Capstone Project Plan: Manufacturing Subsystem Development

Sample Project

"Manufacturing Quality Control and Reporting System"

Objective

Develop a Python-based manufacturing subsystem for **Quality Control and Reporting**. Participants will implement a functional system focusing on core programming skills, data validation, role-based access control, file handling, and basic GUI development within three days.

Scope

The project focuses on:

1. **Product Quality Tracking**:
    - Log product inspection results (e.g., pass/fail, defects).
    - Associate inspection results with machine and production data.
2. **Data Validation**:
    - Validate inputs (e.g., numeric values, dates, product types).
3. **Role-Based Access Control**:
    - Restrict data entry and reports to authorized roles.
4. **Report Generation**:
    - Generate and export quality control summaries and defect reports.
5. **GUI Implementation**:
    - Simple GUI for data input and report generation using tkinter.

Timeline

Day 1: System Design

- **Objective**: Create class hierarchies and validate data inputs.
    - Design Product and Machine class hierarchies.
    - Implement Validator and custom exception classes.
    - Define data structures for quality control logs.

Day 2: GUI and Core Features

- **Objective**: Develop the GUI and core functionalities.
    - Implement forms for logging inspection results.
    - Develop role-based access control.
    - Add functionality to process and validate CSV data.

Day 3: Report Generation and Testing

- **Objective**: Finalize reporting features and test the system.
    - Generate and export quality control and defect reports.
    - Debug and test the application with sample data.
    - Prepare a brief project presentation.

Modules and Tasks

1. Class Design

- Define classes for:
    - **Product**: Store product details.
    - **Machine**: Store machine and production details.
    - **QualityLog**: Record quality control results (e.g., pass/fail, defects).

2. GUI Development

- Develop a GUI using tkinter with the following features:
    - Login screen for user authentication.
    - Forms for logging quality inspection results.
    - Buttons to generate reports.

3. Data Validation

- Use the Validator class to ensure:
    - Inspection results are valid (e.g., "Pass" or "Fail").
    - Numeric fields are positive integers.
    - Dates are correctly formatted.

4. Role-Based Access Control

- Restrict functionalities:
    - **Admin**: Full access.
    - **Operator**: Limited access to data entry.

5. Reporting

- Generate:
    - **Quality Control Summary**: Total inspections, pass rate, and fail rate.
    - **Defect Log**: List of defects recorded per machine or product.

Evaluation Criteria

1. Core Functionality (40%)

- Does the system correctly log and validate quality control data?
- Are the reports generated accurately?

2. GUI Implementation (30%)

- Is the GUI user-friendly and functional?

3. Data Validation (20%)

- Are invalid inputs handled gracefully with appropriate error messages?

4. Presentation (10%)

- Is the project presented clearly, with a functional demonstration?

Project Implementation

1. Class Design

Product Class

python

```
class Product:
    def __init__(self, product_id, name, category):
        self.product_id = product_id
        self.name = name
        self.category = category
```

Machine Class

python

```
class Machine:
    def __init__(self, machine_id, equipment_type):
        self.machine_id = machine_id
        self.equipment_type = equipment_type
        self.inspection_logs = []

    def add_quality_log(self, product, result, defects=None):
        log = {"product": product, "result": result,
"defects": defects}
        self.inspection_logs.append(log)
```

Validator Class

python

```
class Validator:
    @staticmethod
    def validate_text(value):
        if isinstance(value, str) and len(value.strip()) > 0:
            return value.strip()
        raise ValueError("Invalid text input.")

    @staticmethod
    def validate_positive_integer(value):
        if isinstance(value, int) and value > 0:
            return value
        raise ValueError("Value must be a positive integer.")

    @staticmethod
    def validate_date(date_str):
        try:
            return datetime.strptime(date_str.strip(), "%Y-%m-
%d")
        except ValueError:
            raise ValueError("Invalid date format. Expected
YYYY-MM-DD.")
```

2. GUI Development

python

```
import tkinter as tk
from tkinter import ttk

class App(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Quality Control and Reporting")
        self.geometry("800x600")
        self.create_widgets()

    def create_widgets(self):
        # Tabs
        tab_control = ttk.Notebook(self)
        inspection_tab = ttk.Frame(tab_control)
        report_tab = ttk.Frame(tab_control)
        tab_control.add(inspection_tab, text="Inspection
Logs")
        tab_control.add(report_tab, text="Reports")
        tab_control.pack(expand=1, fill="both")

        # Inspection Tab
        ttk.Label(inspection_tab, text="Log Quality
Inspection", font=("Arial", 16)).pack(pady=10)
        ttk.Button(inspection_tab, text="Add Inspection
Result").pack(pady=5)
        ttk.Button(inspection_tab, text="View
Logs").pack(pady=5)

        # Report Tab
        ttk.Label(report_tab, text="Generate Reports",
font=("Arial", 16)).pack(pady=10)
```

```
        ttk.Button(report_tab, text="Quality Control
Summary").pack(pady=5)
        ttk.Button(report_tab, text="Defect
Logs").pack(pady=5)


if __name__ == "__main__":
    app = App()
    app.mainloop()
```

3. Report Generation

Quality Control Summary

python

```
def generate_quality_summary(machine_list):
    summary = []
    for machine in machine_list:
        total_logs = len(machine.inspection_logs)
        pass_logs = sum(1 for log in machine.inspection_logs
if log["result"] == "Pass")
        fail_logs = total_logs - pass_logs
        summary.append({
            "Machine ID": machine.machine_id,
            "Total Inspections": total_logs,
            "Pass Rate": f"{(pass_logs / total_logs) *
100:.2f}%" if total_logs > 0 else "N/A",
            "Fail Rate": f"{(fail_logs / total_logs) *
100:.2f}%" if total_logs > 0 else "N/A",
        })
    return summary
```

Example Workflow

Input:

1. Product and machine data entered through the GUI.
2. Inspection results logged with "Pass", "Fail", or defect details.

Output:

1. **Quality Control Summary**:

markdown

```
Machine ID    Total Inspections    Pass Rate    Fail Rate
-----------------------------------------------------------
MC-5673       100                  95.00%       5.00%
```

2. **Defect Log**:

markdown

```
Machine ID    Defect Details
---------------------------------
MC-5673       Scratch, Misalignment
```

Outcome

Participants will create a functional subsystem for quality control and reporting, showcasing their programming and GUI skills. This reduced timeline ensures focus on critical concepts while delivering a practical application.