

## **BAB II**

### **TINJAUAN PUSTAKA**

#### **2.1 Teori Terkait**

Pada penelitian yang dilakukan oleh Friendly, Harizahayu, dan rekan-rekan (2022) berfokus pada perancangan dan pembuatan alat pemberi pakan otomatis berbasis *IoT* pada UMKM peternakan bebek di Kecamatan Lubuk Pakam, Kabupaten Deli Serdang. Sistem yang dikembangkan bertujuan untuk mengatasi permasalahan keterlambatan pemberian pakan akibat kurangnya tenaga kerja serta ketidakteraturan jadwal pemberian pakan. Penelitian ini menggunakan mikrokontroler *ESP8266* yang mengontrol mekanisme pemberian pakan secara otomatis berdasarkan waktu yang telah ditentukan. Hasil penelitian menunjukkan bahwa sistem ini mampu meningkatkan efisiensi dalam pemberian pakan, sehingga mengurangi jumlah pakan yang terbuang dan meningkatkan pertumbuhan bebek[7].

Pada Penelitian yang dilakukan oleh Fatra Nonggala Putra dan rekan-rekan (2023) melakukan penelitian terkait smart grading berbasis *IoT* untuk peternakan bebek pedaging. Penelitian ini bertujuan untuk mengatasi permasalahan dalam proses grading atau seleksi bebek berdasarkan bobotnya yang sebelumnya dilakukan secara manual. Dengan menggunakan timbangan *IoT*, data bobot bebek secara otomatis masuk ke dalam sistem berbasis *web* yang dapat diakses melalui perangkat seluler. Hasil penelitian menunjukkan bahwa penerapan teknologi ini dapat mengurangi kesalahan pencatatan, serta

membantu peternak dalam mengelola pakan dan biaya operasional dengan lebih efektif[8].

Penelitian yang dilakukan oleh Saparin dan rekan-rekan (2024) membahas tentang modernisasi pengolahan pakan bebek dengan mesin pengaduk di peternakan Taret Jaya, Desa Air Anyir, Kabupaten Bangka. Sebelumnya, pencampuran bahan pakan masih dilakukan secara manual menggunakan cangkul, yang membutuhkan waktu lama dan tenaga yang besar. Dalam penelitian ini, dikembangkan mesin pengaduk pakan otomatis yang dapat mempercepat proses pencampuran hingga 6 kali lebih cepat dibandingkan dengan metode manual. Dengan penerapan mesin ini, peternak dapat menghemat waktu dan meningkatkan efisiensi produksi pakan[9].

Dalam penelitian yang dipublikasikan di Tamkin: Jurnal Pengembangan Masyarakat Islam menunjukkan bahwa integrasi teknologi *monitoring* dengan pendekatan manajemen kelompok tani dapat meningkatkan kesejahteraan peternak. Hal ini memberikan konteks tambahan bagi pentingnya implementasi sistem monitoring dalam upaya peningkatan produktivitas dan kesejahteraan anggota KTTI Berkah Abadi Tegal.

## 2.2 Landasan Teori

### 2.2.1 Bebek

Bebek merupakan unggas air yang memiliki adaptasi *morfologis* khusus, seperti paruh yang lebar, bulu tahan air, dan kaki berselaput. Adaptasi ini mendukung kemampuannya dalam mencari makan di

lingkungan perairan. Secara *fisiologis*, bebek memiliki *metabolisme* yang tinggi dan sistem pencernaan yang efisien untuk mencerna pakan yang beragam. Kondisi *internal* seperti suhu tubuh dan kelembaban juga sangat berpengaruh terhadap kenyamanan dan kesehatan bebek. Misalnya, pada tahap *brooding*, bebek muda memerlukan suhu lingkungan yang stabil (antara 31°C hingga 33°C) untuk menciptakan zona nyaman yang mendukung pertumbuhan optimal.

Pengelolaan kandang bebek mencakup beberapa aspek, antara lain pengaturan suhu, kelembaban, sanitasi, dan pemberian pakan secara teratur. Kondisi lingkungan yang optimal sangat penting, terutama pada masa DOD, karena anak bebek masih sangat rentan terhadap perubahan kondisi lingkungan. Oleh karena itu, penerapan sistem *monitoring* suhu yang akurat dan *real-time* sangat diperlukan untuk menjaga kestabilan lingkungan kandang. Sistem ini tidak hanya mendukung pertumbuhan optimal DOD, tetapi juga membantu mengurangi angka kematian dan meningkatkan produktivitas peternakan. Kesehatan bebek sangat erat kaitannya dengan kualitas lingkungan kandang. Lingkungan yang tidak terjaga dengan baik, seperti adanya kelebihan gas amonia, kelembaban yang tidak sesuai, atau suhu yang tidak stabil, dapat menyebabkan stres dan menurunkan daya tahan tubuh bebek. Oleh karena itu, pengelolaan ventilasi, sanitasi, dan pengaturan suhu di kandang merupakan aspek penting dalam menjaga kesehatan bebek. Sistem *monitoring* yang terintegrasi

dengan sensor-sensor lingkungan membantu peternak untuk mendeteksi perubahan kondisi secara *real-time*, sehingga tindakan preventif dapat segera diambil[10].

Bebek sangat sensitif terhadap perubahan lingkungan. Fluktuasi suhu yang drastis, kelembaban yang rendah atau tinggi, serta adanya kontaminan di udara atau air, dapat mengganggu sistem *fisiologis* dan perilaku bebek. Lingkungan kandang yang optimal tidak hanya menunjang pertumbuhan, tetapi juga mencegah timbulnya penyakit dan stres pada bebek. Dengan demikian, pengaturan lingkungan melalui sistem *monitoring* otomatis menjadi kunci untuk mengoptimalkan kesehatan dan kinerja bebek dalam peternakan[11].

Peternakan bebek tidak hanya memberikan kontribusi dalam penyediaan pangan, tetapi juga berperan dalam peningkatan kesejahteraan ekonomi masyarakat. Kelompok tani, seperti KTTI, memanfaatkan potensi peternakan bebek untuk meningkatkan pendapatan dan memberdayakan anggotanya. Dengan penerapan teknologi modern, seperti sistem monitoring berbasis *IoT*, *efisiensi* operasional dapat ditingkatkan, sehingga memberikan kontribusi positif terhadap peningkatan produktivitas peternakan sekaligus mendukung kesejahteraan peternak secara menyeluruh.

### **2.2.2 Perkandangan Bebek**

Perkandangan bebek mencakup seluruh aspek desain, manajemen, dan pengaturan lingkungan di dalam kandang yang

bertujuan menciptakan kondisi optimal bagi pertumbuhan dan perkembangan bebek, terutama pada masa *Day Old Duck* (DOD). Lingkungan kandang yang baik harus mampu menjaga kestabilan suhu, kelembaban, sirkulasi udara, dan pencahayaan agar mendukung kesehatan bebek dan meminimalkan risiko stres serta penyakit. Kondisi ini sangat krusial pada tahap awal kehidupan bebek yang belum memiliki sistem termoregulasi yang sempurna.

Pada masa *brooding*, bebek DOD memerlukan suhu ideal berkisar antara 31°C hingga 33°C. Fluktuasi suhu yang tajam dapat menyebabkan stres, penurunan daya tahan tubuh, bahkan kematian. Selain itu, kelembaban yang optimal juga penting untuk menghindari kondisi terlalu kering atau lembap, yang keduanya dapat memicu pertumbuhan patogen dan menurunkan kesehatan bebek. Ventilasi yang memadai merupakan aspek penting dalam pengelolaan kandang. Sirkulasi udara yang baik membantu menghilangkan gas berbahaya seperti amonia yang dihasilkan dari kotoran bebek, serta menjaga kadar oksigen yang cukup. Dengan ventilasi yang tepat, suhu dan kelembaban kandang dapat lebih stabil, sehingga menciptakan lingkungan yang sehat bagi bebek. Pencahayaan yang memadai mendukung aktivitas bebek dan membantu pengawasan kondisi kandang. Kebersihan kandang yang terjaga juga sangat penting untuk mencegah penumpukan kotoran dan bakteri, yang jika dibiarkan dapat mempengaruhi kesehatan bebek. Aspek ini mendukung kinerja sistem

*monitoring* sebagai bagian dari pengelolaan kandang secara menyeluruh[12].

Kandang bebek yang dirancang dan dikelola dengan baik memiliki dampak positif terhadap kesehatan dan produktivitas. Kondisi lingkungan yang stabil pada kandang dapat menurunkan tingkat kematian DOD, meningkatkan pertumbuhan, dan menghasilkan bebek dengan produktivitas yang lebih tinggi, baik dari segi produksi telur maupun daging. Beberapa penelitian telah menunjukkan bahwa implementasi sistem *monitoring* otomatis dalam kandang dapat meningkatkan produktivitas hingga 5% lebih tinggi dibandingkan dengan metode konvensional[13].

Untuk mengatasi tantangan pengelolaan lingkungan kandang secara manual, teknologi *Internet of Things (IoT)* telah diterapkan dalam perkandangan bebek. Sistem *monitoring* berbasis *IoT* memungkinkan pengambilan data secara *real-time* dari berbagai sensor yang terintegrasi di dalam lingkungan kandang, seperti sensor suhu, kelembapan udara, serta sensor gas amonia. Data tersebut kemudian diintegrasikan ke dalam platform berbasis *website* yang dapat diakses oleh peternak untuk pemantauan dan pengendalian jarak jauh. Dengan penerapan teknologi ini, pengelolaan kandang dapat menjadi lebih efisien, mengurangi kesalahan manusia, dan meningkatkan respons terhadap perubahan kondisi lingkungan[14].

### 2.2.3 Visual Studio Code



Gambar 2.1 Visual Studio Code

*Visual Studio Code* merupakan *editor teks open-source* yang dikembangkan oleh *Microsoft* dan dapat diakses secara gratis. Aplikasi ini bersifat *cross-platform*, sehingga kompatibel dengan berbagai sistem operasi seperti *Windows*, *Linux*, dan *macOS*. Meski tergolong ringan, *Visual Studio Code* menawarkan beragam fitur canggih yang mendukung proses pengembangan perangkat lunak. Editor ini mendukung banyak bahasa pemrograman, seperti *Java*, *JavaScript*, *Go*, *C++*, dan lainnya. Fitur yang disediakan mencakup *syntax highlighting*, *auto-completion*, *code snippets*, *refactoring*, *debugging*, serta integrasi dengan *Git*. Sama halnya dengan *editor* lain seperti *Sublime* atau *Atom*, *Visual Studio Code* mampu mengenali berbagai bahasa pemrograman dan menyesuaikan *color scheme* secara otomatis berdasarkan struktur kode yang sedang ditulis, menjadikannya alat yang efisien bagi pengembang *website*, *software*, maupun aplikasi kompleks lainnya[15].

#### 2.2.4 Framework Laravel



Gambar 2.2 Logo Framework Laravel

*Framework* merupakan kumpulan komponen pemrograman yang dapat digunakan kembali (*reusable*), sehingga pengembang tidak perlu menulis ulang skrip yang sama untuk tugas serupa. Salah satu contohnya adalah *Laravel*, yaitu *framework web* berbasis *PHP* yang bersifat *open-source* dan gratis, dikembangkan oleh Taylor Otwell, dan dirancang untuk membangun aplikasi web dengan menerapkan pola arsitektur *Model-View-Controller (MVC)*[16].

#### 2.2.5 PHP (Hypertext Preprocessor)



Gambar 2.3 Logo PHP

Dalam menggunakan *framework laravel* PHP menjadi bahasa inti yang digunakan untuk membangun aplikasi *web* berbasis *Laravel*. PHP adalah pemrograman interpreter yaitu proses penerjemahan baris kode sumber menjadi kode mesin yang dimengerti komputer secara langsung pada saat baris kode dijalankan. PHP disebut sebagai

pemrograman *Server Side Programming*, hal ini dikarenakan seluruh prosesnya dijalankan pada server tidak dijalankan pada *client* [17].

#### 2.2.6 Laragon



Gambar 2.4 Laragon

Laragon merupakan perangkat lunak yang dibuat untuk mempermudah proses pembuatan serta pengelolaan *server* lokal. Laragon menawarkan solusi bagi *web developer* yang memerlukan lingkungan pengembangan yang efisien, stabil, dan fleksibel[18].

#### 2.2.7 PhpMyAdmin



Gambar 2.5 phpMyAdmin

*PhpMyAdmin* merupakan perangkat lunak bebas yang dikembangkan menggunakan bahasa pemrograman *PHP*, dan berfungsi untuk mengelola administrasi *MySQL* melalui platform *World Wide Web*. Aplikasi ini mendukung berbagai operasi seperti pengelolaan *database*, *tabel*, *field*, *relation*, *index*, *user*, serta pengaturan *permission*. Dengan antarmuka yang sederhana,

*phpMyAdmin* mempermudah pengguna dalam mengatur *database* secara cepat dan efisien[19].

### 2.2.8 MySQL



Gambar 2.6 MySQL

*MySQL* adalah sistem manajemen basis data *Relational Database Management System (RDBMS)* yang bersifat *open-source*. Awalnya dikembangkan oleh perusahaan asal Swedia sekitar tahun 1994–1995, namun fondasi pengembangannya telah dimulai sejak 1979. Saat ini, kepemilikan *MySQL* berada di bawah *Oracle Corporation* dan didistribusikan dengan lisensi *General Public License (GPL)*. Popularitas *MySQL* didorong oleh kecepatan, keandalan, serta kemudahannya untuk diintegrasikan dengan berbagai bahasa pemrograman seperti *PHP*, *Python*, dan *Java*, yang menjadikannya sangat mendukung dalam pengembangan aplikasi *web*[20].

### 2.2.9 HTML

*HyperText Markup Language (HTML)* adalah suatu bahasa yang digunakan untuk menulis halaman web dan menampilkan berbagai informasi didalam sebuah browser internet. HTML ditulis

dalam format dokumen ASCII atau teks biasa. Artinya, file yang dibuat menggunakan perangkat lunak pengolah kata dapat disimpan dalam format ASCII standar, yang dirancang agar tidak bergantung pada sistem informasi tertentu[21].

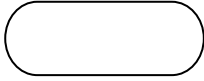
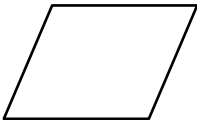
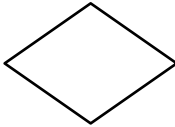
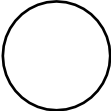
#### **2.2.10 CSS**

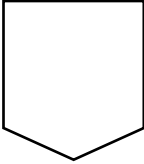
*Cascading Style Sheets (CSS)* adalah bahasa *style sheet* yang berfungsi untuk mengatur tampilan visual dokumen yang ditulis menggunakan bahasa *markup*. Umumnya, CSS digunakan untuk memformat halaman *web* berbasis *HTML* atau *XHTML*, namun juga dapat diterapkan pada dokumen *XML* lain seperti *SVG* dan *XUL*. Standarisasi dan pengembangan CSS dikelola oleh *World Wide Web Consortium (W3C)*[22].

#### **2.2.11 Flowchart**

*Flowchart* adalah sebuah diagram grafis yang menggambarkan alur langkah-langkah dalam suatu program atau proses secara terstruktur. Diagram ini menggunakan simbol-simbol standar untuk mewakili berbagai tahapan atau aktivitas, sehingga mempermudah pemahaman mengenai jalannya program atau proses yang dimaksud. Simbol-simbol yang digunakan dalam *flowchart* memiliki fungsi masing-masing sesuai dengan jenis langkah atau aktifitas yang digambarkan.

Table 2.1 Simbol Flowchart Diagram

| Simbol                                                                              | Keterangan                                                                                                                                                                                                                                                                                                                                                                          |
|-------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|    | <i>Terminator</i> atau <i>terminal</i> merupakan simbol yang digunakan untuk merepresentasikan kondisi awal dan akhir dalam sebuah <i>flowchart</i> . Simbol ini berbentuk lingkaran, dengan label seperti "Mulai" untuk menunjukkan titik awal, serta "Selesai" atau "Akhir" sebagai penanda bahwa proses telah berakhir.                                                          |
|    | <i>Preparation</i> adalah simbol dalam <i>flowchart</i> yang digunakan untuk mengidentifikasi elemen-elemen yang akan digunakan dalam program. Simbol ini biasanya berbentuk persegi atau kotak, dan di dalamnya dapat dicantumkan nama variabel beserta jenis datanya, seperti <i>string</i> , <i>numeric</i> , <i>Boolean</i> , maupun <i>date</i> .                              |
|  | Simbol " <i>Predefined Process</i> " atau " <i>Subroutine</i> " dalam <i>flowchart</i> digunakan untuk menunjukkan proses yang dijelaskan secara terpisah sebagai <i>subprogram</i> , yang dimulai dan diakhiri dengan <i>terminator</i> .                                                                                                                                          |
|  | Dalam <i>flowchat</i> , simbol ini digunakan untuk mewakili proses <i>input</i> dan <i>output</i> . <i>Input</i> mencakup aktivitas seperti menerima data dari pengguna, misalnya melalui keyboard atau perangkat lainnya.                                                                                                                                                          |
|  | Simbol keputusan dalam <i>flowchart</i> digunakan untuk menentukan pilihan suatu kondisi (Ya atau Tidak). Memiliki dua keluaran arus dengan keterangan "Ya" dan "Tidak" untuk membedakannya dari simbol lain.                                                                                                                                                                       |
|  | <i>Connector</i> merupakan simbol berbentuk lingkaran yang berfungsi menghubungkan antar simbol dalam satu halaman, sehingga alur program dapat ditampilkan secara jelas tanpa harus menggambarkan arus yang terlalu panjang. Untuk menyambungkan simbol dari halaman yang berbeda, digunakan pasangan <i>connector</i> dengan label identik agar kesinambungan alur tetap terjaga. |

| Simbol                                                                            | Keterangan                                                                                                                                                                                                                                                                                                    |
|-----------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <i>On-Page Connector</i> dalam <i>flowchart</i> , yang digunakan untuk menghubungkan bagian-bagian <i>flowchart</i> di halaman yang sama. Simbol ini berbentuk segi lima dan dilengkapi dengan identitas berupa huruf atau angka                                                                              |
|  | Simbol <i>arrow</i> atau panah digunakan dalam <i>flowchart</i> untuk menunjukkan arah aliran proses, menghubungkan satu simbol dengan simbol lainnya guna memperjelas urutan jalannya program.                                                                                                               |
|  | Dalam <i>flowchart</i> , simbol <i>process</i> umumnya digunakan untuk menyatakan aktivitas pemberian atau penampilan nilai suatu variabel. Simbol ini biasanya berbentuk panah dan merepresentasikan variabel yang belum mengalami manipulasi oleh operator, seperti proses pemberian atau penambahan nilai. |

### 2.2.12 UML (*Unified Modeling Language*)

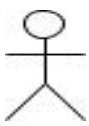
*Unified Modeling Language (UML)* merupakan suatu bahasa standar yang digunakan untuk merepresentasikan desain dari perangkat lunak. Melalui *UML*, pengembang dapat memvisualisasikan, menjabarkan, serta mendokumentasikan berbagai komponen dalam sistem perangkat lunak. Sebagaimana seorang arsitek membuat cetak biru bangunan, arsitek perangkat lunak memanfaatkan diagram *UML* guna mempermudah proses pengembangan. Terdapat berbagai jenis diagram *UML* yang sering diterapkan dalam rekayasa perangkat lunak, di antaranya: (1) *Use Case*, yang menunjukkan fungsi sistem serta interaksi antara pengguna atau sistem eksternal dengan sistem utama; (2) *Activity*

*Diagram*, yang menggambarkan rangkaian aktivitas atau proses dalam sistem; (3) *Sequence Diagram*, yang menampilkan urutan interaksi antar objek sesuai kronologi waktu; dan (4) *Class Diagram*, yang memvisualisasikan struktur statis sistem, mencakup kelas-kelas, atribut, metode, serta relasi antar kelas seperti *association*, *generalization*, dan *dependency*[23].

Beberapa jenis diagram *UML* kerap dimanfaatkan dalam proses pengembangan suatu sistem, antara lain:

1. *Use Case Diagram* menggambarkan fungsionalitas yang diharapkan dari suatu sistem serta interaksi antara *aktor* dengan sistem itu sendiri. Di dalam *use case*, *aktor* merepresentasikan entitas, baik manusia maupun sistem lain, yang terlibat dalam menjalankan fungsi dalam sistem.

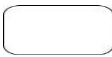
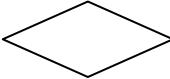
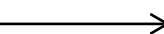
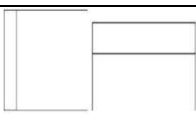
Table 2.2 Simbol Usecase Diagram

| No. | Gambar                                                                              | Nama                  | Keterangan                                                                         |
|-----|-------------------------------------------------------------------------------------|-----------------------|------------------------------------------------------------------------------------|
| 1   |  | <i>Actor</i>          | Individu, proses, atau sistem yang terlibat dalam interaksi dengan sistem lainnya. |
| 2   |  | <i>Dependency</i>     | Hubungan perubahan yang terjadi pada suatu elemen mandiri                          |
| 3   |  | <i>Generalization</i> | Hubungan generalisasi (umum-khusus) antar dua buah <i>use case</i>                 |

| No. | Gambar                                                                              | Nama                 | Keterangan                                                                                                                                                 |
|-----|-------------------------------------------------------------------------------------|----------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 4   |    | <i>Include</i>       | Relasi <i>use case</i> tambahan ke sebuah <i>use case</i> , dimana <i>use case</i> yang ditambahkan dapat berdiri sendiri.                                 |
| 5   |    | <i>Extend</i>        | Relasi tambahan yang Menunjukkan <i>use case</i> lain memerlukan <i>use case</i> . syarat untuk dijalankan atau agar fungsinya dapat berjalan dengan baik. |
| 6   |    | <i>Association</i>   | Komunikasi antar aktor dan <i>use case</i> yang berpartisipasi pada <i>use case</i> atau <i>use case</i> memiliki interaksi dengan aktor.                  |
| 7   |  | <i>System</i>        | Menspesifikasikan paket yang menampilkan sistem secara terbatas.                                                                                           |
| 8   |  | <i>Use Case</i>      | Deskripsi dari urutan aksi-aksi yang ditampilkan sistem                                                                                                    |
| 9   |  | <i>Collaboration</i> | Interaksi aturan-aturan dan elemen lain yang bekerja sama untuk menyediakan perilaku yang lebih besar dari jumlah dan elemen-elemennya (sinergi).          |
| 10  |  | <i>Note</i>          | Elemen fisik yang eksis saat aplikasi dijalankan dan mencerminkan suatu sumber daya komputasi                                                              |

2. *Activity Diagram*: Sebuah diagram aktivitas UML menggambarkan perilaku dinamis dari suatu sistem atau bagian dari sistem melalui aliran kontrol antara aksi yang dilakukan sistem.

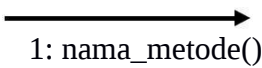
Table 2.3 Simbol Activity Diagram

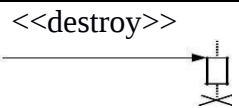
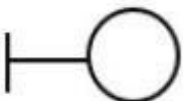
| No. | Gambar                                                                              | Nama                       | Keterangan                                                                    |
|-----|-------------------------------------------------------------------------------------|----------------------------|-------------------------------------------------------------------------------|
| 1   |    | <i>Activity</i>            | Memperlihatkan bagaimana kelas saling berinteraksi                            |
| 2   |    | <i>Decision</i>            | Asosiasi percabangan jika ada pilihan yang lebih dari satu                    |
| 3   |   | <i>Initial Node</i>        | Bagaimana objek diawali. Hanya ada satu initial dalam satu diagram.           |
| 4   |  | <i>Activity Final Node</i> | Bagaimana objek diakhiri. Dalam satu diagram minimal ada satu final state     |
| 5   |  | <i>Fork Node</i>           | Satu aliran pada tahap yang berubah menjadi beberapa aliran                   |
| 6   |  | <i>Control Flow</i>        | Menunjukkan urutan aliran aktivitas.                                          |
| 7   |  | <i>Partition</i>           | Simbol yang membatasi aktivitas antar orang, organisasi, sistem atau kelompok |

3. *Sequence Diagram* menggambarkan interaksi antara objek-objek di dalam maupun di sekitar sistem, termasuk *user*, tampilan (*display*), dan elemen lainnya, dalam bentuk *message* yang disusun berdasarkan urutan waktu. Diagram ini umumnya

digunakan untuk memvisualisasikan skenario atau tahapan respons terhadap suatu *event* guna menghasilkan output tertentu.

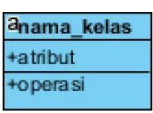
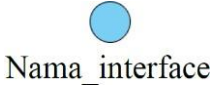
Table 2.4 Simbol Sequence Diagram

| No. | Gambar                                                                              | Nama                        | Keterangan                                                                                                              |
|-----|-------------------------------------------------------------------------------------|-----------------------------|-------------------------------------------------------------------------------------------------------------------------|
| 1   |    | <i>Actor</i>                | Orang , proses, atau sistem lain yang berinteraksi dengan system informasi yang akan dibuat                             |
| 2   |    | <i>Lifeline</i>             | Menyatakan kehidupan suatu objek                                                                                        |
| 3   |    | <i>Object</i>               | Menyatakan objek yang berinteraksi pesan                                                                                |
| 4   |  | <i>Timelife</i>             | Menyatakan objek dalam berinteraksi pesan                                                                               |
| 5   |  | Pesan tipe<br><i>Create</i> | Menyatakan objek dalam keadaan aktif dan berinteraksi                                                                   |
| 6   |  | Pesan tipe<br><i>Call</i>   | Menyatakan suatu objek memanggil operasi/ metode yang ada pada objek lain atau dirinya sendiri                          |
| 7   |  | Pesan tipe<br><i>Send</i>   | Menyatakan bahwa suatu objek mengirimkan data/masukan/informasi ke objek lain                                           |
| 8   |  | Pesan tipe<br><i>Return</i> | Menyatakan bahwa suatu objek telah menjalankan suatu operasi atau metode menghasilkan suatu kembalian ke objek tertentu |

| No. | Gambar                                                                            | Nama                      | Keterangan                                                                                      |
|-----|-----------------------------------------------------------------------------------|---------------------------|-------------------------------------------------------------------------------------------------|
| 9   |  | Pesan tipe <i>Destroy</i> | Menyatakan suatu objek telah menjalankan suatu operasi                                          |
| 10  |  | Boundary Class            | Berupan tepi dari sistem, seperti user interface atau alat yang berinteraksi dengan sistem lain |
| 11  |  | Control Class             | Mengatur aliran dari informasi untuk sebuah skenario                                            |

4. *Class Diagram* menyajikan representasi struktur sistem yang mencakup deskripsi mengenai *class*, *package*, dan *object* beserta hubungan di antaranya, seperti *inheritance*, *association*, dan relasi lainnya.

Table 2.5 Simbol Class Diagram

| No. | Gambar                                                                              | Nama                                             | Keterangan                                                        |
|-----|-------------------------------------------------------------------------------------|--------------------------------------------------|-------------------------------------------------------------------|
| 1   |  | Kelas                                            | Kelas pada struktur sistem                                        |
| 2   |  | Antarmuka/<br><i>Interface</i>                   | Sama dengan konsep interface dalam pemrograman berorientasi objek |
| 3   |  | Asosiasi/<br><i>Association</i>                  | Relasi antar kelas dengan makna umum                              |
| 4   |  | Asosiasi berarah/<br><i>Directed Association</i> | Relasi antar kelas dengan makna kelas                             |
| 5   |  | Generalisasi                                     | Relasi antar kelas dengan makna generalisasi                      |

| No. | Gambar                                                                            | Nama                                     | Keterangan                                                          |
|-----|-----------------------------------------------------------------------------------|------------------------------------------|---------------------------------------------------------------------|
| 6   |  | Kebergantungan<br>/<br><i>Dependency</i> | Relasi antar kelas<br>dengan makna<br>kebergantungan antar<br>kelas |
| 7   |  | Agregasi /<br><i>Aggregation</i>         | Relasi antar kelas<br>dengan makna whole-<br>part                   |

### 2.2.13 Black box testing

*Black box testing*, yang juga dikenal sebagai *Behavioral Testing*, merupakan metode pengujian perangkat lunak yang berfokus pada evaluasi hasil dari *input* dan *output* tanpa melihat proses internal atau cara kerja kode. Pengujian ini umumnya dilakukan pada tahap akhir pengembangan untuk memastikan bahwa sistem berjalan sesuai fungsinya dan telah memenuhi spesifikasi yang ditetapkan.