

ใบงานการทดลอง

เรื่อง การบูรณาการองค์ความรู้และแนวทางปฏิบัติที่ดีในการพัฒนาซอฟต์แวร์

วัตถุประสงค์การทดลอง

เพื่อให้นักศึกษาสามารถ

- บูรณาการความรู้ด้าน Git, OOP, SOLID, และ Component-Based Design ในโปรเจกต์ขนาดเล็กได้
- ฝึกฝนการ Refactor ให้เป็นไปตามหลักการออกแบบที่ดี
- ประยุกต์ใช้แนวคิด Clean Code ได้
- อธิบายบทบาทของ Code Review และ CI/CD ในกระบวนการพัฒนาซอฟต์แวร์ได้

เครื่องมือและอุปกรณ์ที่ใช้ในการทดลอง

- โปรแกรม Thonny หรือ
- โปรแกรมภาษา Python (ควรใช้งานภาษา Python 3.8 ขึ้นไป)
- โปรแกรม Terminal/Command Prompt
- โปรแกรมแก้ไขโค้ด (IDE/Text Editor) เช่น VS Code เป็นต้น
- บัญชี GitHub และ Repository ที่เชื่อมต่อกับ Local (จากครั้งที่ 1)

ขั้นตอนการทดลอง

ในใบงานนี้ นักศึกษาจะได้พัฒนาและปรับปรุงระบบจัดการงาน (Task Management System) เป็นต้น โดยจะได้นำหลักการทั้งหมดที่ผ่านมาประยุกต์ใช้

การทดลองที่ 1 การเตรียมความพร้อมก่อนการทดลอง (หากใช้ Thonny ไม่ต้องทำหัวข้อ 1.1)

1.1 ตรวจสอบสภาพแวดล้อมการทำงานของภาษาไพธอน (Python Environment) โดย

- เปิด Terminal/Command Prompt ขึ้นมาเพื่อใช้พิมพ์คำสั่ง
- พิมพ์ `python --version` เพื่อตรวจสอบรุ่นของภาษา Python โดยผลลัพธ์ที่ได้จะต้องแสดงรุ่นของ Python ขึ้นมา (เช่น Python 3.13.5 เป็นต้น)
- หากไม่มีภาษา Python ให้ติดตั้ง
หมายเหตุ สามารถดาวน์โหลดไฟล์ติดตั้งได้จาก <https://www.python.org/downloads/>

1.2 สร้างไฟล์เดอร์เพื่อใช้เก็บงานของโปรเจกต์สำหรับการทดลองนี้

1) สร้างไฟล์เดอร์ใหม่ เช่น ในที่นี่สมมติให้ตั้งชื่อว่า task_management_system เป็นต้น และเข้าไปในไฟล์เดอร์ซึ่งนักศึกษาทำงานด้วย Git Bash จะใช้วิธีการตั้งต่อไปนี้ (นักศึกษาสามารถใช้วิธีอื่น ๆ ก็ได้)

1.1) เปิด Git Bash ในไฟล์เดอร์ที่นักศึกษาต้องการสร้างโปรเจกต์

1.2) ใช้คำสั่ง mkdir task_management_system เพื่อสร้างไฟล์เดอร์ที่ใช้เก็บโปรเจกต์ที่ต้องการ

1.3) ใช้คำสั่ง cd task_management_system เพื่อเข้าไปในไฟล์เดอร์ที่ได้สร้างขึ้น

1.3 เริ่มต้น Git Repository บนเครื่องของนักศึกษา

1) พิมพ์ git init (ตอนพิมพ์คำสั่งต้องอยู่ในไฟล์เดอร์ชื่อ task_management_system)

2) การทำ git init มีประโยชน์อย่างไร

กู้ง Git Repository ณ Folder ใดๆ ก็ตามที่ต้องการ ผ่าน Git Bash

3) พิมพ์ git branch -m master main เพื่อเปลี่ยน Branch ชื่อ master เป็น Branch ชื่อ main

4) พิมพ์ git status เพื่อตรวจสอบสถานะต่าง ๆ

5) ผลลัพธ์ที่ได้เป็นอย่างไร (ควรจะต้องมี Branch ชื่อ main เพียงอย่างเดียว)

On branch main

1.4 สร้าง Remote Repository บน GitHub

1) ไปที่ GitHub.com และสร้าง Repository ใหม่ชื่อ task_management_system โดยให้เลือกสร้าง Repository เป็นแบบ Public แต่ห้ามตີກ Add README และ Add gitignore เด็ดขาด

2) คัดลอก URL ของ Repository ที่สร้างขึ้นบน GitHub ไว้

3) เชื่อมต่อ Local Repo กับ Remote Repo โดยการใช้คำสั่งต่อไปนี้บน Git Bash ที่อยู่ในไฟล์เดอร์โปรเจกต์บนเครื่องของนักศึกษา

git remote add origin [URL ที่นักศึกษาคัดลอกมาจาก GitHub]

4) ทำไม่ต้องเชื่อมต่อ Local Repository เข้ากับ Remote Repository ตั้งแต่เริ่มต้นการทำงาน

ก้าวที่ 4 นำงานเข้ามาในติดตามกระบวนการ

1.5. สร้างไฟล์ README.md และการ Commit

- 1) สร้างไฟล์ README.md และเพิ่มข้อความ "# Task Management System" เข้าไปในไฟล์
- 2) พิมพ์ git add README.md เพื่อเพิ่มไฟล์เข้าไปใน Staging Area
- 3) พิมพ์ git commit -m "Initial commit: Setup project and add README"
- 4) พิมพ์ git push -u origin main
- 5) ผลลัพธ์ที่ได้บน GitHub เป็นอย่างไร

ไฟล์ README.md ของบน branch main ดูอย่างไร # Task Management System

การทดลองที่ 2 การเขียนโค้ดแบบ Procedural และทำการ Refactor ให้เป็น OOP

2.1 สร้างไฟล์ procedural_tasks.py

- 1) สร้างไฟล์ชื่อ procedural_tasks.py ในโฟลเดอร์ task_management_system
- 2) พิมพ์โค้ดต่อไปนี้

```
# procedural_tasks.py

tasks = [] # Global list to store tasks

def add_task(description, due_date=None):
    task = {"id": len(tasks) + 1, "description": description, "due_date": due_date, "completed": False}
    tasks.append(task)
    print(f"Task '{description}' added.")
    return task

def list_tasks():
    print("\n--- Current Tasks ---")
    if not tasks:
        print("No tasks available.")
        return
    for task in tasks:
        status = "✓" if task["completed"] else " "
        due = f"(Due: {task['due_date']})" if task["due_date"] else ""
        print(f"[{status}] {task['id']}. {task['description']} {due}")
    print("-----")

def mark_task_completed(task_id):
    for task in tasks:
        if task["id"] == task_id:
            task["completed"] = True
            print(f"Task {task_id} marked as completed.")
            return True
    print(f"Task {task_id} not found.")
    return False

def save_tasks_to_file(filename="tasks.txt"):
    with open(filename, "w") as f:
        for task in tasks:
            f.write(f"{task['id']},{task['description']},{task['due_date']},{task['completed']}\n")
```

```

print(f"Tasks saved to {filename}")

# --- Main Program Logic ---
if __name__ == "__main__":
    add_task("Learn Git", "2024-08-01")
    add_task("Practice OOP", "2024-08-05")
    list_tasks()
    mark_task_completed(1)
    list_tasks()
    save_tasks_to_file()

```

หมายเหตุ นักศึกษาอาจคัดลอกเครื่องหมาย “✓” แล้วนำไปวางในโปรแกรมของนักศึกษาได้ ถ้านักศึกษาไม่ทราบว่าจะพิมพ์เครื่องหมายนี้ได้อย่างไร หรือจะใช้ "\u2713" ก็ได้

3) รันโค้ด python procedural_tasks.py เพื่อทำความเข้าใจการทำงาน โดยตอบคำถามต่อไปนี้

3.1) โค้ดนี้มีปัญหาอะไรบ้างในเรื่องของ Global data และ พักรชันที่ทำหน้าที่หลายอย่าง

ຫຼັງຈາກນີ້ຈະມີຄວາມສົບສອງຂອງການນຳໃຊ້ກົດລົງທຶນ

ນຳໃຊ້ກົດລົງທຶນ ກົດລົງທຶນ ກົດລົງທຶນ

3.2) หากนักศึกษาต้องการเพิ่มคุณสมบัติใหม่ ชื่อ priority ให้กับ Task จะต้องแก้ไขพักรชันใดบ้าง

ນຳໃຊ້ກົດລົງທຶນກ່ອນກົດລົງທຶນກ່ອນກົດລົງທຶນ

2.2 การ Refactor ให้เป็น OOP จะสามารถทำได้โดยการสร้าง Class Task และ TaskManager ดังนี้

1) สร้างไฟล์ใหม่ชื่อ oop_tasks.py

2) สร้าง Class Task โดยการพิมพ์โค้ดดังต่อไปนี้

```

# oop_tasks.py

class Task:
    def __init__(self, task_id, description, due_date=None, completed=False):
        self.id = task_id
        self.description = description
        self.due_date = due_date
        self.completed = completed

    def mark_completed(self):
        self.completed = True
        print(f"Task {self.id} '{self.description}' marked as completed.")

    def __str__(self):
        status = "✓" if self.completed else " "
        due = f"(Due: {self.due_date})" if self.due_date else ""
        return f"[{status}] {self.id}. {self.description}{due}"

```

3) สร้าง Class TaskManager โดยพิมพ์คัดต่อไปนี้ ลงในไฟล์ oop_task.py

```
# oop_tasks.py (ต่อจาก Class Task)

class TaskManager:
    def __init__(self):
        self.tasks = []
        self.next_id = 1

    def add_task(self, description, due_date=None):
        task = Task(self.next_id, description, due_date)
        self.tasks.append(task)
        self.next_id += 1
        print(f"Task '{description}' added.")
        return task

    def list_tasks(self):
        print("\n--- Current Tasks ---")
        if not self.tasks:
            print("No tasks available.")
            return
        for task in self.tasks:
            print(task)
        print("-----")

    def get_task_by_id(self, task_id):
        for task in self.tasks:
            if task.id == task_id:
                return task
        return None

    def mark_task_completed(self, task_id):
        task = self.get_task_by_id(task_id)
        if task:
            task.mark_completed()
            return True
        print(f"Task {task_id} not found.")
        return False
```

4) เพิ่ม Logic หลักเพื่อทดสอบ

```
# oop_tasks.py (ต่อจาก Class TaskManager)

if __name__ == "__main__":
    manager = TaskManager()
    manager.add_task("Learn Git", "2024-08-01")
    manager.add_task("Practice OOP", "2024-08-05")
    manager.list_tasks()
    manager.mark_task_completed(1)
    manager.list_tasks()

# Note: Logic สำหรับ Save/Load ยังไม่ครบถ้วน จะเพิ่มในภายหลังโดยใช้หลักการ Single Responsibility Principle (SRP)
```

```

Directory: C:\Users\Acer\OneDrive\Desktop\task_management_system\task_management_system

Mode LastWriteTime Length Name
-a-- 1 8/21/2025 4:03 PM 1948 oop_tasks.py
-a-- 1 8/21/2025 3:48 PM 1427 procedural_tasks.py
-a-- 1 8/21/2025 3:42 PM 24 README.md
[ ] 1. Learn Git (Due: 2024-08-01)
[ ] 2. Practice OOP (Due: 2024-08-05)
Task 1 'Learn git' marked as completed.

--- Current Tasks ---
[✓] 1. Learn Git (Due: 2024-08-01)
[ ] 2. Practice OOP (Due: 2024-08-05)

PS C:\Users\Acer\OneDrive\Desktop\task_management_system\task_management_system>

```

6) ตอบคำถามต่อไปนี้

6.1) การใช้ Class Task และ TaskManager ช่วยให้โค้ดมีความเป็นระเบียบมากขึ้นอย่างไร

Class Task = จัดการข้อมูลของหน้าที่งาน

TaskManager = ดำเนินการทุกชนิดของ Task เท็จ, ॥สอด, ทำได้ในคลาสของ

6.2) การจัดการ tasks อย่างไรไปอยู่ที่ส่วนไหน

การเพิ่มงาน - TaskManager.add_task()

ตรวจสอบงาน - TaskManager.list_task()

ลงร่องเสร็จ - TaskManager.mark_task_completed()

6.3) การเพิ่มคุณสมบัติใหม่ให้กับ Task จะง่ายขึ้นหรือไม่ อย่างไร

ง่ายขึ้น เพราะ ทุกอย่างมีตัวอย่าง Task ถูกเก็บใน class

7) ทำการ Commit การเปลี่ยนแปลงโดย

7.1) พิมพ์ git add oop_tasks.py

7.2) พิมพ์ git commit -m "feat: Refactor task management to OOP with Task and TaskManager classes"

7.3) พิมพ์ git push origin main

7.4) ผลลัพธ์บน GitHub เป็นอย่างไร

ไฟล์ oop_tasks.py เพิ่มมา

การทดลองที่ 3 การประยุกต์ใช้ SRP และ OCP

ในการทดลองส่วนนี้ จะปรับปรุง TaskManager ให้เป็นไปตามหลัก SRP และ OCP

3.1 ประยุกต์ใช้ SRP โดยการแยก TaskStorage

1) สร้างไฟล์ใหม่ชื่อ srp_tasks.py

2) คัดลอกโค้ดจากไฟล์ oop_tasks.py มาใส่

3) สร้าง Abstract Class TaskStorage (สำหรับใช้ตามหลักการของ OCP เพื่อนำไปใช้ในอนาคต)

```
# srp_tasks.py (เพิ่มส่วนนี้ที่ด้านบน)
from abc import ABC, abstractmethod

class TaskStorage(ABC):
    @abstractmethod
    def load_tasks(self):
        pass
    @abstractmethod
    def save_tasks(self, tasks):
        pass
```

4) สร้าง Concrete Class ชื่อ FileTaskStorage

```
# srp_tasks.py (ต่อจาก TaskStorage)

class FileTaskStorage(TaskStorage):
    def __init__(self, filename="tasks.txt"):
        self.filename = filename

    def load_tasks(self):
        loaded_tasks = []
        try:
            with open(self.filename, "r") as f:
                for line in f:
                    parts = line.strip().split(',')
                    if len(parts) == 4:
                        task_id = int(parts[0])
                        description = parts[1]
                        due_date = parts[2] if parts[2] != 'None' else None
                        completed = parts[3] == 'True'
                        loaded_tasks.append(Task(task_id, description, due_date, completed))
        except FileNotFoundError:
            print(f"No existing task file '{self.filename}' found. Starting fresh.")
        return loaded_tasks

    def save_tasks(self, tasks):
        with open(self.filename, "w") as f:
            for task in tasks:
                f.write(f"{task.id},{task.description},{task.due_date},{task.completed}\n")
        print(f"Tasks saved to {self.filename}")
```

5) ปรับปรุง TaskManager ให้รับ TaskStorage (หลักการของ Dependency Injection)

```
# srp_tasks.py (ปรับปรุง TaskManager)

class TaskManager:
    def __init__(self, storage: TaskStorage): # รับ storage object เข้ามา
        self.storage = storage
        self.tasks = self.storage.load_tasks()
        self.next_id = max([t.id for t in self.tasks] + [0]) + 1 if self.tasks else 1
        print(f"Loaded {len(self.tasks)} tasks. Next ID: {self.next_id}")

    def add_task(self, description, due_date=None):
```

```

task = Task(self.next_id, description, due_date)
self.tasks.append(task)
self.next_id += 1
self.storage.save_tasks(self.tasks) # Save after adding
print(f"Task '{description}' added.")
return task

# ... (list_tasks, get_task_by_id, mark_task_completed methods เหลือเดิม) ...

def mark_task_completed(self, task_id):
    task = self.get_task_by_id(task_id)
    if task:
        task.mark_completed()
        self.storage.save_tasks(self.tasks) # Save after marking
        return True
    print(f"Task {task_id} not found.")
    return False

```

6) ปรับปรุง Logic หลัก

srp_tasks.py (ปรับปรุง Logic หลัก)

```

if __name__ == "__main__":
    file_storage = FileTaskStorage("my_tasks.txt")
    manager = TaskManager(file_storage) # ส่ง FileTaskStorage เข้าไปเป็นอภิเษนต์

    manager.list_tasks()
    manager.add_task("Review SOLID Principles", "2024-08-10")
    manager.add_task("Prepare for Final Exam", "2024-08-15")
    manager.list_tasks()
    manager.mark_task_completed(1)
    manager.list_tasks()

```

7) รันโค้ดจากไฟล์ srp_tasks.py ผลลัพธ์ที่ได้เป็นอย่างไร

```

ps C:\Users\Acer\OneDrive\Desktop\task_management_system\task_management_system python srp_task
srp
No existing task file 'my_tasks.txt' found. Starting fresh.
Loaded 0 tasks. Next ID: 1
--- Current Tasks ---
No tasks available.
--- Adding Task ---
Task 'Review SOLID Principles' added.
Tasks saved to my_tasks.txt
Task 'Prepare for Final Exam' added.
--- Current Tasks ---
[] 1. Review SOLID Principles (Due: 2024-08-10)
[] 2. Prepare for Final Exam (Due: 2024-08-15)
Task 1 'Review SOLID Principles' marked as completed.
Tasks saved to my_tasks.txt
--- Current Tasks ---
[✓] 1. Review SOLID Principles (Due: 2024-08-10)
[] 2. Prepare for Final Exam (Due: 2024-08-15)

```

8) ตอบคำถามต่อไปนี้

8.1) Class TaskManager และ FileTaskStorage มีหน้าที่รับผิดชอบอะไรบ้าง

class TaskManager - เป็นหน้า ตรวจสอบ, ตั้งแต่วันที่ ID , หักเดือนและวันเวลา

class FileTaskStorage - หน้าที่จะจัดเก็บ Tasks ลงไฟล์ , รับผิดชอบ เก็บข้อมูล Persistent storage ลงไฟล์

8.2) ถ้าหากนักศึกษาต้องการเปลี่ยนแปลงการทำงานให้ Tasks เก็บข้อมูลลงในฐานข้อมูล (แทนการบันทึกเป็นไฟล์) จะต้องแก้ไข Class ใดบ้าง และสอดคล้องกับประโยชน์ของ SRP และ OCP อย่างไร

សង្គម class តើជាសម្រាប់ Task Storage
SRP និង class ដែលអាចត្រួតពេញ

- ## 9) Commit การเปลี่ยนแปลง โดย

- 9.1) พิมพ์ git add srp tasks.py

- 9.2) พิมพ์ git commit -m "refactor: Apply SRP by separating TaskStorage"

- 9.3) พิมพ์ git push origin main

การทดลองที่ 4 การจำลอง Code Quality และ Collaboration

ในการทดลองส่วนนี้ จะเน้นที่การเขียนโค้ดที่สะอาด (Clean Code) และเข้าใจกระบวนการ Code Review และ CI/CD

4.1 แนวปฏิบัติในการทำ Clean Code

- 1) ตรวจสอบโค้ด srp_tasks.py ของนักศึกษาดังต่อไปนี้

- 1.1) ข้อตัวแปรหรือฟังก์ชัน มีความหมายชัดเจนหรือไม่ เช่น task_id, description, add_task, mark_completed เป็นต้น

- 1.2) **Comment** มี Comment ที่อธิบายว่า ทำไม หรืออธิบาย Logic ที่ซับซ้อนหรือไม่ โดยต้องเพิ่ม Comment ในส่วนที่สำคัญเสมอ

- 1.3) Formatting ควรใช้ VS Code (หรือ Text Editor อื่น ๆ) เพื่อจัด Format โค้ดให้สวยงาม

- 1.4) DRY (Don't Repeat Yourself) ตรวจสอบว่ามีโค้ดส่วนใดที่ซ้ำซ้อน และสามารถ Refactor เป็นฟังก์ชัน หรือเมอรอดแยกได้หรือไม่ เช่น การหา Task ด้วย ID เป็นต้น

- 2) ตอบคำถาม การเขียน Clean Code มีผลต่อการทำงานร่วมกันในทีมอย่างไร อธิบาย

ກົດລົບຄົວພາດ ແລະ ສັນຍາ ຖ້ອນໂຮງ Refactor ມາເປັນໄດ້ກຳນົດ ອົງກວມ ຂະໜາງ ດີເລີກ

ເພີ້ມຄວາມຮັງສະເໝົ່າໂລກ ມີສຶກສືການຊອງກຳນົມ ລົດລາຍກົດແລ້ວ

4.2 แนวคิด Testable Code (ไม่ต้องเขียน Test เพื่อทดสอบจริง อาจารย์แค่อธิบาย และให้นักศึกษาตอบคำถาม) การออกแบบ Class Task และ Class TaskManager ให้มีหน้าที่ชัดเจน (High Cohesion) และพึงพาภันน้อย (Low Coupling) จะช่วยให้การเขียน Unit Test ทำได้ง่าย

ตัวอย่างเช่นหากศึกษาต้องการทดสอบ `Task.mark_completed()` ก็เพียงแค่สร้าง Object ของ Class ชื่อ `Task` และเรียก Method นี้ได้เลย โดยไม่จำเป็นต้องมี `TaskManager` หรือ `FileTaskStorage` นักศึกษาคิดว่าที่เป็นเช่นนี้

เพราะเหตุได

เพราะ Task ที่มีหัวข้อที่สำคัญ ก็คือ Task class ซึ่งสามารถอ่านและเขียนข้อมูลในไฟล์ โดยใช้ TaskManager, Storage

4.3 จำลอง Code Review ให้ทำตามขั้นตอนดังนี้

1) สร้าง Branch ใหม่ โดยพิมพ์ git checkout -b feat/add-task-priority

2) เพิ่มฟีเจอร์ดังต่อไปนี้

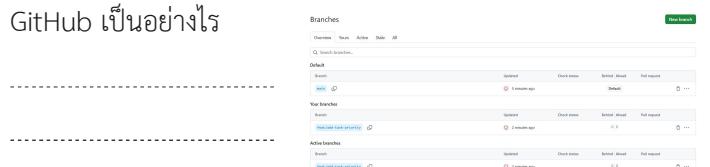
2.1) แก้ไข Class Task ใน srp_tasks.py เพื่อเพิ่ม priority (เช่น low, medium, high) เป็น Attribute

2.2) แก้ไข __init__ และ __str__ ของ Task เพื่อรับและใช้งานค่า priority

2.3) แก้ไข add_task ใน TaskManager เพื่อรับ priority

2.4 ทำการ Commit การเปลี่ยนแปลง โดยพิมพ์ git add srp_tasks.py; git commit -m "feat: Add priority to tasks"

3) ทำการ Push Branch ขึ้น GitHub โดยพิมพ์ git push -u origin feat/add-task-priority ผลลัพธ์ที่ได้บน GitHub เป็นอย่างไร



4) สร้าง Pull Request (PR) บน GitHub โดย

4.1) ไปที่ GitHub Repository ของนักศึกษา

4.2) จะเห็นข้อความให้สร้าง Pull Request จาก Branch feat/add-task-priority ไปยัง main ให้คลิกเพื่อสร้าง

4.3) เขียนคำอธิบาย PR ว่านักศึกษาเพิ่มอะไรเข้าไปบ้าง แล้วกด Create pull request ผลลัพธ์ที่ได้เป็นอย่างไร

branch feat/add-task-priority จะถูกดึงจาก main และมีการ Merge

5) ทำการ Self-Review (การรีวิวด้วยตนเอง) บน GitHub โดย

5.1) ในหน้า PR บน GitHub ไปที่แท็บ Pull requests แล้วเลือก Request ที่ต้องการ แล้วเลือก Files changed

5.2) ให้นักศึกษาลองอ่านโค้ดของตัวนักศึกษาเองและลองให้ Comment เสนอแนะ เช่น ควรเพิ่ม Default priority หรือไม่ หรืออื่นๆ แล้วเลือกที่ Review changes เพื่อ Comment

5.3) การทำ Code Review มีประโยชน์อย่างไรบ้างในการพัฒนาซอฟต์แวร์เป็นทีม

พิจกรรม ทำางานแบบ ทีม โครงการ 001 กษาภิกาพ ลดข้อผิดพลาด ของดานรุ่งในทีม

6) Merge Pull Request ให้ทำหลังจาก Self-Review แล้ว ให้ Merge pull request เพื่อนำโค้ดเข้าสู่ Branch ชื่อ main ผลลัพธ์ที่ได้เป็นอย่างไร

ก็ตจาก Branch ดูหนา รวมกัน main ทีนี้ฟรัตกรหัณฑ์ชุดงาน

7) เมื่อ Merge บน GitHub เรียบร้อยแล้วให้ดึงโค้ดล่าสุดลง Local Repository โดยพิมพ์ git checkout main; git pull origin main ผลลัพธ์ใน Local Repository เป็นอย่างไร

Local Repository ซึ่งมี main แล้ว

การทดลองที่ 5 CI/CD Conceptual Walkthrough

ในการทดลองส่วนนี้ จะดูว้อย่างไฟล์ CI/CD Workflow เพื่อให้เห็นภาพการทำงานอัตโนมัติ

5.1) สร้างไฟล์ .github/workflows/main.yml (โดยยังไม่รัน) สามารถทำได้ดังนี้

1) ในโฟลเดอร์ task_management_system สร้างโฟลเดอร์ .github และสร้างไฟลเดอร์ย่อย workflows ภายใน

2) สร้างไฟล์ชื่อ main.yml ภายในโฟลเดอร์ workflows

3) คัดลอกโค้ดต่อไปนี้ ใส่ลงไปใน main.yml

```
# .github/workflows/main.yml
name: Python CI/CD Pipeline
```

```
on:
push:
  branches:
    - main
pull_request:
  branches:
    - main

jobs:
  build-and-test:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v4
      - name: Set up Python
```

```

uses: actions/setup-python@v5
with:
  python-version: '3.9' # หรือเวอร์ชันที่ใช้

- name: Install dependencies
  run: |
    python -m pip install --upgrade pip
    # pip install -r requirements.txt # สำหรับไฟล์ requirements.txt
    pip install flake8 black # ติดตั้ง linter/formatter

- name: Run Linters (Code Quality Check)
  run: |
    flake8 . --count --select=E9,F63,F7,F82 --show-source --statistics
    # black --check . # สำหรับ black

- name: Run Tests (Conceptual - no actual tests yet)
  run: |
    echo "Running conceptual tests..."
    # python -m unittest discover # สำหรับ Unit Tests จริงๆ
    # pytest # สำหรับ pytest
    echo "All checks passed!"

- name: Display success message
  run: echo "CI/CD Pipeline finished successfully!"

```

4) ตอบคำถามต่อไปนี้

4.1 ไฟล์ main.yml นี้จะถูก Trigger หรือทำงานเมื่อใด

จะถูก Trigger หลัง push หรือ pull request ไปยัง branch (main)

4.2) Pipeline นี้จะทำอะไรบ้าง

ก่อตั้ง ติดตั้ง Py ตรวจสอบหาผิดพลาด ทดสอบ

5) Commit และ Push ไฟล์ CI/CD

5.1) พิมพ์ git add .github/workflows/main.yml

5.2) พิมพ์ git commit -m "ci: Add basic Python CI/CD workflow with linting"

5.3) พิมพ์ git push origin main

5.4) ไปที่แท็บ Actions บน GitHub Repository ของนักศึกษาเพื่อตรวจสอบ โดยนักศึกษาจะเห็น Pipeline เริ่มทำงาน ผลลัพธ์ที่ได้ในรายการที่ Actions เป็นอย่างไร

หมายเหตุ: รายงาน แจ้ง Status

6) ให้นักศึกษาทดลองแก้ไขไฟล์ srp_tasks.py ใน Branch ชื่อ main โดยการเพิ่ม print("Finished")

ต่อท้ายส่วนของ Logic หลัก แล้วทำการ Commit และ Push ขึ้นบน GitHub แล้วสั่งเกตในรายการ Actions ผลลัพธ์ที่ได้ เป็นอย่างไร

github trigger workflow อีกครั้ง 1 ทำการ push ให้ branch (main)

7) ตอบคำถาม ส่วนของ Actions ใน GitHub จะทำงานเมื่อไร
ทำจากหัวข้อมาร์ติ ไฟล์ trigger ในไฟล์ .yml

สรุปผลการทดลอง (ให้นักศึกษาสรุปตามวัตถุประสงค์)

ฝึกการ ศึกษาโปรแกรม เทคนิคที่ใช้ ในการฝึกอบรม รับเรียน เข้าใจจริงๆ ต่อยอดการนำร่องเทคโนโลยี กระบวนการซื้อขาย ห้องเรียนให้ดี ใช้งานง่าย ใช้การทำงานแบบทีม อย่างมีประสิทธิภาพ
เพื่อพัฒนาไปสู่ยั่งยืน การ SRP , OCP

คำตามท้ายการทดลอง

- อธิบายบทบาทของ Code Review และ CI/CD ในกระบวนการพัฒนาซอฟต์แวร์

Code Review = เพื่องห่วงที่นักศึกษาตรวจสอบ ดูงานพิจารณา ให้ความเห็นว่าดีไม่ดี

CI / CD = การทำงานรีทึคชัล ที่มีต่อเนื่อง ทำให้เราสามารถรัน ทดสอบ ได้โดยอัตโนมัติ

CI / CD = การทำงานรีทึคชัล ที่มีต่อเนื่อง ทำให้เราสามารถรัน ทดสอบ ได้โดยอัตโนมัติ

- นักศึกษาคิดว่าการนำแนวทางปฏิบัติเหล่านี้ไปใช้จะช่วยให้นักศึกษาเป็นนักพัฒนาที่มีประสิทธิภาพมากขึ้นในอนาคตได้อย่างไร

- ทำงานร่วมกันทีมได้ดี
- ทีมงานร่วมใจกันเพื่อมร่วมกัน
- ฝึกฝนการทำงานร่วมกัน ให้ครบ
- ส่งผลงานทันเวลา พร้อมกันทุกที

การส่งใบงาน

ให้นักศึกษาส่งแบบฝึกหัดใน Microsoft Team ในช่องทางที่กำหนด โดยส่งเป็นไฟล์ pdf โดยตั้งชื่อว่า “ID_รหัส นักศึกษา_Session07.pdf”

เกณฑ์การให้คะแนนของใบงาน สำหรับคนที่มาเรียนตรงเวลา

ส่งภายในคืนเรียน	ได้คะแนนไม่เกิน 10 คะแนน
ส่งก่อนเที่ยงคืนของวันที่เรียน	ได้คะแนนไม่เกิน 8 คะแนน
ส่งวันก่อนเรียนครั้งถัดไป	ได้คะแนนไม่เกิน 6 คะแนน