

# GOOD MORNING!

早上好!

안녕하세요!

---

PROJECT INTRODUCTION



# HOW TO WORK TOGETHER

---

- Participate, Participate, Participate!!!
- No long emails or Kakaotalk, prefer face to face
- Be open to suggestions and idea
- Be proactive, take initiative
- HOW is as important as WHAT

# BRAINSTORMING RULES

---

- Every input is good input
- Do not critique inputs only seek to understand
- Organize inputs into logical groupings
- Sequence or show relationships as needed
- Use Posted Notes on Flip Chart



# DAY I (DONE?)

---

- Welcome
- Project Introduction
- Introduction to Project Development Process
- Business Requirement Development
- System Requirement Development
- System(High Level) Design
- Time Management



# DAY 2

---

- YOLOv8 기반 데이터 수집/학습/deploy (Detection Alert)
  - 감시용 데이터 수집(bus, truck, tank 등)
  - 감시용 데이터 라벨링
  - YOLOv8 기반 학습
  - YOLOv8 Object Detection
- Porting to ROS
  - Create Detection Alert Node
  - Generate Topics to send image and Obj. Det. results
  - Create Subscriber node and display image and print data from the Topic

# DAY 3

---

- AMR (Autonomous Mobile Robot)기반 카메라 인식 autonomous driving 시스템 with obstacle avoidance 구축 (AMR Controller)
  - Digital Mapping of environment
  - Goal Setting and Obstacle Avoidance using Navigation
  - Object Tracking w/ AMR camera
  - Control logic between navigation/obj. tracking/ obj. following (teleop)
- Porting to ROS
  - Create AMR Controller Node
  - Create and send Obj.Tracking Image and data to Sysmon

# 프로젝트 RULE

---

**80/20 → 20/80**



# TEAMWORK AND PROJECT MANAGEMENT

---





프로젝트 RULE NUMBER ONE!!!

---

Have Fun Fun Fun!

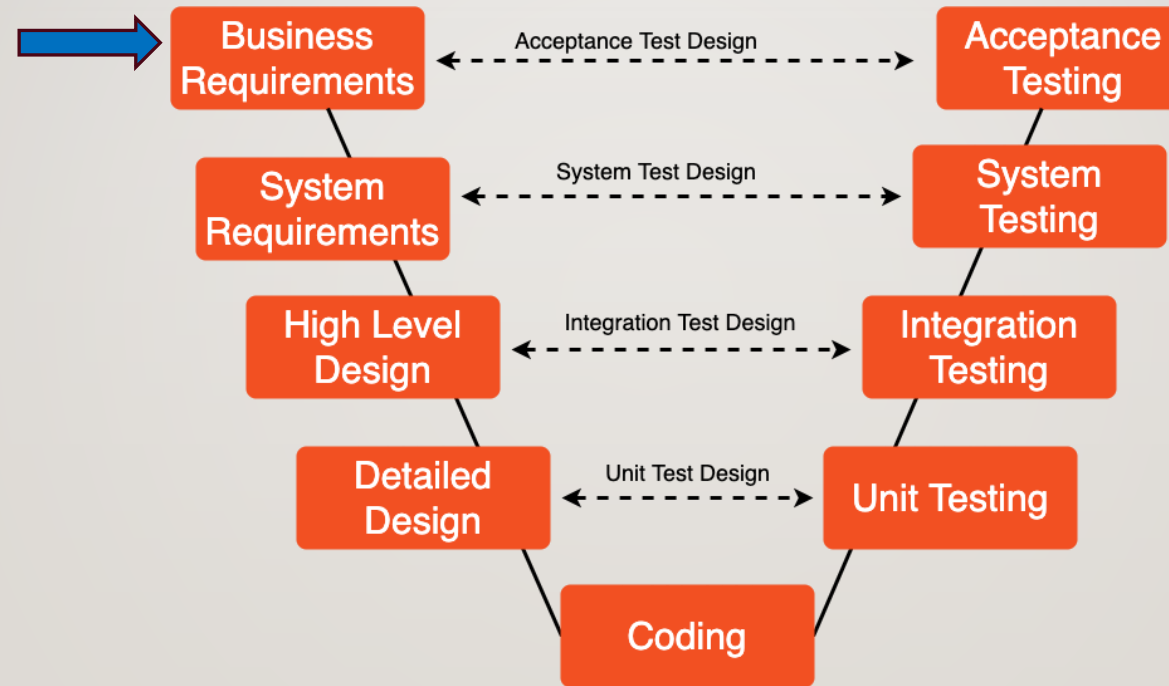


PROJECT DEVELOPMENT  
IS A PROCESS

---



# SW DEVELOPMENT PROCESS



SDLC - V Model - notepub.io

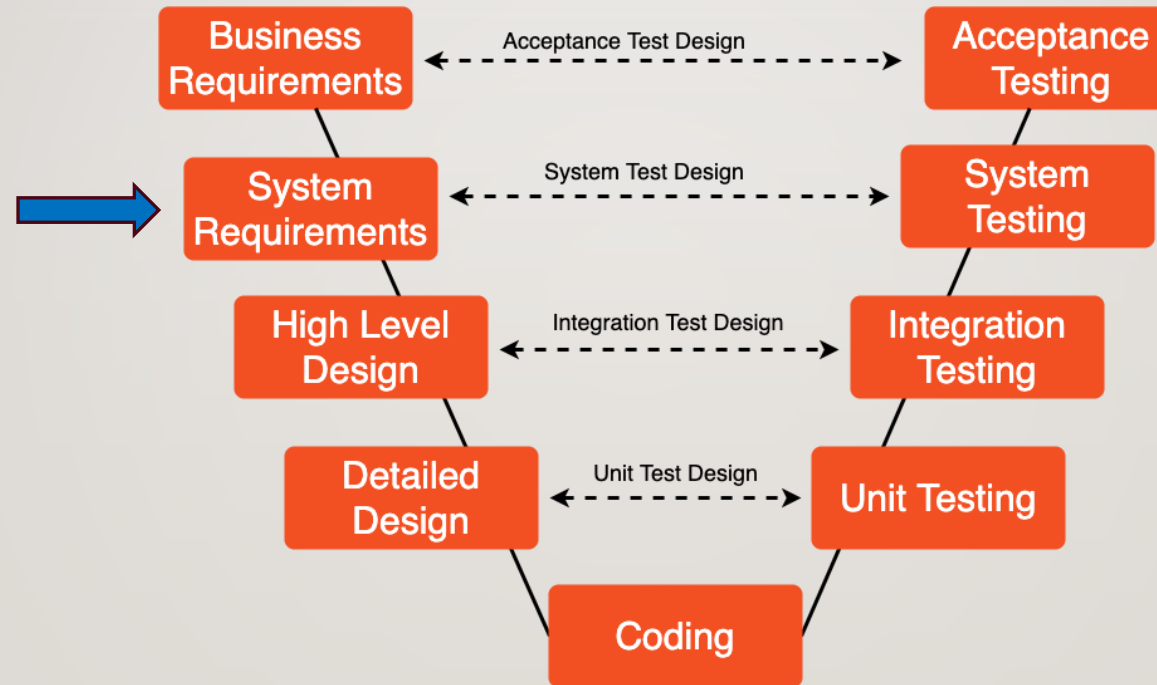


# TEAM EXERCISE I

---

Brainstorm Business Requirement for the project and write business requirement statement

# SW DEVELOPMENT PROCESS



SDLC - V Model - notepub.io

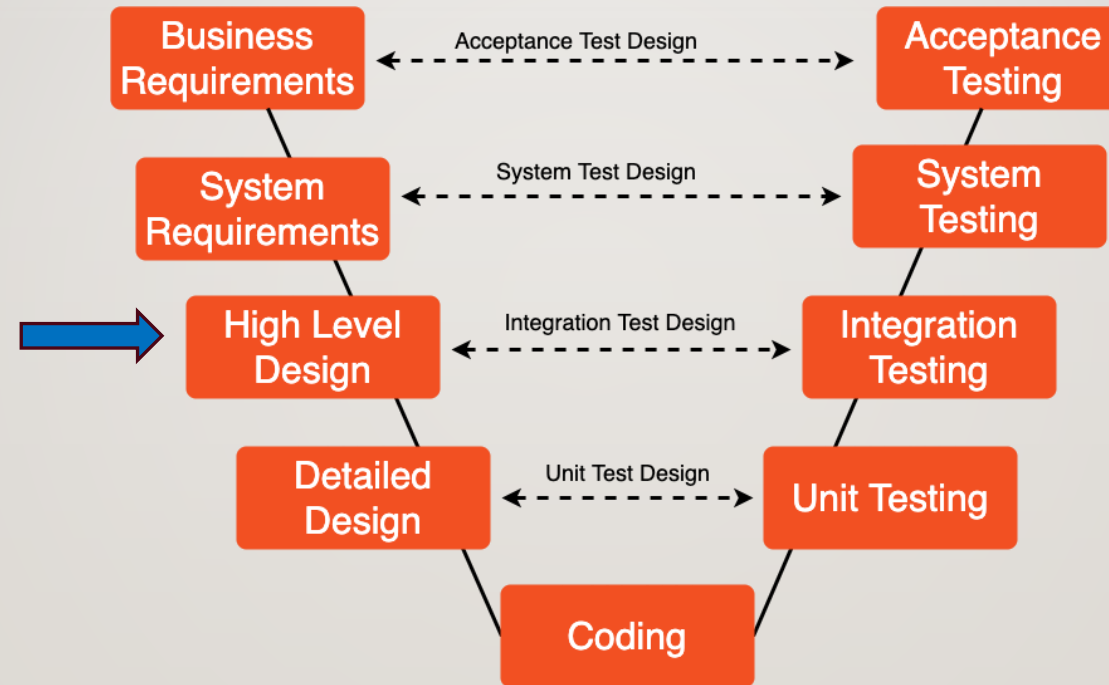
# TEAM EXERCISE 2

---

Brainstorm System Requirement for the project and document



# SW DEVELOPMENT PROCESS



SDLC - V Model - notepub.io

# KEY SUBSYSTEM (MODULES) TO DEVELOP

---

- Detection Alert

- Camera Capture
- Object Detection
- Send messages to other subsystems

- System Monitor

- Receive and Display Detection Camera and info
- Receive and Display AMR Camera and info
- Store, display, and report Information and Alerts

- AMR Controller

- Receive messages and act accordingly
- Move using (SLAM) with Obstruction avoidance
- Target Acquisition (Obj. Det.) and Tracking
- Follow target using camera and motor control

# TEAM EXERCISE 3

---

Create System Design using Process Flow Diagram.

Use the posted notes and flipchart as needed



# SYSTEM DESIGN PRESENTATION BY EACH TEAM

---



# TEAM EXERCISE 4

---

System Environment and Development Environment Setup

# USEFUL COMMANDS

---

\$ lsb\_release -a

- Linux distribution info

\$ echo \$ROS\_DISTRO

- ROS: Humble

\$ code --version

- Vscode

\$ python3 --version

- Python

\$ sudo apt update

\$ sudo apt upgrade

\$ python -m ensurepip --upgrade

- Assumes Linux (Ubuntu 22.04), ROS Humble, VScode, and Python are already installed globally



# REQUIRED PACKAGES SETUP

---

\$ pip list | grep opencv

If doesn't exist....

\$ pip3 install opencv-python

\$ pip3 install opencv-contrib-python

\$ pip list | grep ultra

If doesn't exist....

\$ pip install ultralytics

\$ pip freeze > requirements.txt

\$ pip install -r requirements.txt

# ROS2 DEVELOPMENT WORKSPACE

---

```
$ cat ~/.bashrc
```

```
$ echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc #check path
```

```
$ sudo apt install python3-colcon-common-extensions
```

```
$ echo "source /usr/share/colcon_argcomplete/hook/colcon_argcomplete.bash" >> ~/.bashrc  
#check path
```

```
$ source ~/.bashrc
```

# ROS2 DEVELOPMENT WORKSPACE

---

## CREATE WORKSPACE

```
$ mkdir -p ~/ros2_ws/src
```

```
$ cd ~/ros2_ws
```

```
$ rosdep install --from-paths src --ignore-  
src -r -y
```

If not installed...

```
$ sudo rosdep init
```

```
$ rosdep update
```

## \*NOT CREATED UNTIL COLCON

```
workspace/      # Root of the workspace  
├── src/         # Source code (ROS packages)  
├── build/       # Build files (generated by colcon)  
├── install/     # Installed packages and setup scripts  
└── log/         # Build logs
```

```
$ colcon build
```

```
$ source install/setup.bash
```

# ROS2 DEVELOPMENT WORKSPACE

```
$ cd ~/ros2_ws/src
```

```
$ ros2 pkg create --build-type  
ament_python <my_package>
```

```
my_package/  
├─ package.xml          # Package metadata and dependencies  
├─ setup.py             # Build instructions for Python packages  
├─ setup.cfg            # Optional, configures metadata for setuptools  
├─ launch/              # Launch files for starting nodes (optional)  
├─ config/              # Configuration files (optional)  
├─ resource/            # Empty file matching package name for ament index  
├─ my_package/          # Python package directory (contains code)  
│   └─ __init__.py      # Makes this directory a Python package  
│   └─ my_node.py       # Example Python node  
└─ msg/                 # Message definitions (optional)
```



# ROS2 DEVELOPMENT WORKSPACE

---

Write your code below the `my_package/` directory under `my_package/ package directory`

```
my_package/
├─ package.xml          # Package metadata and dependencies
├─ setup.py             # Build instructions for Python packages
├─ setup.cfg            # Optional, configures metadata for setuptools
├─ launch/              # Launch files for starting nodes (optional)
├─ config/              # Configuration files (optional)
├─ resource/            # Empty file matching package name forament inc
├─ my_package/          # Python package directory (contains code)
│   └─ __init__.py      # Makes this directory a Python package
│   └─ my_node.py       # Example Python node
└─ msg/                 # Message definitions (optional)
```

# ROS2 DEVELOPMENT WORKSPACE

---

\$ cd ~/ros2\_ws

\$ colcon build

\$ echo "source

~/ros2\_ws/install/setup.bash" >>

~/.bashrc

\$ source ~/.bashrc

```
workspace/      # Root of the workspace
├─ src/         # Source code (ROS packages)
├─ build/       # Build files (generated by colcon)
├─ install/     # Installed packages and setup scripts
└─ log/         # Build logs
```

#check path

# PROJECT TIMELINE/CRITICAL PATH ITEM MANAGEMENT

---



# EX. IMPLEMENTATION TIMELINE

Function Backlog	Owner	5월 20일	5월 21일	5월 22일	5월 23일	5월 24일	5월 25일
<b>Unloading Module</b>	John						
Input1	John						
Input2	John						
Output 1	John						
Unit Test	John						
<b>Receiving Module</b>	Jan						
Input1	Feb						
Input2	Mar						
Output 1	Apr						
Unit Test	John						
Integration Test	John/Jan						

이 타임라인을 생성할 때  
먼저 시스템 및 시스템  
설계의 기능 프로세스  
다이어그램(To-Be)을  
완료해야 합니다.

그런 다음 각 기능(하위  
함수/모듈 및  
입력/출력)에 대해 누가,  
무엇을, 언제, 어떻게  
정의합니다. 표에 설명  
타임라인 형식의 무엇을,  
누가, 언제를 입력합니다.



# CRITICAL PATH ITEMS LIST

---

- tasks that directly impact the project timeline. Delays in these tasks would delay the project's overall completion because they represent the longest stretch of dependent activities
- 프로젝트 타임라인에 직접적인 영향을 주는 작업입니다. 이러한 작업이 지연되면 종속 활동이 가장 길어지기 때문에 프로젝트의 전체 완료가 지연됩니다

# CRITICAL PATH ITEMS LIST

---

- Examples:
  1. **AI Model Development:** Fine-tuning deep learning models like CNNs for precise item recognition and sorting, requiring data gathering, model training, and testing.
  2. **Robot Integration:** Embedding AI software into robots to enable precise task execution, focusing on software-hardware compatibility and function tests.
  3. **System Testing:** Comprehensive testing of AI and robot performance in simulated environments to ensure operational reliability.
- AI 모델 개발: 정확한 항목 인식 및 정렬을 위해 CNN과 같은 딥 러닝 모델을 미세 조정하여 데이터 수집, 모델 학습 및 테스트가 필요합니다.
- 로봇 통합: AI 소프트웨어를 로봇에 내장하여 소프트웨어-하드웨어 호환성 및 기능 테스트에 중점을 두고 정확한 작업 실행을 가능하게 합니다.
- 시스템 테스트: 시뮬레이션 환경에서 AI 및 로봇 성능을 종합적으로 테스트하여 운영 안정성을 보장합니다.

# GUIDE TO PROGRESS INDICATORS

---

- Project Timeline
  - **Green** – less 10% of the listed items are delayed
  - **Yellow** – more than 10% but less than 20% of listed items are delayed
  - **Red** – more than 20% of listed items are delayed
- Critical Path Items
  - **Green** – reduced number of item(s)
  - **Yellow** – no new item(s)
  - **Red** – additional item(s)

# PROJECT TIMELINE/CRITICAL PATH ITEM MANAGEMENT

---





# EX. IMPLEMENTATION TIMELINE

Function Backlog	Owner	5월 20일	5월 21일	5월 22일	5월 23일	5월 24일	5월 25일
<b>Unloading Module</b>	John						
Input1	John						
Input2	John						
Output 1	John						
Unit Test	John						
<b>Receiving Module</b>	Jan						
Input1	Feb						
Input2	Mar						
Output 1	Apr						
Unit Test	John						
Integration Test	John/Jan						

이 타임라인을 생성할 때  
먼저 시스템 및 시스템  
설계의 기능 프로세스  
다이어그램(To-Be)을  
완료해야 합니다.

그런 다음 각 기능(하위  
함수/모듈 및  
입력/출력)에 대해 누가,  
무엇을, 언제, 어떻게  
정의합니다. 표에 설명  
타임라인 형식의 무엇을,  
누가, 언제를 입력합니다.

# CRITICAL PATH ITEMS LIST

---

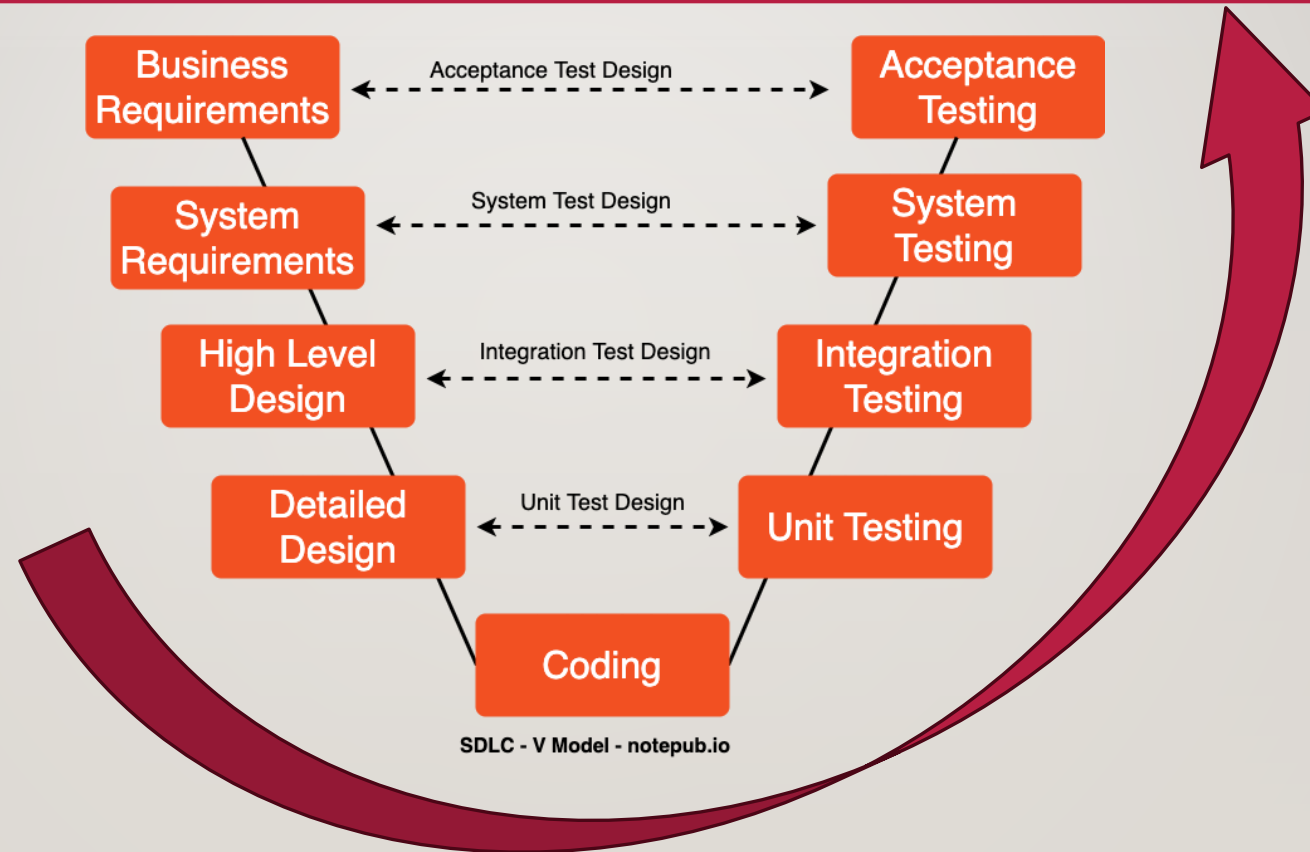
- tasks that directly impact the project timeline. Delays in these tasks would delay the project's overall completion because they represent the longest stretch of dependent activities
- 프로젝트 타임라인에 직접적인 영향을 주는 작업입니다. 이러한 작업이 지연되면 종속 활동이 가장 길어지기 때문에 프로젝트의 전체 완료가 지연됩니다

# DETAIL DESIGN TO USER ACCEPTANCE

---

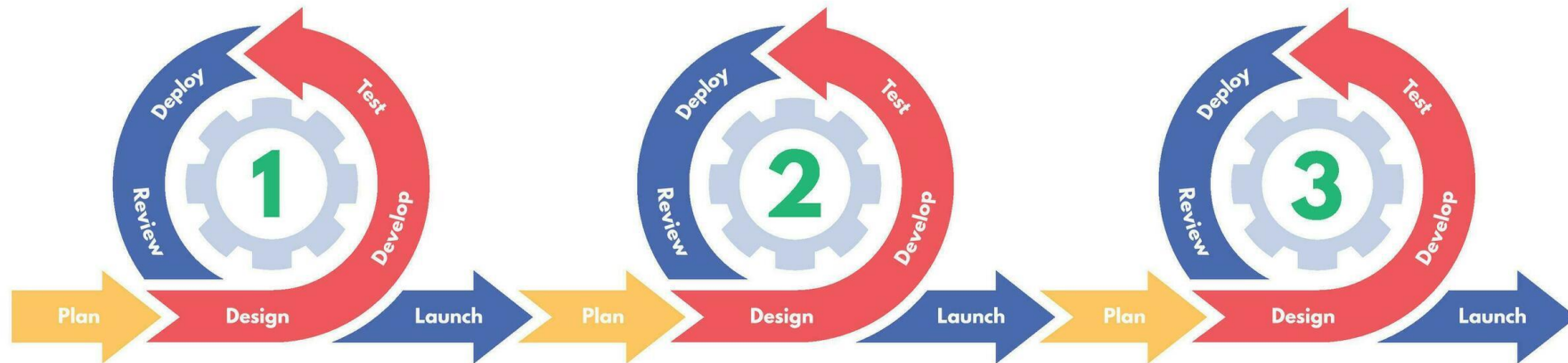


# SW DEVELOPMENT PROCESS

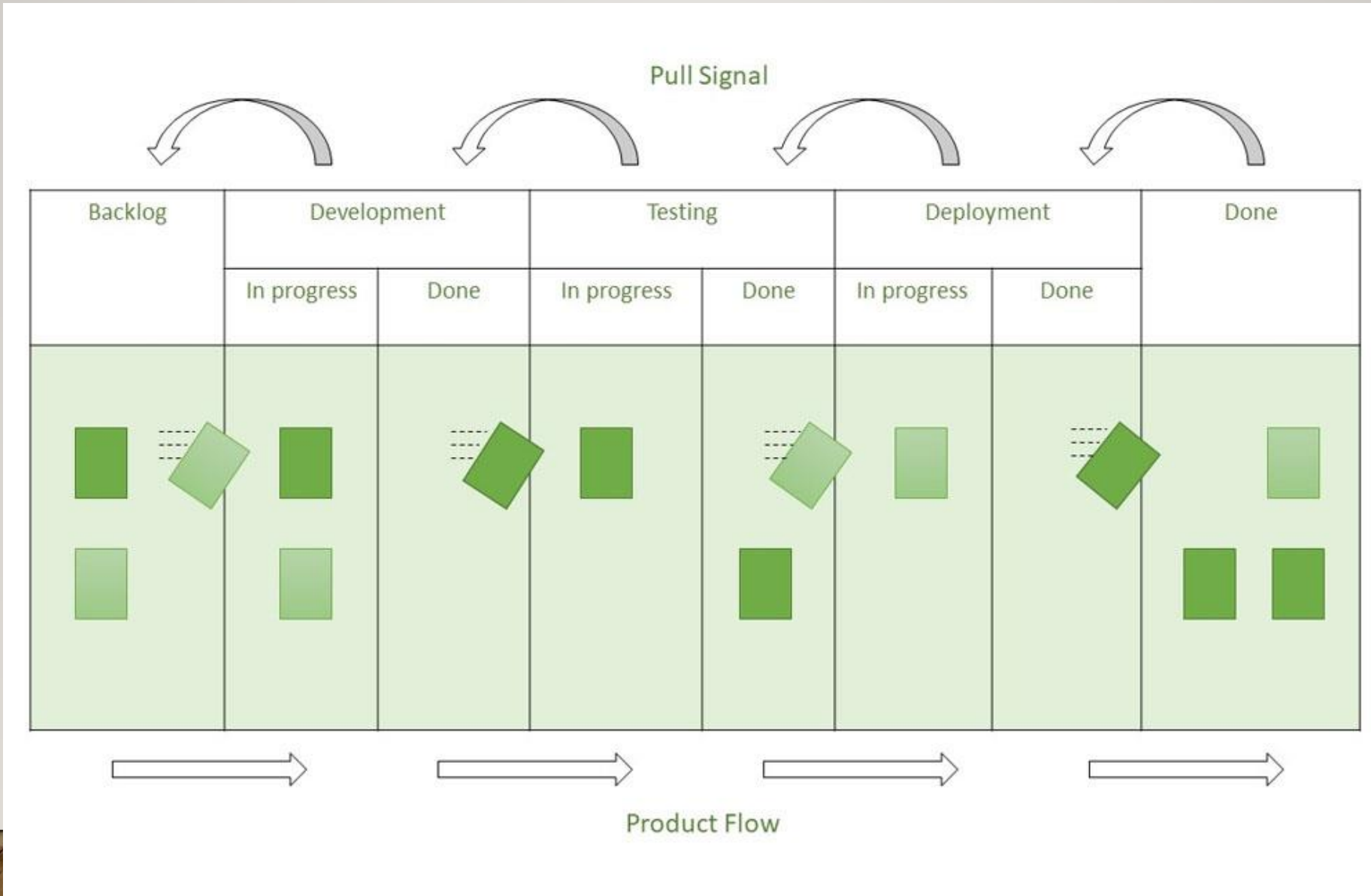




# AGILE DEVELOPMENT



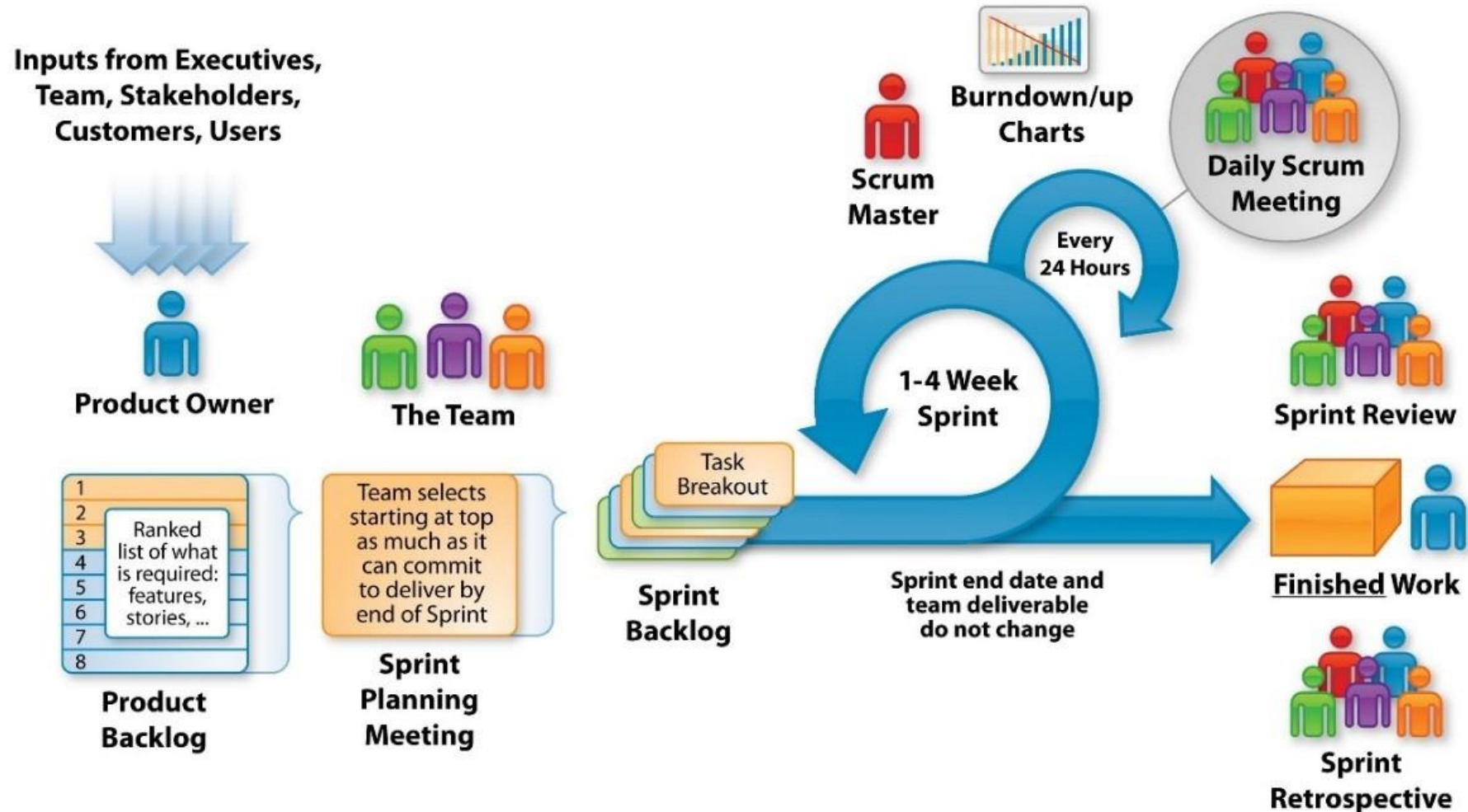
# KANBAN METHODOLOGY







# The Agile - Scrum Framework

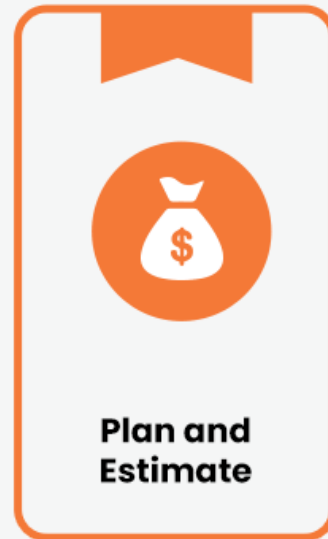




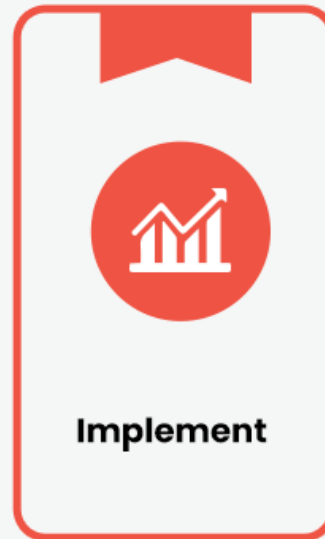
# 5 Stages of Scrum Sprint



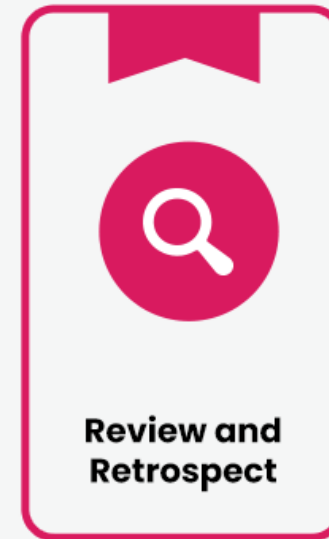
This phase includes the processes related to the commencement of a project, such as a scope and objectives, creating and distributing its charter, and taking other steps to guarantee success.



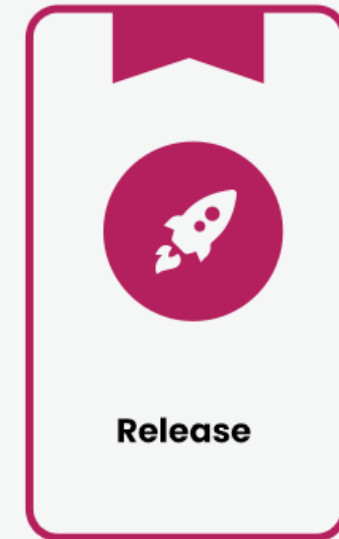
This phase involves planning and estimating processes, including creating user stories, approving, assessing, committing user stories, creating tasks, evaluating tasks, and creating a Sprint backlog.



This phase is about executing the tasks and activities to create a product. These activities include building the various outputs, conducting daily standup meetings, and grooming the product backlog.

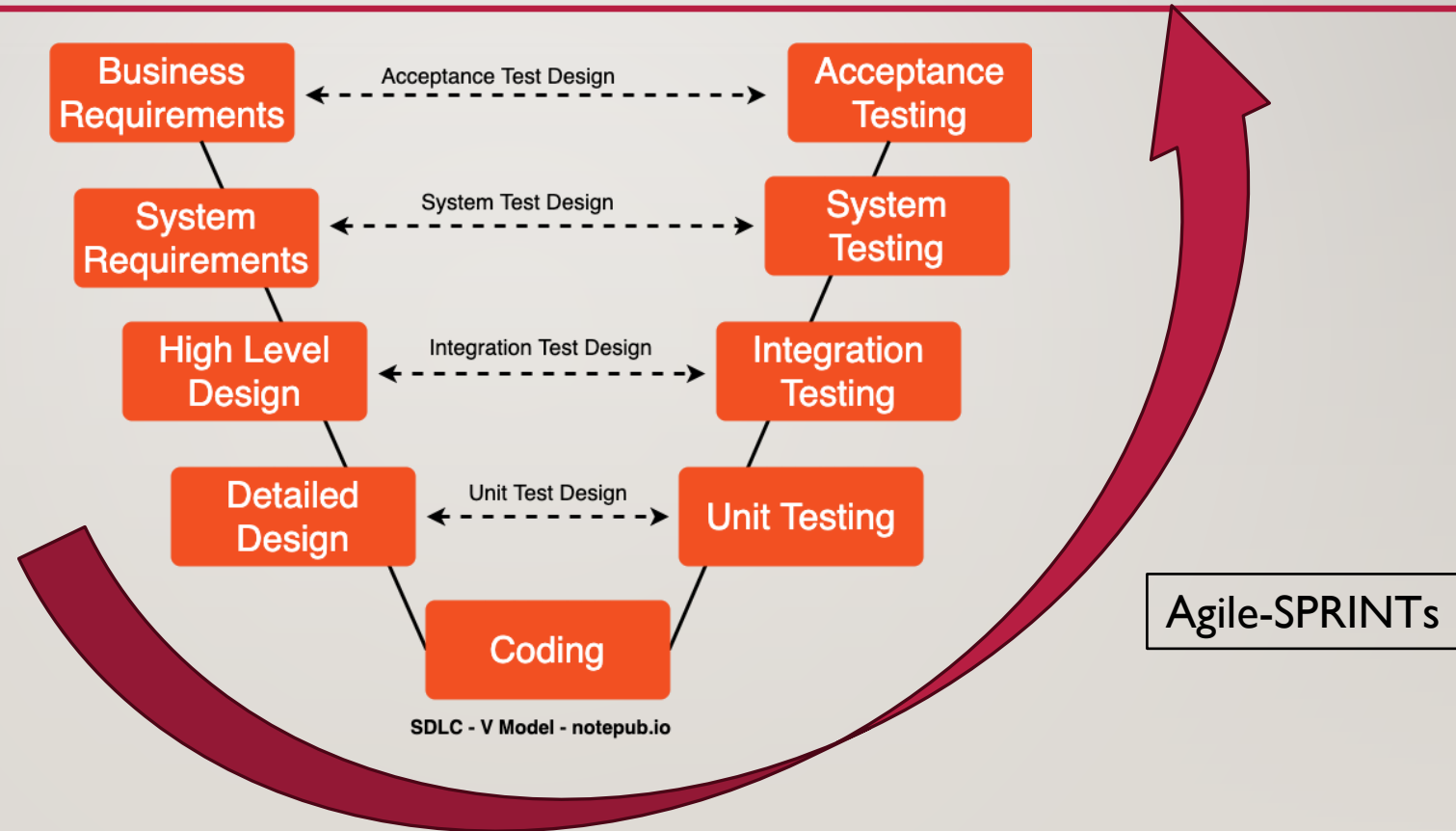


This stage of the project lifecycle is concerned with evaluating what has been accomplished so far, whether the team has worked to plan, and how it can do things better in the future.



This stage highlights delivering the accepted deliverables to the customer and determining, documenting, and absorbing the lessons learned during the project.

# SW DEVELOPMENT PROCESS



# PROJECT SPRINTS

- Detection Alert

- Camera Capture
- Object Detection
- Send messages to other subsystems

- AMR Controller

- Receive messages and act accordingly
- Move using (SLAM) with Obstruction avoidance
- Target Acquisition (Obj. Det.) and Tracking
- Follow target using camera and motor control

- System Monitor

- Receive and Display Detection Camera and info
- Receive and Display AMR Camera and info
- Store, display, and report Information and Alerts

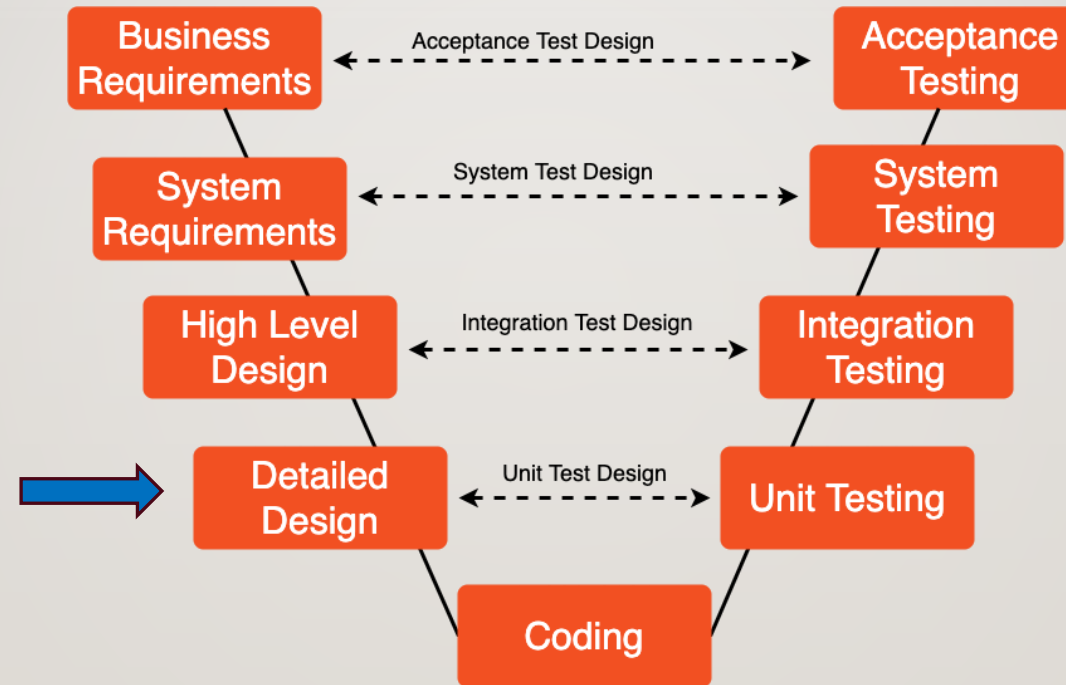
# DETECTION ALERT SPRINT

---





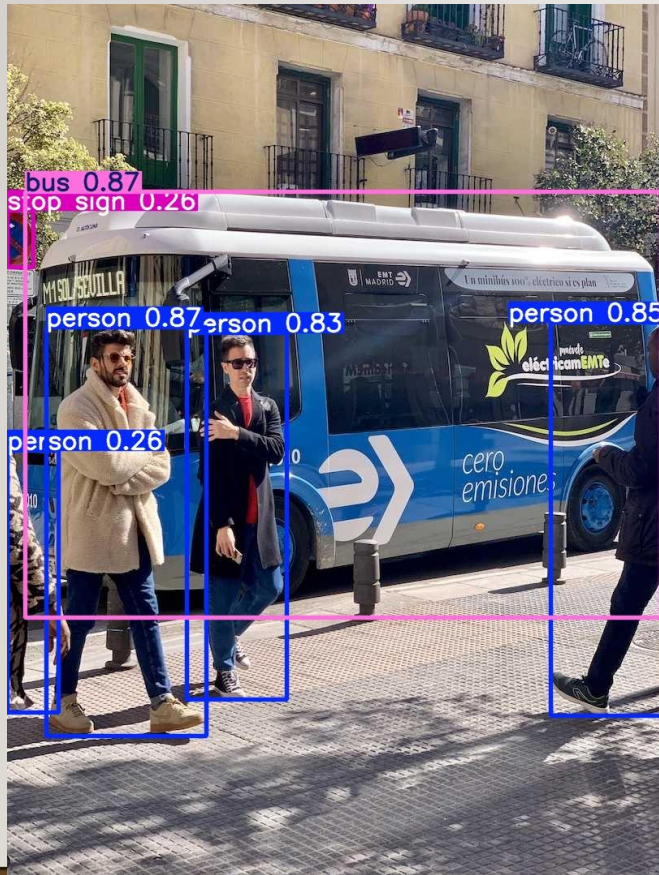
# SPRINT I - DETECTION ALERT



SDLC - V Model - notepub.io

# YOLO OBJ. DET. VS. YOLO TRACKING

---

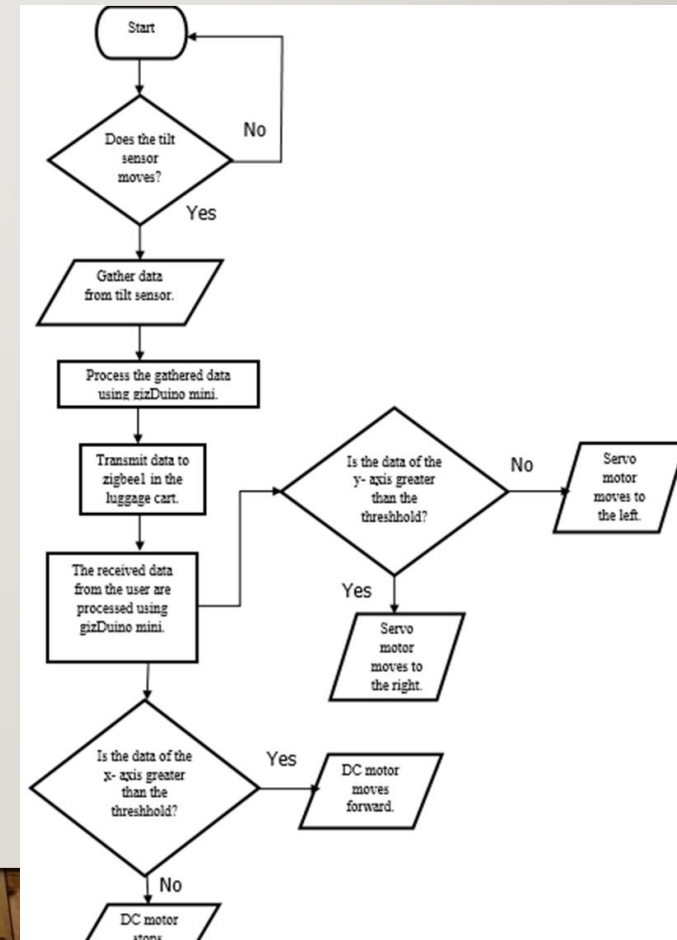


- [\(469\) YOLO people detection + SORT tracking – YouTube](#)
- [Bing Videos](#)
- [Track - Ultralytics YOLO Docs](#)

# VISUALIZATION – SYSTEM FUNCTIONAL PROCESS DIAGRAMS

- To-Be Functional Process Diagram

Detection Alert



# TEAM EXERCISE 5

---

Perform Detail Design of Detection Alert Module using Process Flow Diagram



# DETAIL DESIGN REVIEW BY EACH TEAM

---

Using the process flow diagram present team's design

# EXAMPLE DETAILED DESIGN DOCUMENT

## Detailed Design Document: AMR Navigation and Threat Detection

Project Title: Autonomous Mobile Robot (AMR) Security System

Version: 1.0

Date: [Insert Date]

### 1. Overview

This document outlines the detailed design for the Autonomous Mobile Robot (AMR) navigation and threat detection components. It covers the architecture, algorithms, data processing, and system interactions necessary to enable autonomous navigation within a secure area and real-time threat detection using onboard sensors.

### 2. System Architecture

The AMR system relies on onboard hardware (e.g., sensors, cameras, Jetson-Orin processor) and software (ROS2, OpenCV, YOLO) for autonomous navigation and real-time threat detection. All processing occurs locally on the AMR, with the capability to transmit alerts to a monitoring PC via Wi-Fi.

## 상세 설계 문서: AMR 네비게이션 및 위협 탐지

프로젝트 제목: 자율 이동 로봇(AMR) 보안 시스템

버전: 1.0

날짜: [날짜 삽입]

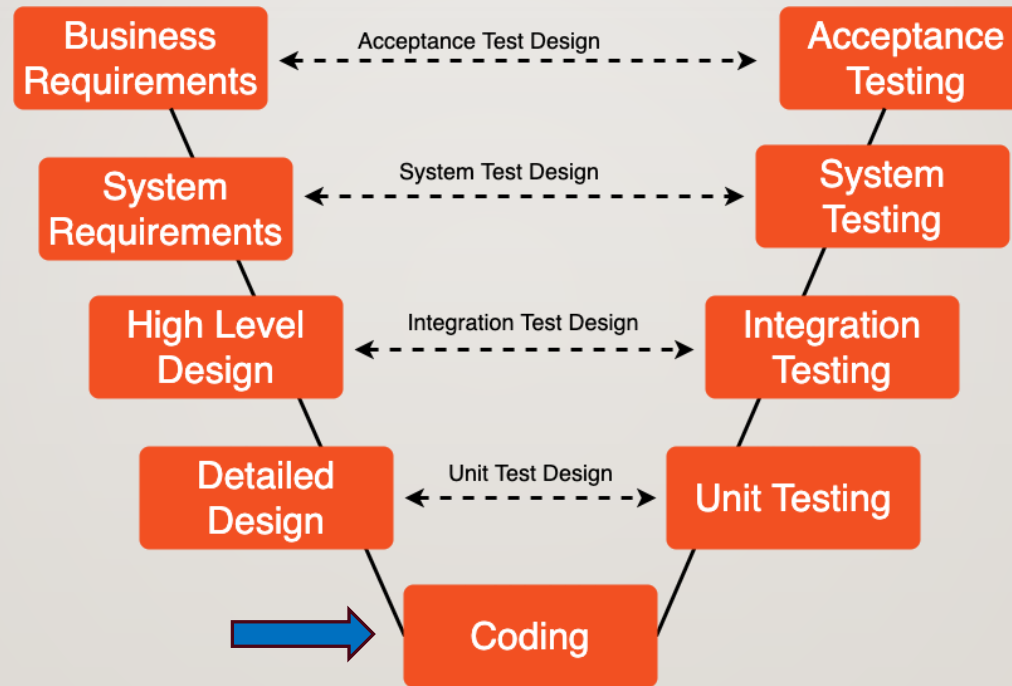
### 1. 개요

이 문서는 자율 이동 로봇(AMR)의 네비게이션 및 위협 탐지 구성 요소에 대한 상세 설계를 다룹니다. 자율 네비게이션과 실시간 위협 탐지를 위해 온보드 센서를 사용하는 데 필요한 아키텍처, 알고리즘, 데이터 처리 및 시스템 상호작용이 포함되어 있습니다.

### 2. 시스템 아키텍처

AMR 시스템은 자율 네비게이션 및 실시간 위협 탐지를 위해 온보드 하드웨어(예: 센서, 카메라, Jetson-Orin 프로세서)와 소프트웨어(ROS2, OpenCV, YOLO)를 활용합니다. 모든 처리는 AMR 내에서 로컬로 수행되며, 잠재적인 위협이 감지되면 Wi-Fi를 통해 모니터링 PC로 알림을 전송할 수 있습니다.

# SPRINT I - DETECTION ALERT



SDLC - V Model - notepub.io

# CODING HINTS

---


- create your team working folder and place files and work from here


\$ mkdir


~/rokeyl\_<grp\_letter><grp\_num>\_ws

(i.e. mkdir ~/rokeyl\_A2\_ws)

- Image Capture

 0.capture\_image.py 1

 1.cont\_capture\_image.py

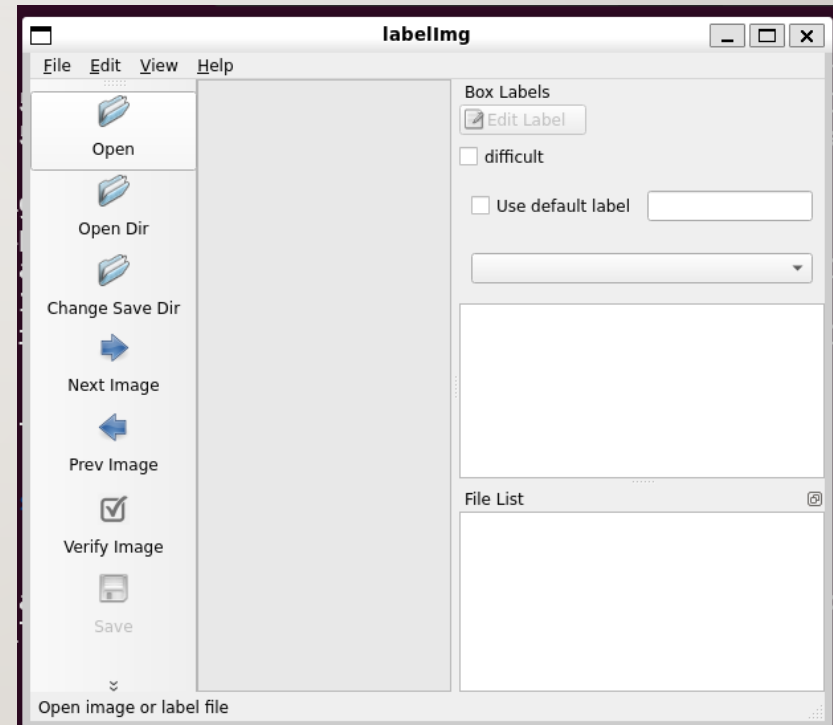
 2.labellmg.exe



# CODING HINTS

---

- Data Labelling : use previously installed Labellmg
- Or
- `pip3 install PyQt5 lxml`
- `pip3 install labellmg`
- `labellmg`



# CODING HINTS

---

- Data Labelling : Labellmg

## 라벨링 순서

1. 이미지파일 불러오기 (Open Dir)
2. 저장형식 변경 (PascalVOC, YOLO)
3. 이미지 선택
4. 바운딩 박스 그리기(create rectbox)
5. Class 지정
6. 저장경로 생성 및 변경(Change Save Dir)
7. 저장(Save)

## 단축키

Ctrl + u	Load all of the images from a directory
Ctrl + r	Change the default annotation target dir
Ctrl + s	Save
Ctrl + d	Copy the current label and rect box
Ctrl + Shift + d	Delete the current image
Space	Flag the current image as verified
w	Create a rect box
d	Next image
a	Previous image
del	Delete the selected rect box
Ctrl++	Zoom in
Ctrl--	Zoom out
↑→↓←	Keyboard arrows to move selected rect box

# CODING HINTS

---

- Image Capture
- Data Labelling
- Preprocessing

```
0.capture_image.py
1.cont_capture_image.py
2.labellmg.exe
3.create_data_dirs.py
4.move_image.py
5.move_labels.py
```

# PERFORM DATA COLLECTION FOR DETECTION ALERT

---





# CODING HINTS

---

- Image Capture
- Data Labelling
- Preprocessing
- Yolo8 Object Det.

```
0.capture_image.py
1.cont_capture_image.py
2.labellmg.exe
3.create_data_dirs.py
4.move_image.py
5.move_labels.py
6.yolov8_obj_det_ak.ipynb
7.yolov8_obj_det_best.py
```

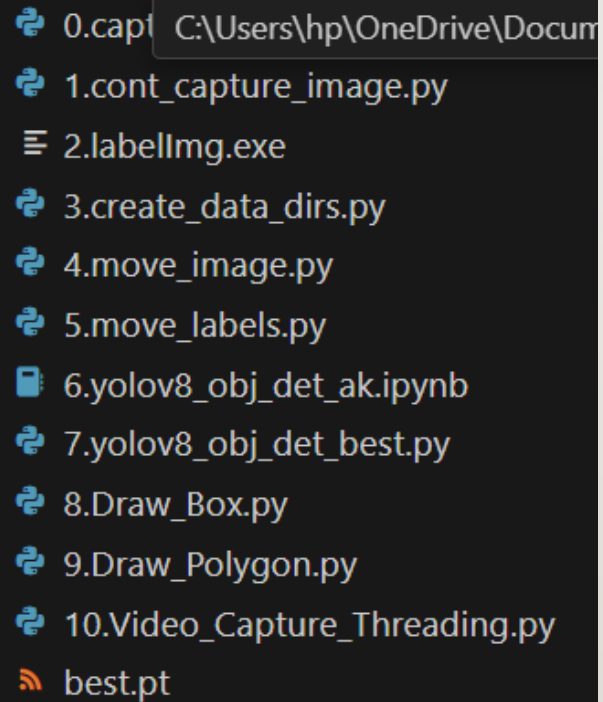
HOW DID YOU DEFINE DETECTION  
AREA?

---



# CODING HINTS

---



0.capt C:\Users\hp\OneDrive\Docum  
1.cont\_capture\_image.py  
2.labellmg.exe  
3.create\_data\_dirs.py  
4.move\_image.py  
5.move\_labels.py  
6.yolov8\_obj\_det\_ak.ipynb  
7.yolov8\_obj\_det\_best.py  
8.Draw\_Box.py  
9.Draw\_Polygon.py  
10.Video\_Capture\_Threading.py  
best.pt

- Create your alert condition

# PORTING TO ROS

---



# ROS2 DEVELOPMENT WORKSPACE

---

## CREATE/GOTO WORKSPACE

```
$ mkdir  
~/rokeyl_<grp_letter><grp_num>_ws
```

(i.e. mkdir ~/rokeyl\_A2\_ws)

Or

```
$ cd ~/rokeyl_A2_ws
```

## \*NOT CREATED UNTIL COLCON

```
workspace/      # Root of the workspace  
├── src/         # Source code (ROS packages)  
├── build/       # Build files (generated by colcon)  
├── install/     # Installed packages and setup scripts  
└── log/        # Build logs
```

# ROS2 DEVELOPMENT WORKSPACE

---

## CREATE ROS VIRTUAL ENVIRONMENT

If you want to usual virtual env, you should create it at the workspace level

```
$ cd ~/rokeyl_A2_ws
```

```
$ python3 -m venv <NAME>
```

```
$ source <NAME>/bin/activate
```

```
$ deactivate
```

# ROS2 DEVELOPMENT WORKSPACE

```
$ cd ~/rokeyl_A2_ws
```

```
$ ros2 pkg create --build-type  
ament_python <my_package>
```

Or

```
$ ros2 pkg create <package_name> --  
build-type ament_python --dependencies  
rcldpy std_msgs ament_index_python
```

```
my_package/  
├─ package.xml          # Package metadata and dependencies  
├─ setup.py             # Build instructions for Python packages  
├─ setup.cfg            # Optional, configures metadata for setuptools  
├─ launch/              # Launch files for starting nodes (optional)  
├─ config/              # Configuration files (optional)  
├─ resource/            # Empty file matching package name for ament index  
├─ my_package/          # Python package directory (contains code)  
│   └─ __init__.py      # Makes this directory a Python package  
│   └─ my_node.py       # Example Python node  
└─ msg/                 # Message definitions (optional)
```

# ROS2 DEVELOPMENT WORKSPACE

---

Write your code below the `my_package/` directory under `my_package/ package directory`

```
my_package/
├─ package.xml          # Package metadata and dependencies
├─ setup.py             # Build instructions for Python packages
├─ setup.cfg            # Optional, configures metadata for setuptools
├─ launch/              # Launch files for starting nodes (optional)
├─ config/              # Configuration files (optional)
├─ resource/            # Empty file matching package name forament inc
├─ my_package/          # Python package directory (contains code)
│   └─ __init__.py      # Makes this directory a Python package
│   └─ my_node.py       # Example Python node
└─ msg/                 # Message definitions (optional)
```



# ROS2 DEVELOPMENT WORKSPACE

---

```
$ cd ~/rokeyl_A2_ws
```

```
$ colcon build
```

or

```
$ colcon build --packages-select  
  <package_name>
```

or

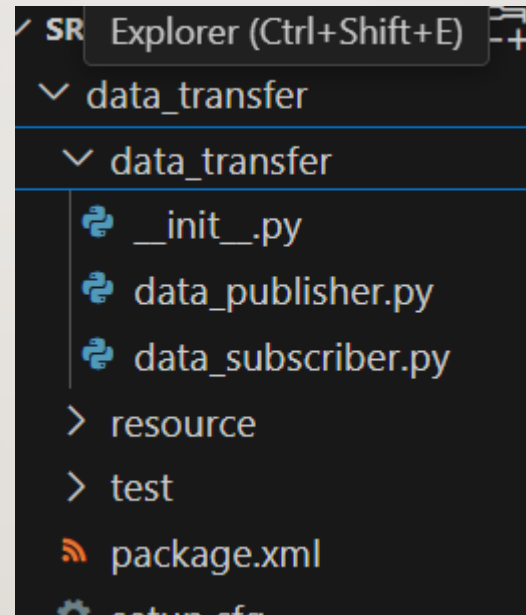
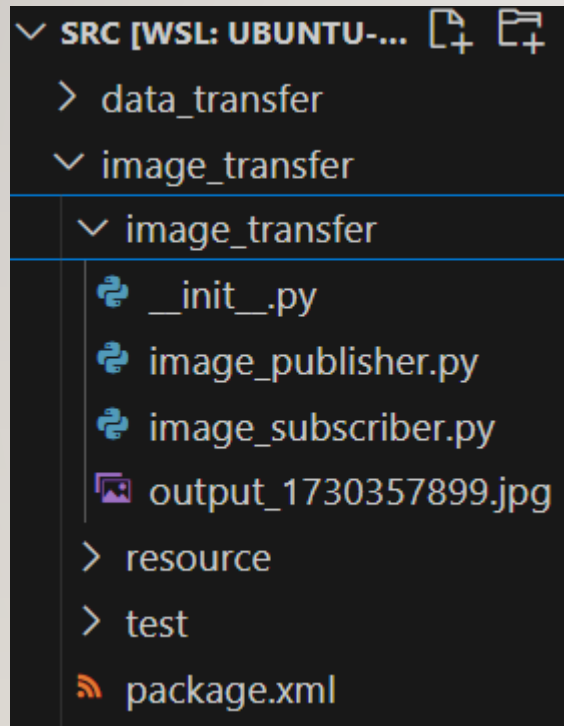
```
$ colcon build --packages-select  
  <package_name> --packages-skip-up-to-  
  date
```

```
$ rm -rf build/<package_name>  
  install/<package_name> log
```

```
$ colcon build --packages-select  
  <package_name>
```

```
workspace/      # Root of the workspace  
├─ src/         # Source code (ROS packages)  
├─ build/       # Build files (generated by colcon)  
├─ install/     # Installed packages and setup scripts  
└─ log/         # Build logs
```

# ROS HINTS



\$ ros2 interface list

# ROS HINTS

---

- Edit setup.py under <package\_name>  
directory add entry for each node

```
entry_points={ 'console_scripts':  
[ '<command_name> =  
<package_name>.<code_filename>:main', },
```

<command\_name> is used when ros2 run  
is executed i.e. data\_publisher

```
entry_points={  
    'console_scripts': [  
        'data_pub = data_transfer.data_publisher:main',  
        'data_sub = data_transfer.data_subscriber:main',  
    ],  
}
```

# ROS HINTS

---

\$ cd ~/rokeyl\_A2\_ws

\$ source  
~/rokeyl\_A2\_ws/install/setup.bash

\$ ros2 run <package\_name>  
<command\_name>

\$ sudo apt update

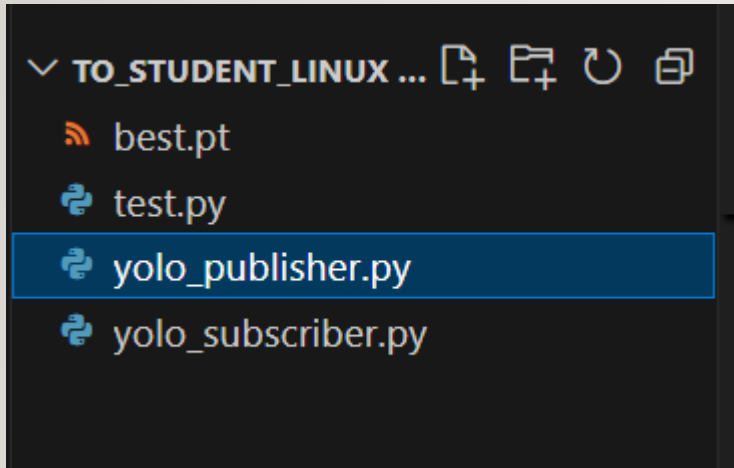
\$ sudo apt install terminator

```
391  ros2 run data_transfer data_pub
```



# ROS HINTS

---



\$ ros2 run rqt\_graph rqt\_graph

\$ ros2 node list

\$ ros2 node info <node\_name>

\$ ros2 topic list

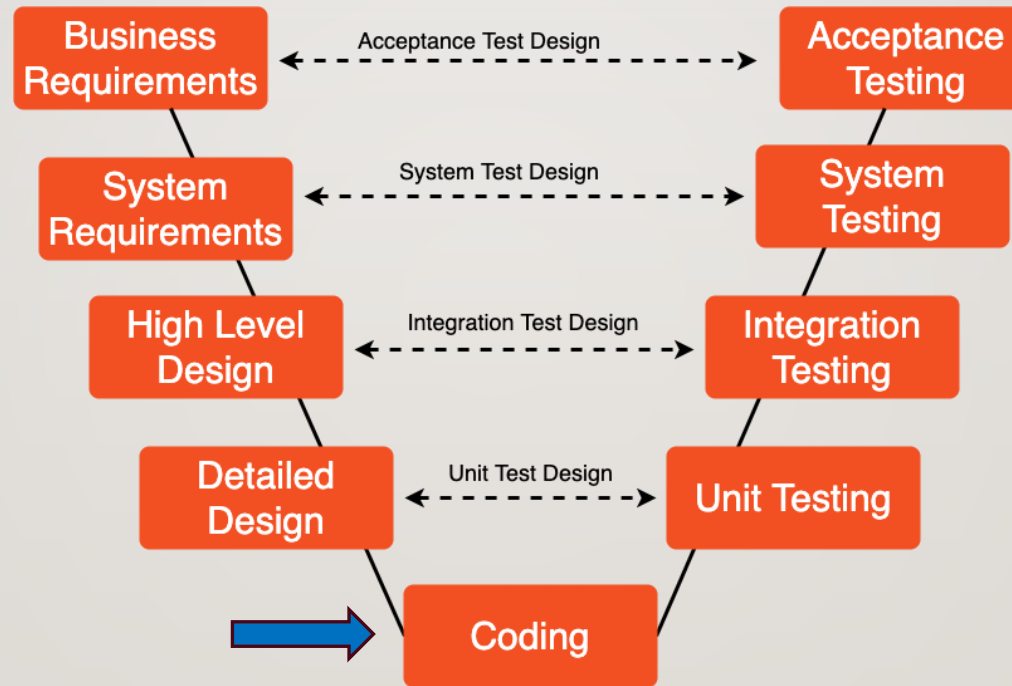
\$ ros2 topic info <topic\_name>

\$ ros2 topic echo /chatter

\$ ros2 interface list

\$ ros2 interface show  
<package\_name>/msg/<MessageName>

# SPRINT I - DETECTION ALERT



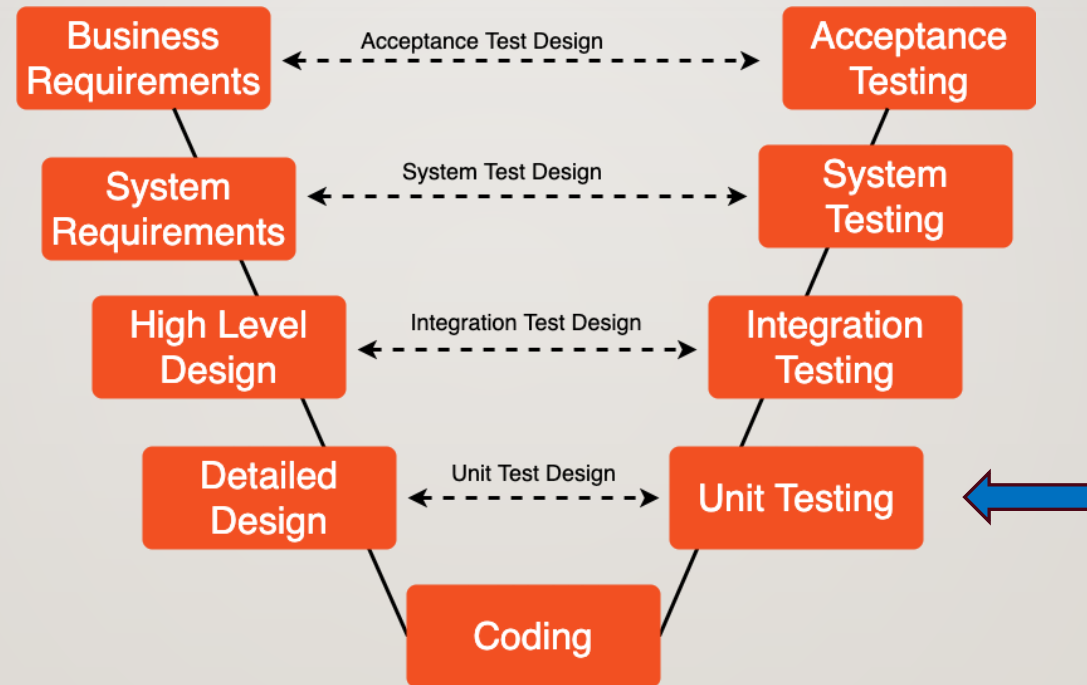
SDLC - V Model - notepub.io

# EXPECTED OUTCOME

---

- Successful object detection
- ROS Nodes, and Topics created to send and display images and data

# SPRINT I - DETECTION ALERT



SDLC - V Model - notepub.io



# TEAM EXERCISE 6

---

Perform coding and testing of Detection Alert Module

# RESULTS & CODE REVIEW BY EACH TEAM

---

Show actual results against the expected results and explain the code written