V(G) = E−N+2 dimana

E = jumlah *edge* pada flowgraph

N = jumlah *node* pada flowgraph

1. Kompleksitas siklomatis dilambangkan dengan V(G). Rumus penghitungan kompleksitas siklomatis yang lain adalah:

V(G) = P + 1 dimana

P = jumlah *predicates nodes* pada flowgraph