รายงานวิชาการประกันคุณภาพซอฟต์แวร์

เรื่องการวัดประสิทธิภาพและการทดสอบ

จัดทำโดย

นายพิทักษ์พงศ์ สุภาพเพ็ชร รหัสนักศึกษา 653380042-6
นางสาวกรกนก พฤทธิพันธุ์ รหัสนักศึกษา 653380187-0
นายจักรภัทร วงษ์ศรีวรรณ รหัสนักศึกษา 653380190-1

เสนอ

ผศ.ดร.ชิตสุธา สุ่มเล็ก

รายงานนี้เป็นส่วนหนึ่งของรายวิชาการประกันคุณภาพซอฟต์แวร์
รหัสวิชา CP353201 ภาคเรียนที่ 1 ปีการศึกษา 2567
สาขาวิทยาการคอมพิวเตอร์
วิทยาลัยวิทยาการคอมพิวเตอร์ มหาวิทยาลัยขอนแก่น

บทคัดย่อ

โครงงานการวัดประสิทธิภาพและการทดสอบ มุ่งเน้นการประเมินประสิทธิภาพและคุณภาพของโค้ดโดยการนำปัญญาประดิษฐ์หลายตัวในการทดสอบ โครงงานนี้ได้นำ Chain of Responsibility Pattern และ State Pattern มาใช้เป็นแนวทางในการทดสอบ การใช้ Chain of Responsibility ช่วยกระจายความรับผิดชอบในกระบวนการพัฒนาและทดสอบผ่านหลาย ๆ ส่วนที่ทำหน้าที่เฉพาะเจาะจง ขณะที่ State Pattern ช่วยในการจัดการสถานะต่าง ๆ ของการพัฒนาโค้ดอย่างมีประสิทธิภาพ โดยมีภาษาไพทอนและภาษาจาวาเป็นภาษาที่ใช้ในการศึกษา

โครงงานนี้วัดผลลัพธ์จากประสิทธิภาพของการทดสอบคุณภาพของโค้ด ทั้งในด้านความเร็วและประสิทธิภาพของโค้ด โดยมีการใช้เครื่องมือปัญญาประดิษฐ์อย่างหลากหลายเพื่อทดสอบและวัดคุณภาพของโค้ด ซึ่งคาดหวังว่าการศึกษาครั้งนี้จะมีส่วนช่วยในการทำให้โค้ดมีคุณภาพที่ดียิ่งขึ้น

คณะผู้จัดทำ

กิตติกรรมประกาศ

รายงานฉบับนี้เป็นส่วนหนึ่งของรายวิชา การประกันคุณภาพซอฟต์แวร์ รหัสวิชา CP353201 ภาคเรียนที่ 1 ปีการศึกษา 2567 สาขาวิทยาการคอมพิวเตอร์ วิทยาลัยวิทยาการคอมพิวเตอร์ มหาวิทยาลัยขอนแก่น

กลุ่มของข้าพเจ้าขอขอบคุณอาจารย์ประจำวิชา ผศ.ดร.ชิตสุธา สุ่มเล็ก ผู้ที่ให้ความรู้ที่เป็นประโยชน์และชี้แนะแนวทางในการทำโครงงานนี้ โดยเฉพาะในเรื่องของการทดสอบซอฟต์แวร์และการประกันคุณภาพซอฟต์แวร์ ทำให้โครงงานนี้ดำเนินไปอย่างราบรื่นและมีประสิทธิภาพ และเพื่อร่วมทีมสำหรับการทำงานร่วมกันอย่างดีเยี่ยม การแบ่งปันความรู้ด้านการเขียนโปรแกรม และการจัดการโครงงานอย่างมีประสิทธิภาพ

คณะผู้จัดทำ

คำนำ

รายงานฉบับนี้เป็นส่วนหนึ่งของวิชา การประกันคุณภาพซอฟต์แวร์ รหัสวิชา CP353201 ภาคเรียนที่ 1 ปีการศึกษา 2567 สาขาวิทยาการคอมพิวเตอร์ วิทยาลัยวิทยาการคอมพิวเตอร์ มหาวิทยาลัยขอนแก่น จัดทำขึ้นเพื่อศึกษาวิธีการประกันคุณภาพซอฟต์แวร์ รวมถึงการใช้ปัญญาประดิษฐ์เพื่อการเรียนรู้ที่เน้นประสิทธิภาพและความถูกต้องผ่านการทดสอบและตรวจสอบคุณภาพ

การประกันคุณภาพซอฟต์แวร์เป็นกระบวนการที่สำคัญในการพัฒนาซอฟต์แวร์ที่มีคุณภาพสูง โครงงานนี้ได้เลือกใช้ Chain of Responsibility และ State Pattern ซึ่งเป็นรูปแบบการออกแบบซอฟต์แวร์ที่ช่วยให้การพัฒนาเป็นระบบและมีความยืดหยุ่นมากขึ้น นอกจากนี้ ยังได้ใช้ภาษา Python และ Java เป็นเครื่องมือหลักในการพัฒนาและทดสอบประสิทธิภาพของระบบ

ข้าพเจ้าหวังว่ารายงานฉบับนี้จะเป็นประโยชน์สำหรับผู้ที่สนใจในการประกันคุณภาพซอฟต์แวร์ และหวังว่าจะสามารถนำไปประยุกต์ใช้ในโครงการหรือการพัฒนาซอฟต์แวร์อื่น ๆ ได้

สุดท้ายนี้ ข้าพเจ้าขอขอบคุณ ผศ.ดร.ชิตสุธา สุ่มเล็ก อาจารย์ประจำวิชา ที่ได้ให้การสั่งสอนและชี้แนะในการจัดทำโครงงานนี้ รวมถึงขอขอบคุณทุกท่านที่มีส่วนร่วมในการสนับสนุนและให้กำลังใจจนรายงานฉบับนี้สำเร็จลุล่วง

ขอแสดงความนับถือ

คณะผู้จัดทำ

สารบัญ

เรื่อง	**หน้า**
บทคัดย่อ	ก
กิตติกรรมประกาศ	ข
คำนำ	ค
สารบัญ	ง
บทที่ 1 บทนำ	1-2
บทที่ 2 เอกสารและทฤษฎีที่เกี่ยวข้อง	3-5
บทที่ 3 วิธีการดำเนินการ	6-8
บทที่ 4 ผลการดำเนินงาน	9-41
บทที่ 5 อภิปราย	40-42
อ้างอิง	43

บทที่ 1
บทนำ

แนวคิด และความสำคัญ

การประกันคุณภาพซอฟต์แวร์ (Software Quality Assurance : SQA) เป็นกระบวนการที่มีเป้าหมายหลักในการตรวจสอบและควบคุมคุณภาพของซอฟต์แวร์ตลอดกระบวนการพัฒนา ตั้งแต่การวางแผน การออกแบบ การพัฒนา ไปจนถึงการทดสอบ โดยการนำแนวคิดนี้มาประยุกต์ใช้ในการพัฒนา จะช่วยลดข้อผิดพลาดในการทำงาน เพิ่มประสิทธิภาพของซอฟต์แวร์ และเพิ่มความน่าเชื่อถือให้กับผู้ใช้งาน

โครงงานนี้ได้ใช้รูปแบบการออกแบบซอฟต์แวร์ที่เรียกว่า Chain of Responsibility Pattern และ State Pattern ซึ่งมีบทบาทสำคัญในการพัฒนาซอฟต์แวร์ที่มีความยืดหยุ่นและจัดการได้ง่าย โดย Chain of Responsibility ช่วยในการกระจายความรับผิดชอบของแต่ละส่วนในระบบ ทำให้การจัดการกระบวนการทำงานเป็นระบบและชัดเจน ขณะที่ State Pattern ช่วยในการจัดการสถานะต่าง ๆ ของระบบ เพื่อให้สามารถทำงานได้อย่างต่อเนื่องและปรับตัวได้ตามสถานการณ์

การใช้เครื่องมือปัญญาประดิษฐ์ (Artificial Intelligence : AI) ต่าง ๆ ในการศึกษาและวิเคราะห์คุณภาพของโค้ดเป็นปัจจัยที่ช่วยยกระดับกระบวนการประกันคุณภาพซอฟต์แวร์ให้มีความแม่นยำและประสิทธิภาพสูงขึ้น

ความสำคัญของโครงงานนี้ อยู่ที่การทดสอบและประเมินประสิทธิภาพของโค้ดโดยใช้เครื่องมือที่ทันสมัย คือ ปัญญาประดิษฐ์ในการสร้างโค้ดโดยใช้คำสั่งเพื่อสั่งการ พร้อมกับการประยุกต์ใช้หลักการออกแบบที่เหมาะสม เพื่อศึกษาการทำงานของ AI ความเร็วในการประมวลผล ความถูกต้องของโค้ด โดยใช้ AI หลากหลายรูปแบบ ๆ ละสามรอบ อีกทั้งยังวาดไดอะแกรมเพื่อดูลำดับการทำงานของโค้ดที่ได้มา

วัตถุประสงค์

1. เพื่อศึกษาและประยุกต์ใช้ Chain of Responsibility Pattern และ State Pattern ในการสร้างโค้ดด้วย AI หลากหลายรูปแบบเพื่อทดสอบประสิทธิภาพและวัดคุณภาพของโค้ด
2. เพื่อศึกษาแนวทางการประกันคุณภาพซอฟต์แวร์ (SQA) และนำไปประยุกต์ใช้ในกระบวนการพัฒนาซอฟต์แวร์ต่อไปในอนาคต
3. เพื่อประเมินประสิทธิภาพและคุณภาพของโค้ดที่ได้จากการประมวลของ AI โดยใช้เครื่องมือในการช่วยทดสอบและวิเคราะห์ผลลัพธ์

ขั้นตอนการดำเนินงาน

1. ศึกษาการสร้างพร้อมท์เพื่อสั่งการ AI ให้ได้ผลลัพธ์ที่ต้องการ
2. ออกแบบพร้อมท์ให้ตรงตามความต้องการ
3. สั่งการ AI เพื่อประมวลผลพร้อมทั้งจัดเก็บโค้ดที่ได้มาอย่างเป็นระเบียบ
4. ทดสอบและประเมินคุณภาพของโค้ดโดยใช้เครื่องมือ
5. วิเคราะห์และทำการบันทึกผลลัพธ์ลงในเทมเพลต
6. จัดทำรายงานและสไลด์ในการนำเสนอ

7. นำเสนอผลงาน

แผนการปฏิบัติงาน

ขั้นตอนการดำเนินงาน	กิจกรรม	ระยะเวลา
1. ศึกษาการสร้างพร้อมท์เพื่อสั่งการ AI	- ศึกษาแนวทางการสร้างพร้อมท์ - ทดลองใช้พร้อมท์ต่าง ๆ	เดือนสิงหาคม
2. ออกแบบพร้อมท์ให้ตรงตามความต้องการ	- ออกแบบและปรับปรุงพร้อมท์ - ตรวจสอบความเหมาะสมของพร้อมท์	เดือนสิงหาคม
3. สั่งการ AI เพื่อประมวลผล	- สั่งการ AI โดยใช้พร้อมท์ที่ออกแบบ - จัดเก็บโค้ดที่ได้อย่างเป็นระเบียบ	เดือนสิงหาคม
4. ทดสอบและประเมินคุณภาพของโค้ด	- ใช้เครื่องมือในการทดสอบโค้ด - ประเมินผลการทำงานของโค้ด	เดือนกันยายน
5. วิเคราะห์และทำการบันทึกผลลัพธ์ลงในเทมเพลต	- วิเคราะห์ผลลัพธ์จากการทดสอบ - บันทึกผลลัพธ์ลงในเทมเพลต	เดือนกันยายน
6. จัดทำรายงานและสไลด์ในการนำเสนอ	- จัดทำเอกสารรายงาน - สร้างสไลด์สำหรับการนำเสนอ	เดือนกันยายน
7. นำเสนอผลงาน	- เตรียมการนำเสนอ - นำเสนอผลงานต่อคณะกรรมการ	เดือนตุลาคม วันสุดท้ายของการเรียนการสอน

ประโยชน์ที่คาดว่าจะได้รับ

1. การเข้าใจแนวทางการประกันคุณภาพซอฟต์แวร์ (SQA): การศึกษาวิธีการและแนวทางในการประกันคุณภาพซอฟต์แวร์จะช่วยให้มีความเข้าใจในหลักการที่สำคัญในการควบคุมคุณภาพซอฟต์แวร์ ส่งผลให้สามารถนำไปประยุกต์ใช้ในโครงการในอนาคตได้อย่างมีประสิทธิภาพ
2. การพัฒนาซอฟต์แวร์ที่มีคุณภาพสูงขึ้น: การศึกษาและประยุกต์ใช้ Chain of Responsibility Pattern และ State Pattern จะช่วยให้กระบวนการพัฒนาโค้ดมีความเป็นระบบมากขึ้น ทำให้โค้ดที่ผลิตออกมามีความชัดเจน สามารถปรับปรุงและดูแลรักษาได้ง่าย ส่งผลให้คุณภาพของซอฟต์แวร์ดีขึ้น
3. การประเมินและวิเคราะห์คุณภาพของโค้ด: การประเมินประสิทธิภาพและคุณภาพของโค้ดที่ได้จาก AI จะช่วยให้สามารถระบุปัญหาและข้อบกพร่องในโค้ดได้รวดเร็วขึ้น รวมถึงการใช้เครื่องมือในการช่วยทดสอบจะเพิ่มความน่าเชื่อถือของผลลัพธ์ที่ได้

บทที่ 2
เอกสารที่เกี่ยวข้อง

โครงงานเรื่องการวัดประสิทธิภาพและการทดสอบจัดทำขึ้นเพื่อศึกษาและประยุกต์ใช้เทคนิคการออกแบบซอฟต์แวร์ในกระบวนการพัฒนาโค้ด โดยเน้นการใช้ AI ในการสร้างโค้ดที่มีประสิทธิภาพสูงและคุณภาพที่ดีประกอบวิชาการประกับคุณภาพซอฟต์แวร์ โดยมีเนื้อหาเกี่ยวกับการวิเคราะห์และออกแบบซอฟต์แวร์ การประกันคุณภาพซอฟต์แวร์ (SQA) การสร้างโค้ดด้วย AI การทดสอบประสิทธิภาพของโค้ด และการวิเคราะห์ผลลัพธ์ที่ได้จากการใช้ AI มีหลักการและทฤษฎีที่เกี่ยวข้องดังนี้

1 หลักการที่เกี่ยวข้อง

1.1 การออกแบบซอฟต์แวร์ (Software Design Principles)

แนวทางหรือหลักการที่ช่วยให้นักพัฒนาสามารถสร้างซอฟต์แวร์ที่มีคุณภาพสูง มีความยืดหยุ่น และสามารถบำรุงรักษาได้ง่าย โดยมีเป้าหมายเพื่อให้โค้ดมีโครงสร้างที่ดีและสามารถปรับปรุงหรือขยายได้ในอนาคต หลักการเหล่านี้มักจะช่วยในการลดความซับซ้อน และทำให้โค้ดมีความเข้าใจง่ายขึ้น โดยมีหลักการดังนี้

1.1.1 Single Responsibility Principle (SRP) คลาสหรือโมดูลควรมีความรับผิดชอบเพียงหนึ่งอย่าง เพื่อให้โค้ดง่ายต่อการเข้าใจและบำรุงรักษา

1.1.2 Open/Closed Principle (OCP) โค้ดควรเปิดสำหรับการขยาย แต่ปิดสำหรับการแก้ไข ซึ่งช่วยให้สามารถเพิ่มฟีเจอร์ใหม่ได้โดยไม่ต้องเปลี่ยนแปลงโค้ดเดิม

1.1.3 Liskov Substitution Principle (LSP) วัตถุของคลาสลูกควรสามารถใช้แทนที่วัตถุของคลาสพ่อแม่ได้ โดยไม่ทำให้พฤติกรรมของระบบเปลี่ยนแปลง

1.1.4 Interface Segregation Principle (ISP) ควรใช้หลาย ๆ อินเทอร์เฟซที่เฉพาะเจาะจงแทนที่จะใช้หนึ่งอินเทอร์เฟซที่ใหญ่และไม่จำเป็น

1.1.5 Dependency Inversion Principle (DIP) โมดูลระดับสูงไม่ควรขึ้นอยู่กับโมดูลระดับต่ำ แต่ควรขึ้นอยู่กับ abstractions แทน

1.2 การประกันคุณภาพซอฟต์แวร์ (Software Quality Assurance : SQA)

กระบวนการที่มุ่งเน้นการตรวจสอบและประกันว่าการพัฒนาซอฟต์แวร์มีคุณภาพตามมาตรฐานที่กำหนด โดย SQA รวมถึงกิจกรรมต่าง ๆ ที่ช่วยในการวางแผน ตรวจสอบ และประเมินคุณภาพซอฟต์แวร์ในทุกขั้นตอนของการพัฒนา เพื่อให้มั่นใจว่าโปรแกรมที่ผลิตออกมานั้นตรงตามความต้องการของผู้ใช้งานและมีคุณภาพที่สูง โดยมีหลักการดังนี้

1.2.1 Quality Planning กระบวนการในการกำหนดวิธีการและมาตรฐานที่ใช้ในการควบคุมและประกันคุณภาพในกระบวนการพัฒนาซอฟต์แวร์ โดยมีเป้าหมายเพื่อให้แน่ใจว่าผลิตภัณฑ์ซอฟต์แวร์ที่พัฒนาขึ้นมีคุณภาพสูงและตอบสนองความต้องการของผู้ใช้ ในการวางแผนคุณภาพ

1.2.2 Review and Testing กระบวนการสำคัญใน Software Quality Assurance (SQA) ที่มุ่งเน้นในการค้นหาข้อผิดพลาดและประเมินคุณภาพของซอฟต์แวร์ในแต่ละขั้นตอนของการพัฒนา โดยประกอบไปด้วยสองส่วนหลัก คือ การตรวจสอบและการทดสอบ

1.2.3 Defect Management กระบวนการที่เกี่ยวข้องกับการติดตาม การวิเคราะห์ การจัดการ และการแก้ไขข้อผิดพลาด (defects) ที่เกิดขึ้นในซอฟต์แวร์ระหว่างการพัฒนาและหลังการใช้งาน โดยมีจุดมุ่งหมายเพื่อให้มั่นใจว่าข้อผิดพลาดจะถูกจัดการอย่างมีประสิทธิภาพ และซอฟต์แวร์ที่ผลิตออกมามีคุณภาพสูง

1.2.4 Change Control กระบวนการที่ใช้ในการจัดการกับการเปลี่ยนแปลงที่เกิดขึ้นในโครงการซอฟต์แวร์ เพื่อให้แน่ใจว่าการเปลี่ยนแปลงเหล่านั้นจะถูกจัดการอย่างมีระบบ และไม่ส่งผลกระทบต่อคุณภาพของซอฟต์แวร์หรือความสามารถในการให้บริการ

2.ทฤษฎีที่เกี่ยวข้อง

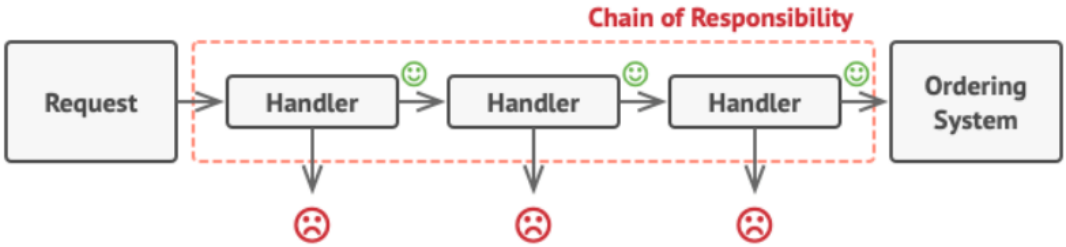
2.1 Chain of Responsibility Pattern

เป็นรูปแบบการออกแบบซอฟต์แวร์ (design pattern) ที่ใช้ในการจัดการกับคำร้องขอ (request) โดยการส่งคำร้องขอผ่านสายโซ่ของผู้จัดการหรือ handler ที่สามารถจัดการคำร้องขอนั้นได้ รูปแบบนี้ช่วยให้การจัดการคำร้องขอเป็นไปอย่างมีระเบียบและยืดหยุ่น โดยไม่จำเป็นต้องระบุให้ชัดเจนว่าผู้จัดการใดจะต้องจัดการคำร้องขอนั้น

หลักการทำงานของ Chain of Responsibility ประกอบด้วยการกำหนดอินเทอร์เฟซสำหรับผู้จัดการ ซึ่งมีเมธอดในการจัดการคำร้องขอ และอาจมีเมธอดในการกำหนดผู้จัดการถัดไปในสายโซ่ โดยจะสร้างผู้จัดการที่เป็นจริง (concrete handler) ซึ่งจะจัดการคำร้องขอที่เหมาะสม หากผู้จัดการไม่สามารถจัดการได้ จะส่งต่อคำร้องขอไปยังผู้จัดการถัดไปในสายโซ่ ในส่วนของ client จะเป็นผู้ส่งคำร้องขอไปยังสายโซ่ของผู้จัดการ โดยไม่ต้องรู้จักลำดับของผู้จัดการในสายโซ่

การใช้ Chain of Responsibility จะมีประโยชน์ในเรื่องของความยืดหยุ่นที่สามารถเพิ่มหรือลบผู้จัดการในสายโซ่ได้อย่างง่ายดาย โดยไม่กระทบกับโค้ดที่ใช้งาน นอกจากนี้ยังช่วยลดการเชื่อมโยง (coupling) ระหว่างผู้จัดการที่ทำงานร่วมกัน สามารถจัดการคำร้องขอหลายประเภทได้โดยการสร้างผู้จัดการที่แตกต่างกันในสายโซ่

ตัวอย่างการใช้งาน Chain of Responsibility ได้แก่ การจัดการอีเมลที่เข้ามาในระบบ เช่น อีเมลทั่วไป สแปม หรืออีเมลด่วน และการประมวลผลคำร้องขอในแอปพลิเคชัน ซึ่งสามารถใช้ได้กับระบบที่ต้องการการจัดการคำร้องขออย่างมีประสิทธิภาพและความยืดหยุ่นในการบำรุงรักษาโค้ด



2.2 State Pattern

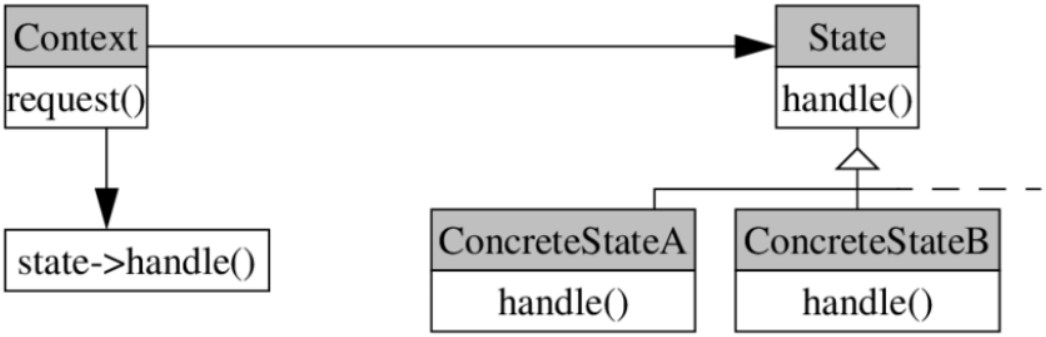
เป็นรูปแบบการออกแบบซอฟต์แวร์ (design pattern) ที่ใช้ในการจัดการสถานะต่างๆ ของวัตถุ โดยช่วยให้วัตถุสามารถเปลี่ยนพฤติกรรมได้ตามสถานะที่อยู่ในขณะนั้น รูปแบบนี้ช่วยให้การจัดการกับสถานะมีความชัดเจนและมีความยืดหยุ่นมากขึ้น

ใน State Pattern วัตถุจะมีการแยกสถานะออกเป็นคลาสต่างๆ ซึ่งแต่ละสถานะจะมีการดำเนินการและพฤติกรรมที่เฉพาะเจาะจงสำหรับสถานะนั้นๆ เมื่อสถานะของวัตถุเปลี่ยนไป วัตถุสามารถเปลี่ยนการทำงานของมันไปยังสถานะใหม่ได้โดยไม่ต้องแก้ไขโค้ดที่เกี่ยวข้องกับสถานะก่อนหน้า

หลักการทำงานของ State Pattern คือ การสร้างอินเทอร์เฟซที่กำหนดการทำงานของสถานะ ซึ่งจะถูกนำไปใช้งานในแต่ละคลาสของสถานะที่เฉพาะเจาะจง โดยวัตถุหลักจะเก็บการอ้างอิงไปยังสถานะปัจจุบัน และเมื่อมีการเปลี่ยนแปลงสถานะ จะมีการอัพเดตการอ้างอิงไปยังสถานะใหม่

State Pattern ช่วยให้โค้ดมีความชัดเจนและเข้าใจง่ายขึ้น เนื่องจากพฤติกรรมของวัตถุจะถูกจัดการแยกจากกันตามสถานะ นอกจากนี้ยังช่วยลดความซับซ้อนในโค้ดโดยการเลี่ยงการใช้โครงสร้างเงื่อนไข (if-else statements) ที่ซับซ้อน

ตัวอย่างการใช้งาน State Pattern อาจพบในเกมคอมพิวเตอร์ที่ตัวละครมีสถานะต่างๆ เช่น วิ่ง, เดิน, หรือโจมตี โดยตัวละครจะเปลี่ยนพฤติกรรมและการกระทำไปตามสถานะที่อยู่ในขณะนั้น การใช้ State Pattern จะทำให้การจัดการสถานะและพฤติกรรมของตัวละครมีความยืดหยุ่นและง่ายต่อการบำรุงรักษา



บทที่ 3
วิธีดำเนินการ

ในบทนี้จะนำเสนอวิธีดำเนินการของโครงการ ซึ่งประกอบด้วยขั้นตอนและเทคนิคที่ใช้ในการพัฒนาระบบ AI-Assisted Coding and Testing โดยมุ่งเน้นที่การประยุกต์ใช้ Chain of Responsibility Pattern และ State Pattern เพื่อประเมินประสิทธิภาพและคุณภาพของโค้ดที่สร้างขึ้น

1.เครื่องมือ

1.1 Design Pattern

1.1.1 Chain of Responsibility Pattern

1.1.2 State Pattern

1.2 ภาษา (Programming language)

1.2.1 ไพธอน (Python)

1.2.2 จาวา (Java)

1.3 Unit testing

1.3.1 Pytest

1.3.2 JUnit

1.3.3 Statement coverage

1.3.4 Branch coverage

2. ขั้นตอนการดำเนินการ

ขั้นตอนการดำเนินการโครงการนี้แบ่งออกเป็นหลายส่วน โดยมีรายละเอียดและการบรรยายในแต่ละหัวข้อดังนี้:

1.1 การศึกษาการสร้างพร้อมท์ (Prompt)เพื่อสั่งการ AI

ในขั้นตอนแรก จะทำการศึกษาและวิเคราะห์วิธีการสร้างพร้อมท์ที่มีประสิทธิภาพเพื่อสั่งการ AI ให้ได้ผลลัพธ์ที่ต้องการ การสร้างพร้อมท์ที่ดีนั้นมีความสำคัญอย่างมาก เพราะมันจะส่งผลต่อความสามารถในการสร้างโค้ดที่มีคุณภาพและตรงตามความต้องการ โดยรวบรวมข้อมูลจากแหล่งข้อมูลที่มีอยู่ เช่น เอกสารและบทความที่เกี่ยวข้อง

1.2 การออกแบบพร้อมท์ให้ตรงตามความต้องการ

เมื่อมีการศึกษาเสร็จสิ้น จะดำเนินการออกแบบพร้อมท์ที่เหมาะสมกับ AI แต่ละตัวและความต้องการของจุดประสงค์โครงงาน การออกแบบนี้จะพิจารณาถึงรูปแบบของคำสั่งและผลลัพธ์ที่คาดหวัง โดยจะมีการระบุสิ่งที่ต้องการให้ AI สร้างโค้ดให้ จากนั้นจะทำการสร้างแบบร่างของพร้อมท์ที่สามารถใช้กับ AI ได้อย่างมีประสิทธิภาพ ตัวอย่างเช่น

1. ใช้ Chain of Responsibility pattern เพื่อเขียนโค้ดในภาษา Python ที่เกี่ยวข้องกับการขนส่งพัสดุ โดยมีข้อกำหนดดังนี้: Code.py: สร้าง chain of responsibility สำหรับการขนส่งพัสดุที่มี handler ที่แตกต่างกัน เช่น:
LocationHandler: ตรวจสอบว่าพัสดุสามารถส่งไปยังสถานที่ที่กำหนดได้หรือไม่
PaymentHandler: จัดการการชำระเงินสำหรับการส่งพัสดุ

DeliveryHandler: จัดการการส่งพัสดุจริง หาก handler ไม่สามารถประมวลผลคำขอได้ ควรส่งคำขอไปยัง handler ถัดไปใน chain หากหนึ่งใน handler ประมวลผลคำขอสำเร็จ chain ควรหยุด

Test.py: เขียน unit tests โดยใช้ Pytest เพื่อให้แน่ใจว่ามี 100% branch coverage ของโค้ด chain of responsibility การทดสอบควรตรวจสอบว่า handler แต่ละตัวประมวลผลคำขออย่างไรและคำขอถูกส่งต่อไปยัง chain เมื่อ handler ไม่สามารถประมวลผลได้

2. ใช้ State pattern ในการเขียนโค้ดในภาษา Python ตามความต้องการด้านล่างนี้:

เขียนไฟล์ Code.py เพื่อสร้างคลาสสำหรับการควบคุมการเปิดปิดไฟ โดยมีสถานะต่าง ๆ เช่น เปิด (On) และปิด (Off) แต่ละสถานะจะต้องกำหนดพฤติกรรมของไฟสำหรับการเปิด (turn_on) และปิด (turn_off)

เขียนไฟล์ Test.py เพื่อตรวจสอบโค้ดที่เขียน โดยต้องมีการทดสอบครอบคลุม 100% ทุก branch ของโค้ด เพื่อยืนยันพฤติกรรมของไฟในแต่ละสถานะและการเปลี่ยนสถานะที่ถูกต้อง

การออกแบบ Prompt มีความแตกต่างกันบ้างตามแต่ละเอไอเนื่องจากการกระจายงานของสมาชิกภายในกลุ่ม หากแต่มีจุดมุ่งหมายในการสร้าง Prompt เดียวกันให้ครบตามโจทย์ที่กำหนด เพื่อบรรลุวัตถุประสงค์ของโครงงานนี้

1.3 การสั่งการ AI เพื่อประมวลผล

ทำการสั่งการ AI โดยใช้พร้อมท์ที่ออกแบบขึ้น เพื่อประมวลผลโค้ดที่ต้องการ การสั่งการนี้จะถูกทำอย่างมีระบบ โดยจะให้คำสั่งที่ชัดเจนและมีโครงสร้าง เพื่อให้ AI สามารถเข้าใจและดำเนินการได้อย่างถูกต้อง เมื่อได้ผลลัพธ์ที่เป็นโค้ดที่สร้างขึ้นแล้ว จะทำการจัดเก็บโค้ดที่ได้มาอย่างเป็นระเบียบ โดยใช้ Github เพื่อให้สามารถทำงานร่วมกันภายในกลุ่มและติดตามงานอย่างมีประสิทธิภาพ

1.4 การทดสอบและประเมินคุณภาพของโค้ด

เมื่อได้โค้ดที่สร้างขึ้นแล้ว จะทำการทดสอบและประเมินคุณภาพของโค้ดโดยใช้เครื่องมือที่เหมาะสม เช่น การทดสอบหน่วย (unit testing) และการตรวจสอบคุณภาพของโค้ด เครื่องมือเหล่านี้จะช่วยให้สามารถตรวจสอบว่าโค้ดทำงานรูปแบบใดหลังจากการประมวลของ AI ทั้งสามรอบ

1.5 การวิเคราะห์และทำการบันทึกผลลัพธ์

ผลลัพธ์ที่ได้จากการทดสอบจะถูกวิเคราะห์เพื่อทำความเข้าใจในด้านประสิทธิภาพและคุณภาพของโค้ด จะพิจารณาถึงข้อแตกต่างของโค้ดในแต่ละรอบที่สร้างขึ้น การวิเคราะห์จะรวมถึงการประเมินว่าทำไมโค้ดบางส่วนทำงานได้ดีในขณะที่บางส่วนอาจมีปัญหา จะทำการบันทึกผลลัพธ์ลงในเทมเพลตเพื่อให้เป็นระเบียบและช่วยสรุปผลข้อมูลเหล่านี้อย่างชัดเจนและเข้าใจง่าย

1.6 การจัดทำรายงานและสไลด์ในการนำเสนอ

หลังจากวิเคราะห์และบันทึกผลลัพธ์แล้ว จะจัดทำรายงานและสไลด์สำหรับการนำเสนอผลงาน ซึ่งจะรวมถึงการแสดงผลลัพธ์และการอธิบายวิธีการดำเนินการทั้งหมด รายงานนี้จะเป็นเครื่องมือที่สำคัญในการสื่อสารข้อมูลที่สำคัญและแสดงให้เห็นถึงความสำเร็จของโครงงาน

การจัดทำรายงานจะเน้นไปที่ข้อมูลที่สำคัญ เช่น วัตถุประสงค์ วิธีการดำเนินการ ผลลัพธ์ที่ได้ และข้อเสนอแนะสำหรับการทำโครงงานในครั้งนี้ สไลด์จะถูกออกแบบให้มีความกระชับและเข้าใจง่าย เพื่อให้การนำเสนอเป็นไปอย่างราบรื่น

1.7 การนำเสนอผลงาน

ขั้นตอนสุดท้ายคือการนำเสนอผลงานต่ออาจารย์ประจำวิชาและเพื่อนร่วมชั้นเรียน ซึ่งจะรวมถึงการแสดงผลลัพธ์และการอธิบายวิธีการดำเนินการทั้งหมด การนำเสนอจะเป็นโอกาสได้แสดงถึงความรู้และความเข้าใจในกระบวนการที่ดำเนินการ

ทางคณะจะเตรียมตัวอย่างชัดเจน เพื่อให้สามารถตอบคำถามจากผู้ฟังได้อย่างมั่นใจ และเพื่อให้ผู้ฟังเข้าใจถึงคุณค่าของผลงานที่ได้ทำ การนำเสนอผลงานนี้จึงมีความสำคัญต่อการสร้างความเข้าใจและการสื่อสารถึงความสำเร็จของโครงงาน

ในส่วนนี้จะเห็นได้ว่าแต่ละขั้นตอนมีความสำคัญในการสร้างผลงานที่มีคุณภาพและสามารถตอบสนองต่อวัตถุประสงค์ของโครงงานได้อย่างมีประสิทธิภาพ

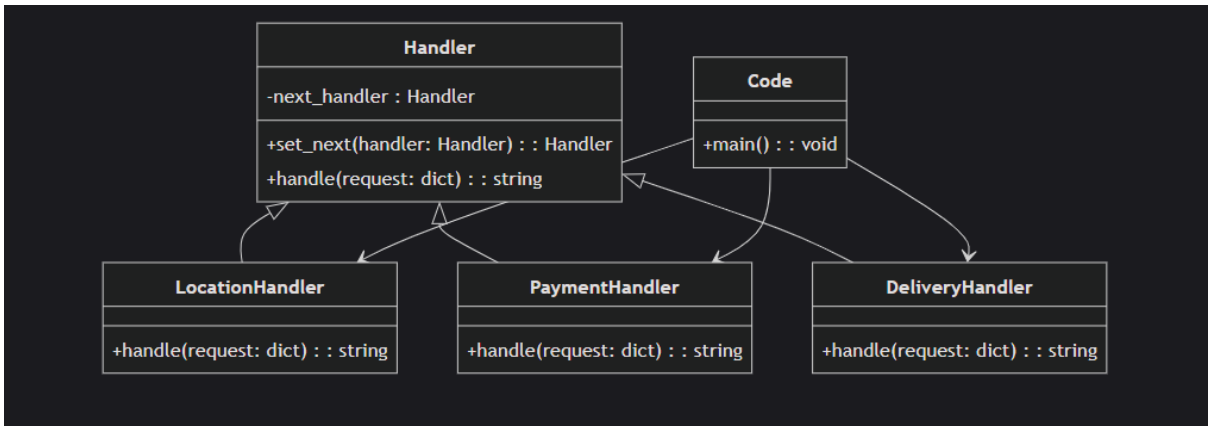
บทที่ 4

ผลการดำเนินการ

การศึกษาครั้งนี้มุ่งเน้นการทดสอบและเปรียบเทียบประสิทธิภาพของโค้ดที่สร้างขึ้นโดยเครื่องมือปัญญาประดิษฐ์ต่างๆ โดยใช้รูปแบบการออกแบบ (Design Pattern) สองแบบ คือ Chain of Responsibility Pattern และ State Pattern ผลการทดสอบแสดงให้เห็นถึงความแตกต่างในประสิทธิภาพและความถูกต้องของโค้ดที่สร้างขึ้นโดยเครื่องมือต่างๆ อีกทั้งยังมีการจัดทำแผนภาพไดอะแกรมหลังจากการประมวล AI เพื่อดูการทำงานของโค้ดว่าเป็นไปตามที่คาดหวังหรือไม่

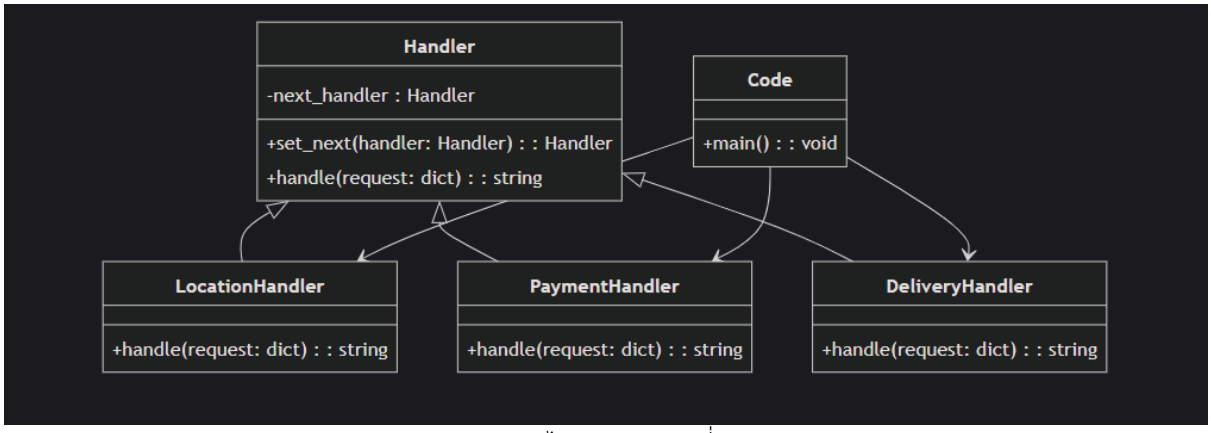
1 ChatGPT (Python)

1.1 Chain of responsibility Pattern



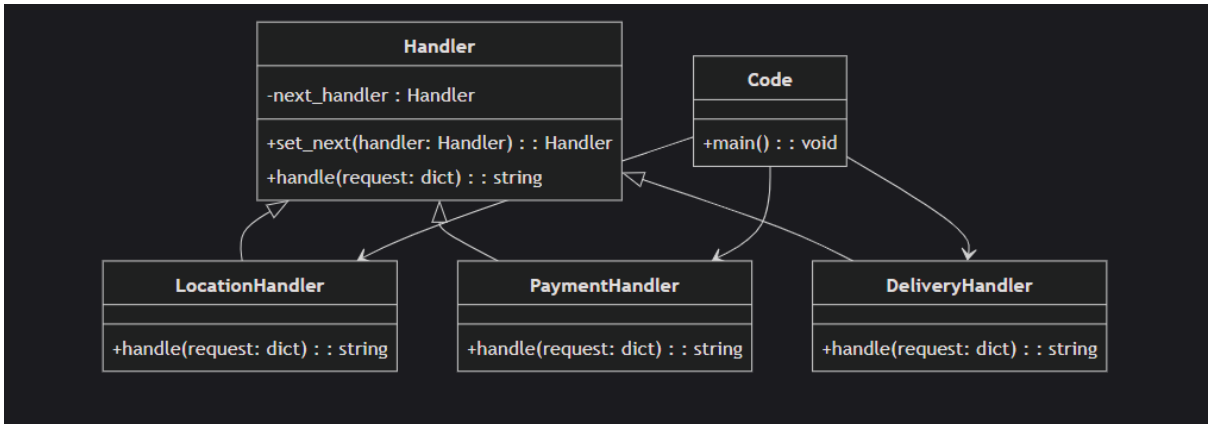
แผนภาพไดอะแกรมรอบที่หนึ่ง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบหน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สอง

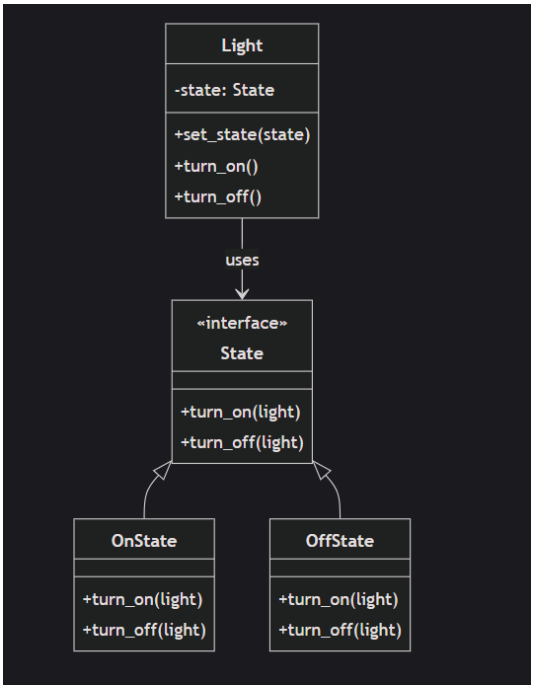
จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบหน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สาม

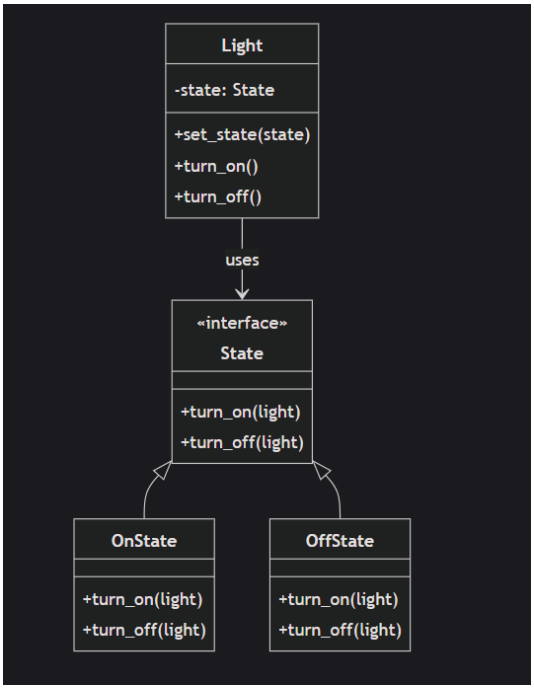
จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบหน่วยการทดสอบ เช่น Branch Coverage

1.2 State Pattern



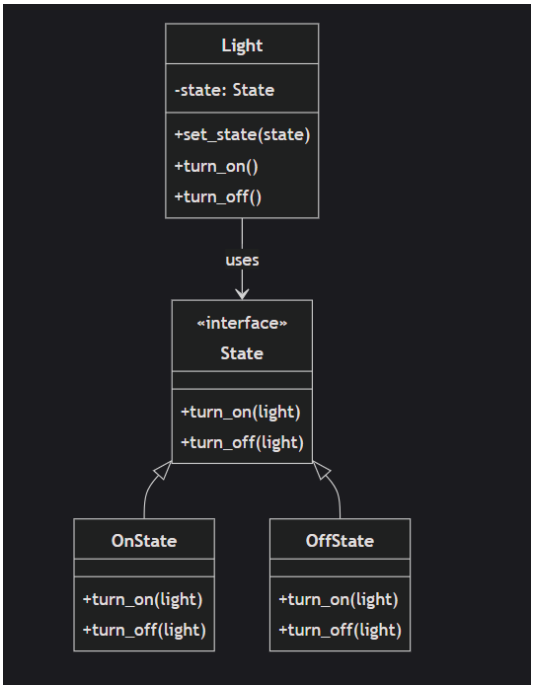
แผนภาพไดอะแกรมรอบที่หนึ่ง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบหน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สอง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage

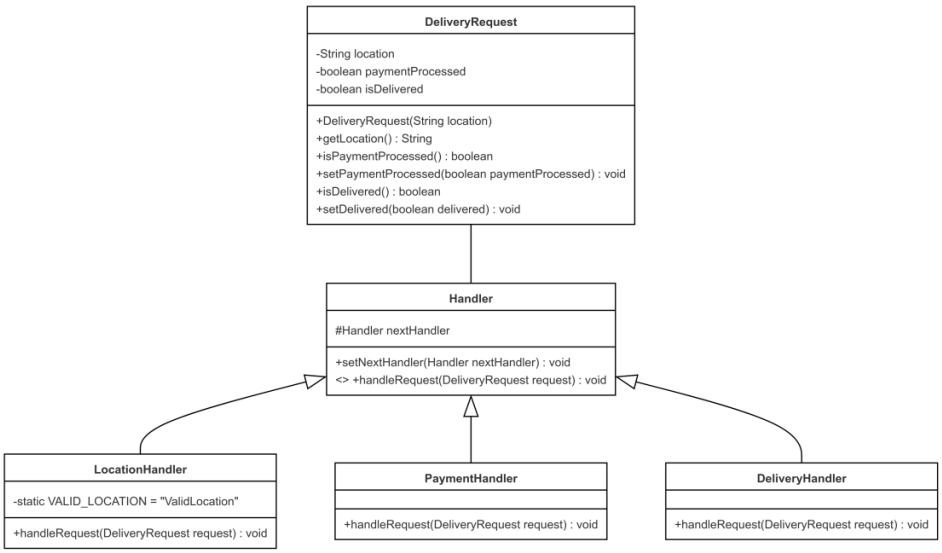


แผนภาพไดอะแกรมรอบที่สาม

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage

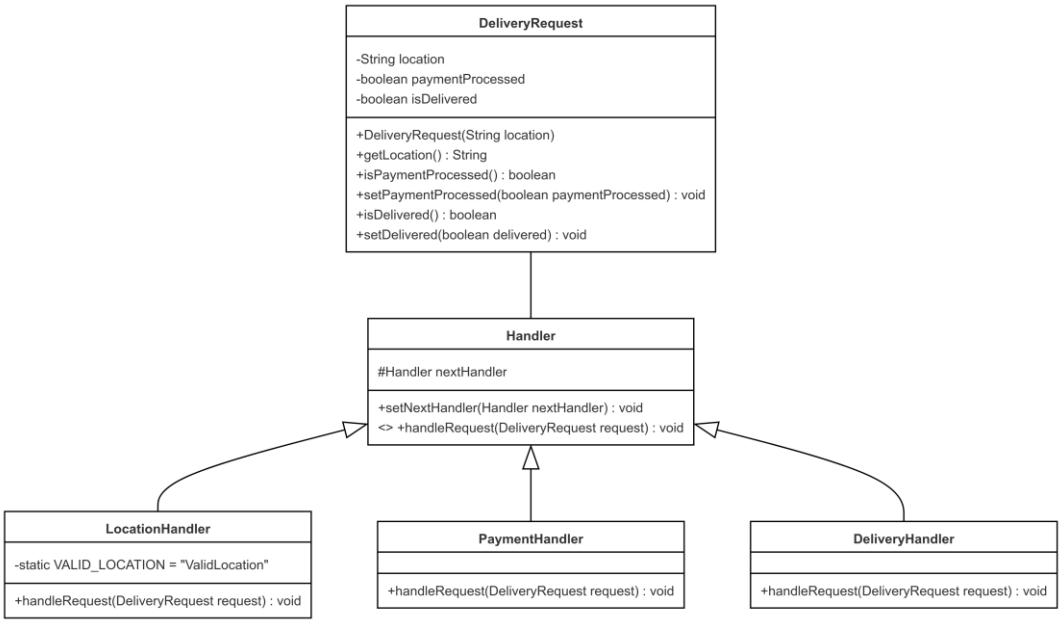
2. ChatGPT (Java)

2.1 Chain of responsibility



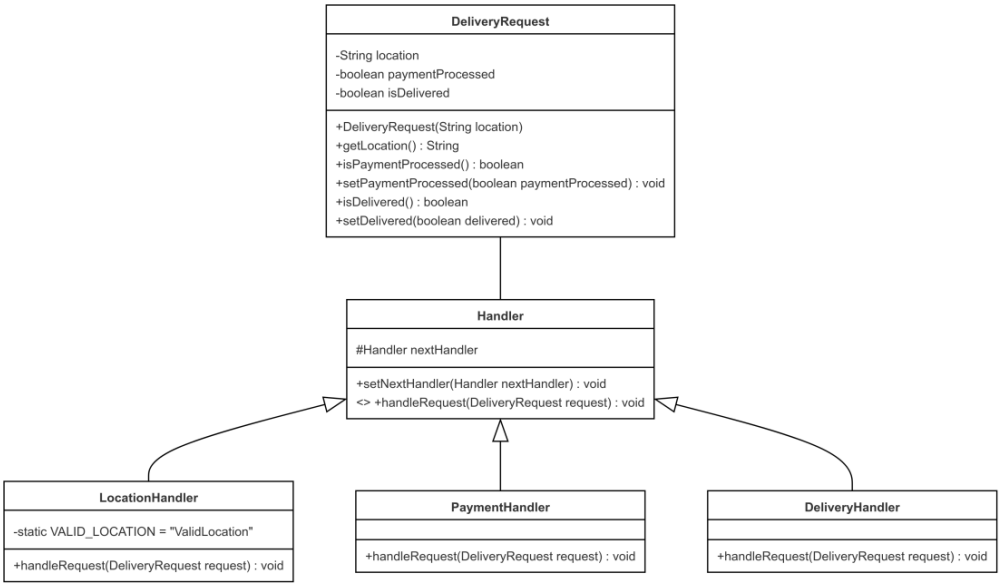
แผนภาพไดอะแกรมรอบที่หนึ่ง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สอง

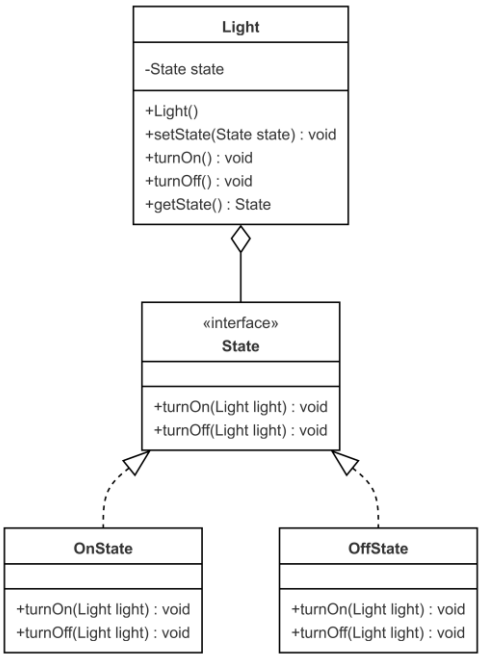
จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สาม

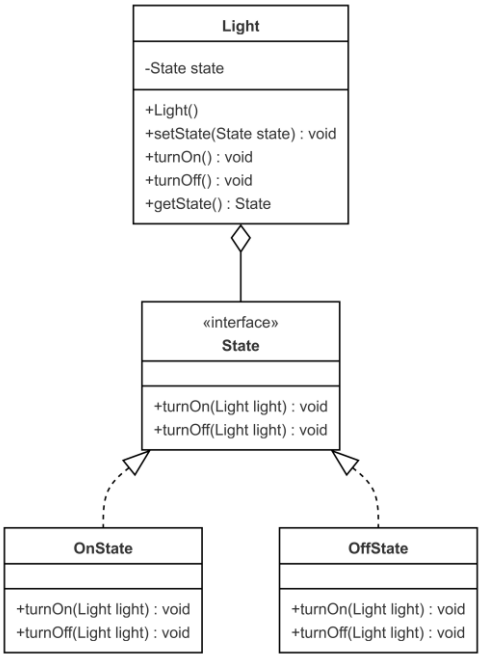
จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage

2.2 State Pattern



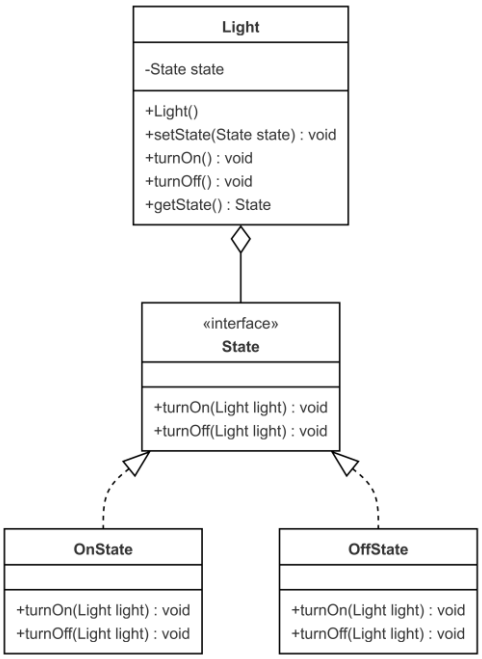
แผนภาพไดอะแกรมรอบที่หนึ่ง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สอง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage

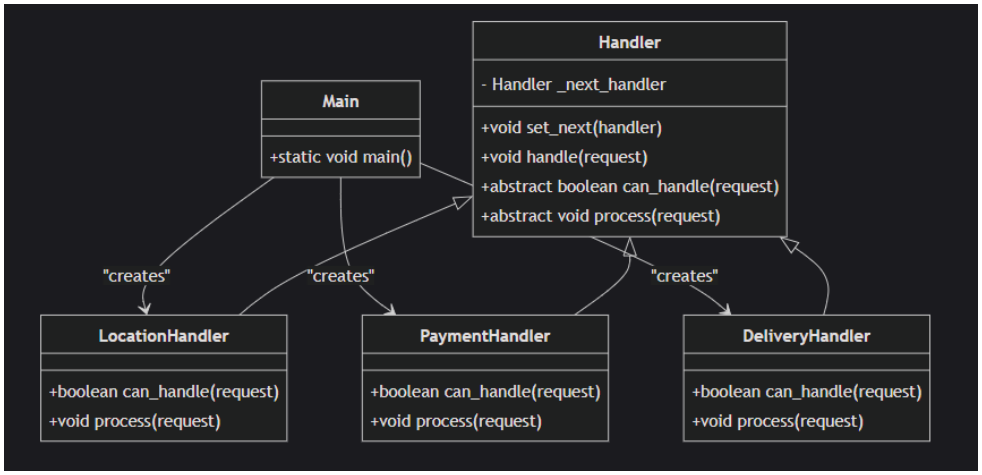


แผนภาพไดอะแกรมรอบที่สาม

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage

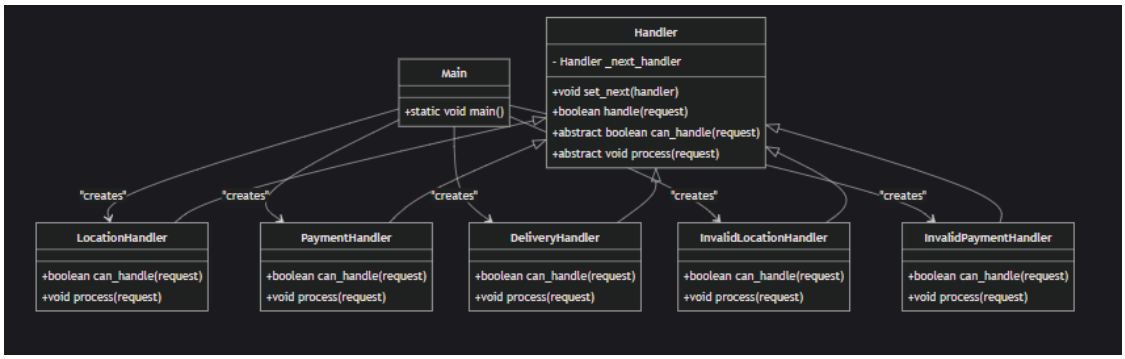
3. Gemini Pro (Python)

3.1 Chain of responsibility



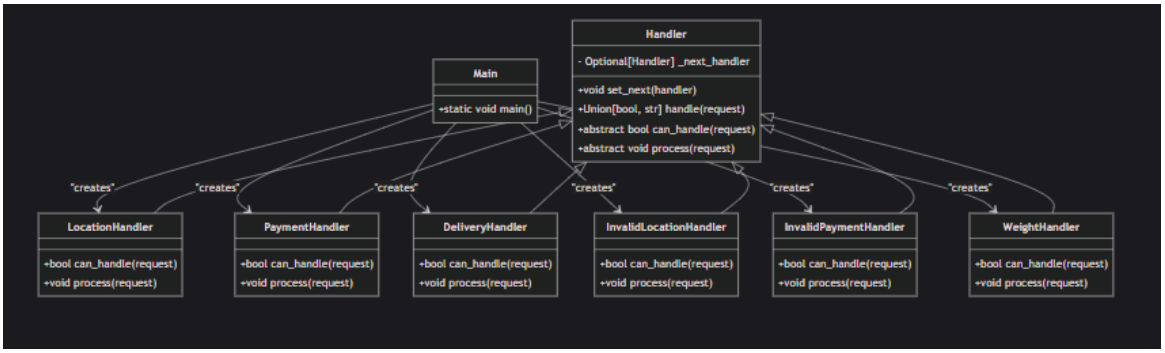
แผนภาพไดอะแกรมรอบที่หนึ่ง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สอง

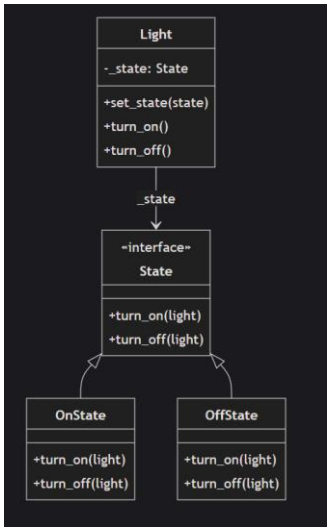
จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สาม

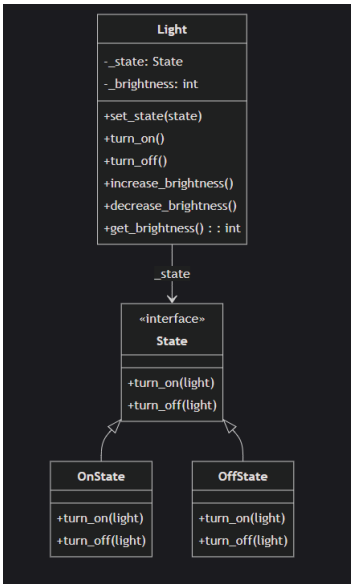
จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage

3.2 State Pattern



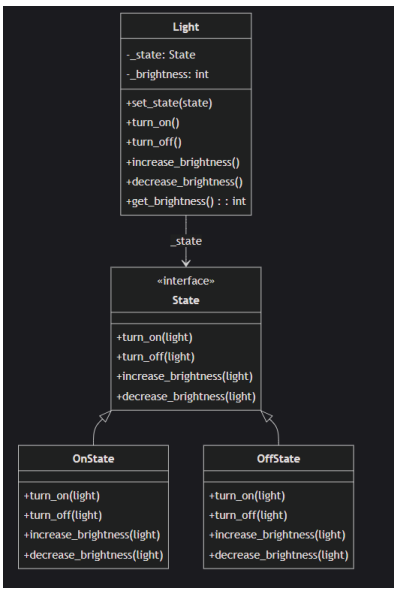
แผนภาพไดอะแกรมรอบที่หนึ่ง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สอง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage

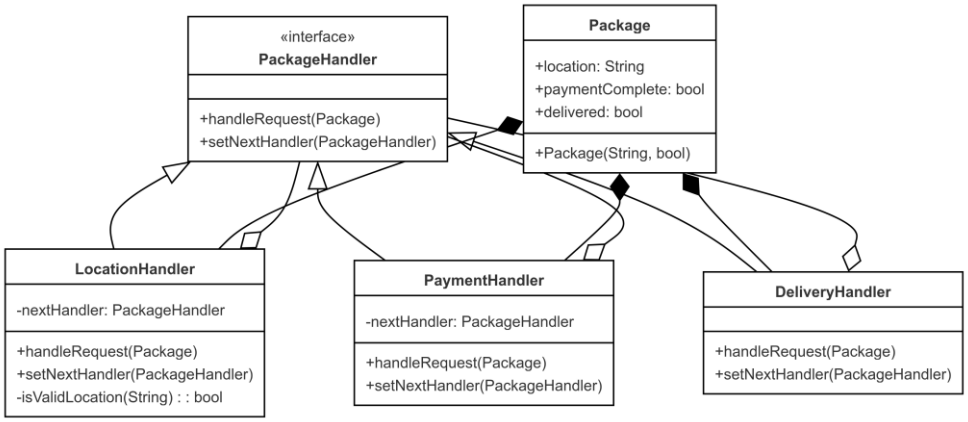


แผนภาพไดอะแกรมรอบที่สาม

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage

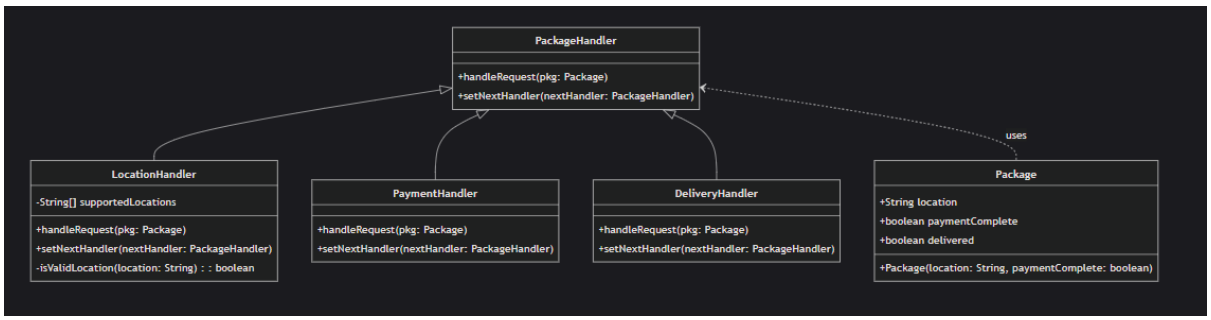
4. Gemini Pro (Java)

4.1 Chain of responsibility



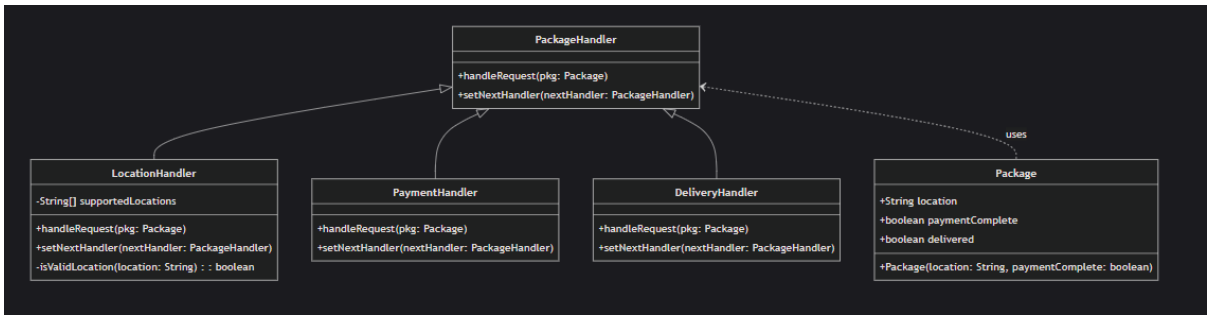
แผนภาพไดอะแกรมรอบที่หนึ่ง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สอง

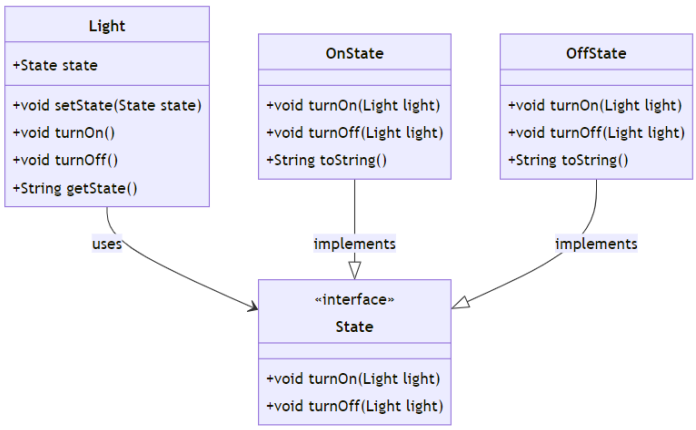
จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สาม

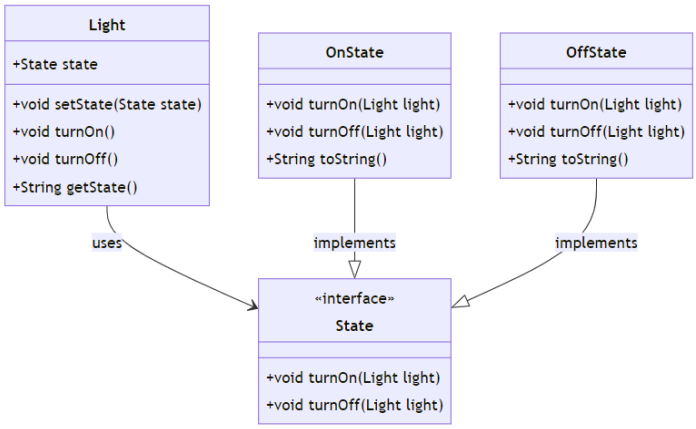
จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage

4.2 State Pattern



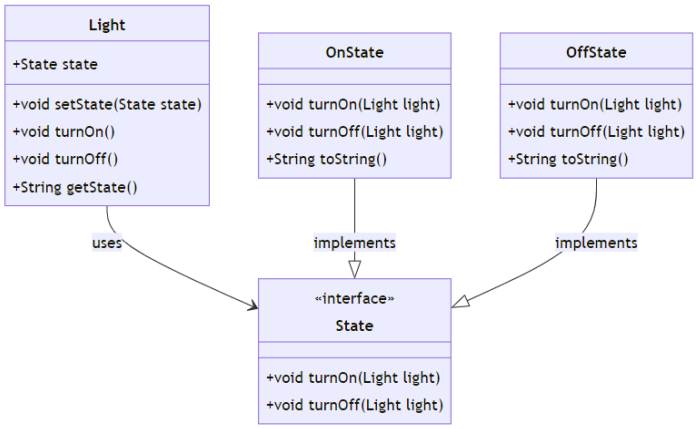
แผนภาพไดอะแกรมรอบที่หนึ่ง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบหน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สอง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบหน่วยการทดสอบ เช่น Branch Coverage

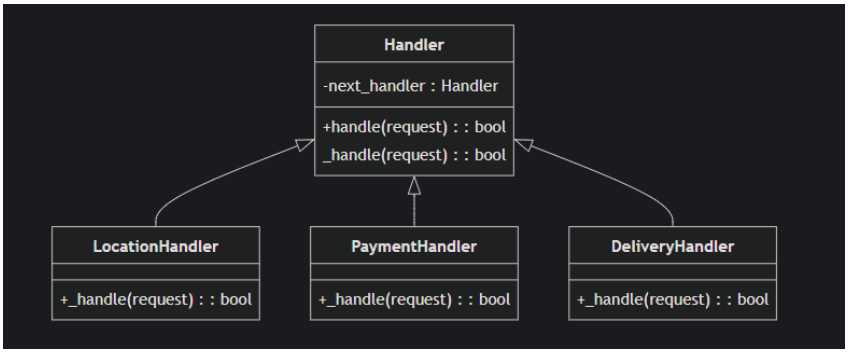


แผนภาพไดอะแกรมรอบที่สาม

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage

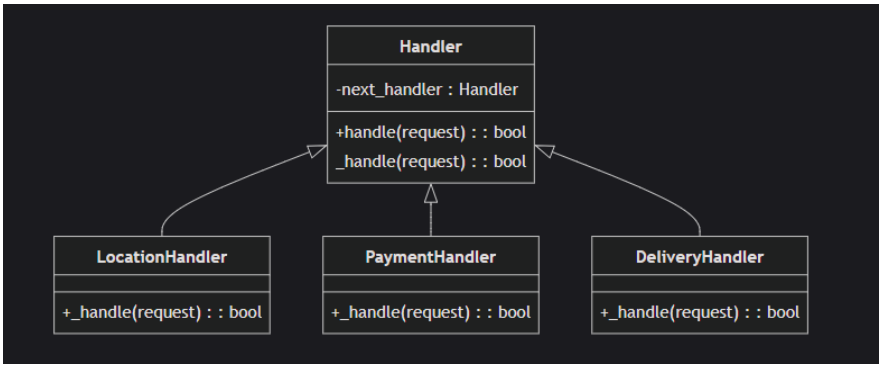
5. Gemini Flash (Python)

5.1 Chain of responsibility



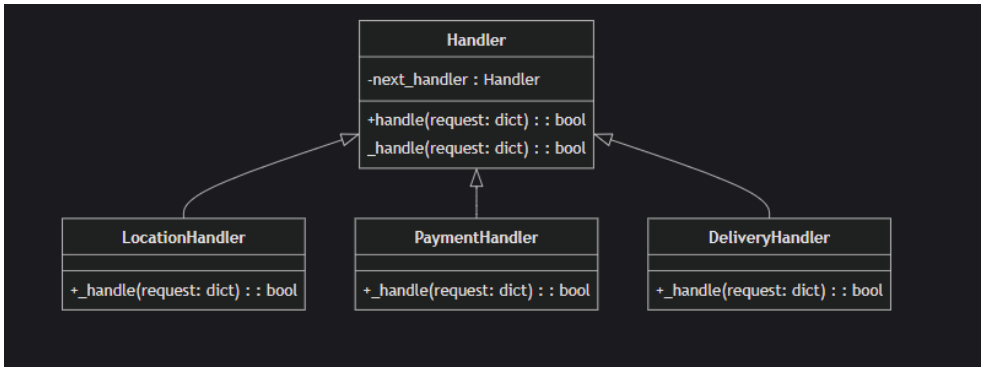
แผนภาพไดอะแกรมรอบที่หนึ่ง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สอง

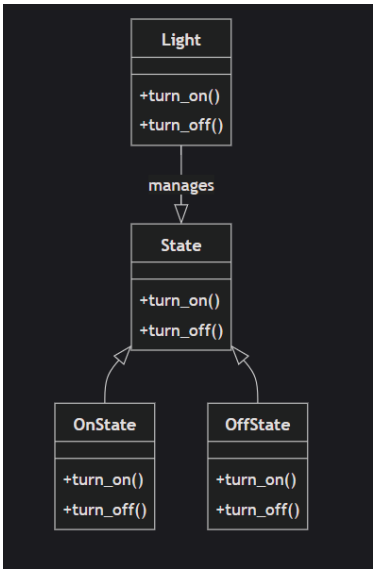
จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สาม

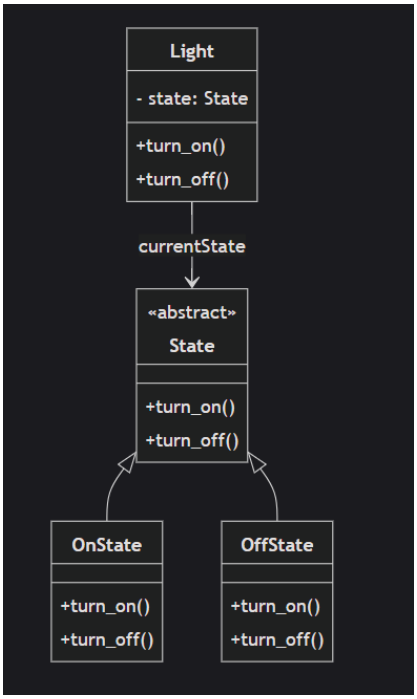
จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage

5.2 State Pattern



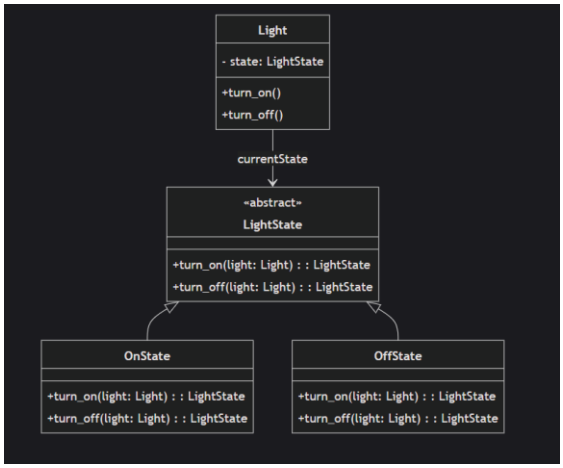
แผนภาพไดอะแกรมรอบที่หนึ่ง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สอง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage

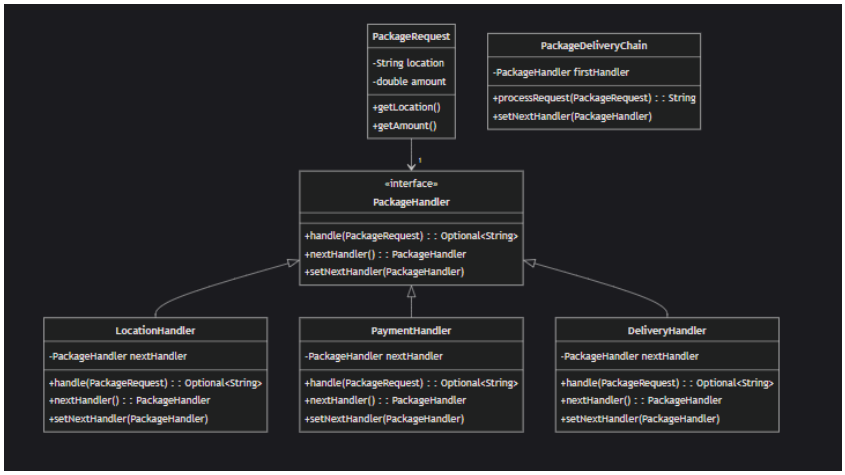


แผนภาพไดอะแกรมรอบที่สาม

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage

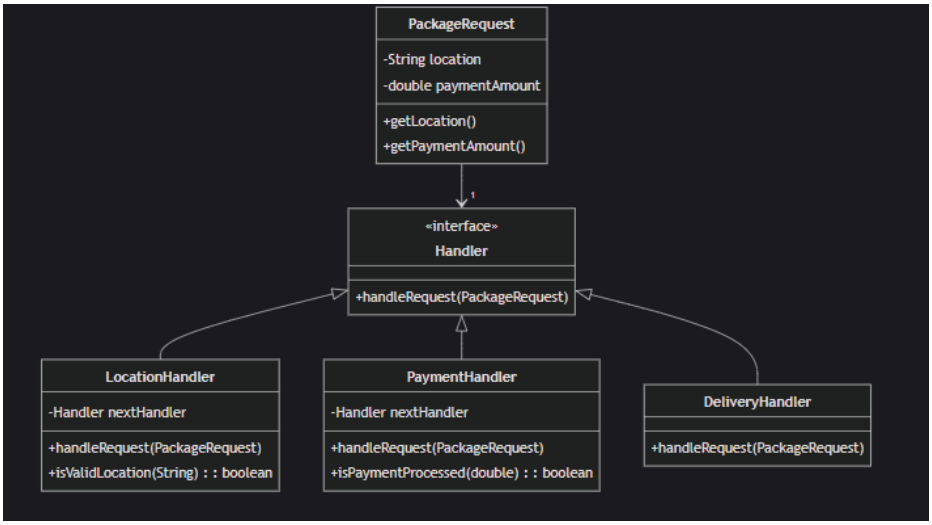
6 Gemini Flash (Java)

6.1 Chain of responsibility



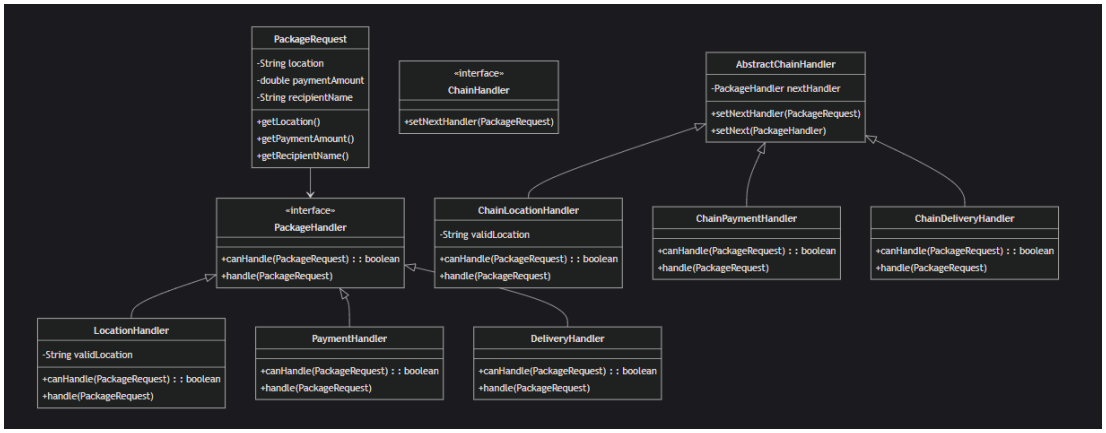
แผนภาพไดอะแกรมรอบที่หนึ่ง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สอง

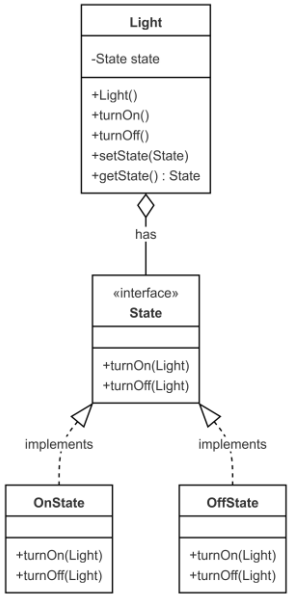
จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สาม

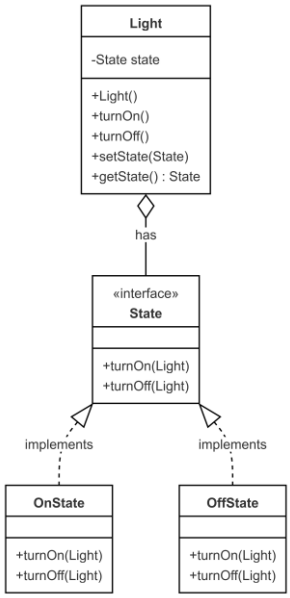
จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบหน่วยการทดสอบ เช่น Branch Coverage

6.2 State Pattern



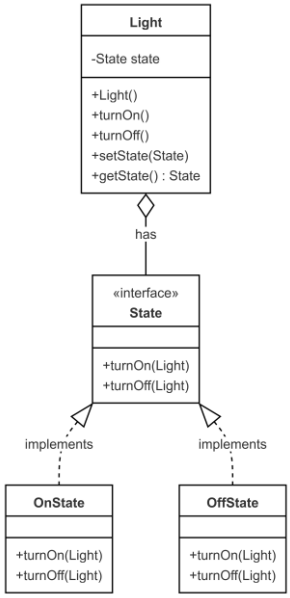
แผนภาพไดอะแกรมรอบที่หนึ่ง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบหน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สอง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage

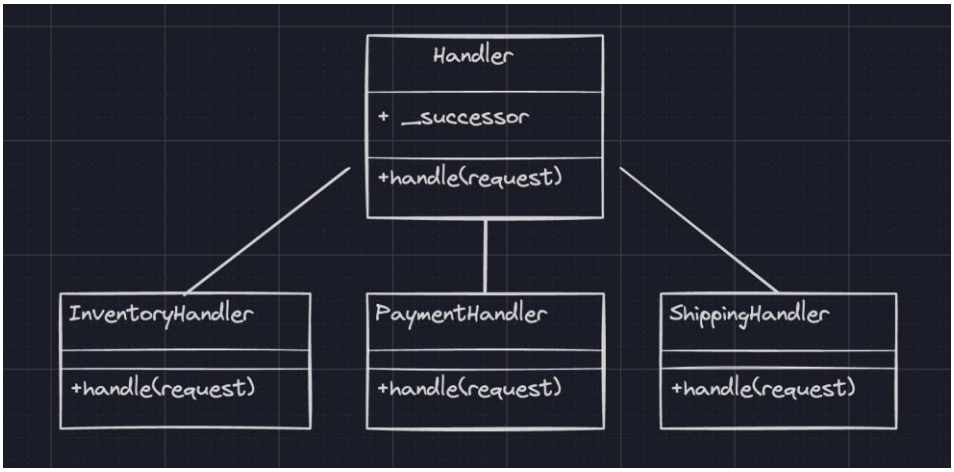


แผนภาพไดอะแกรมรอบที่สาม

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage

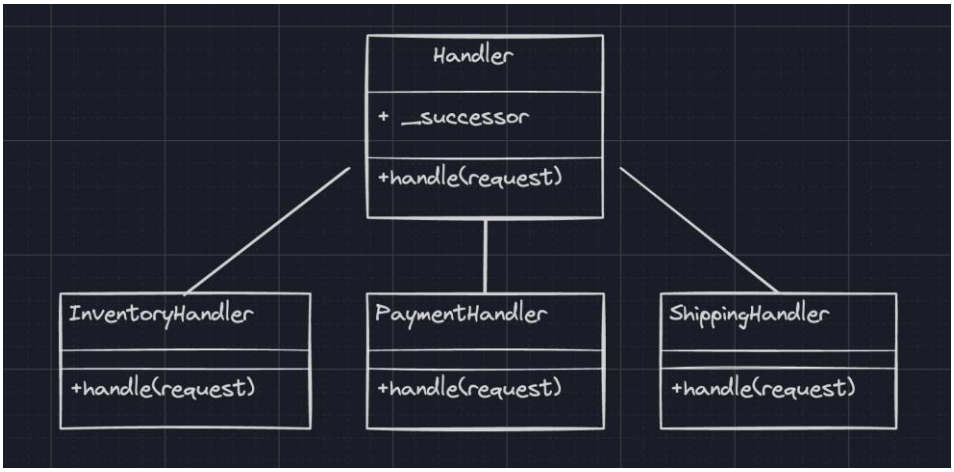
7 Github Copilot (Python)

7.1 Chain of responsibility



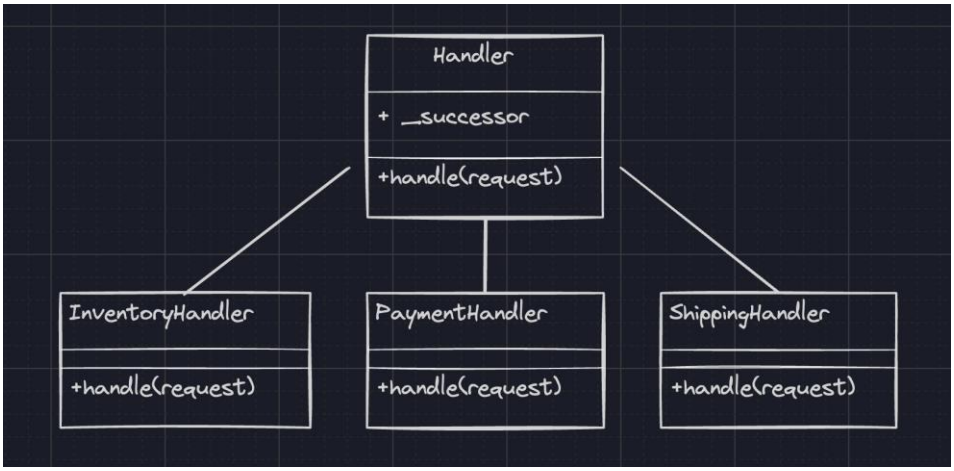
แผนภาพไดอะแกรมรอบที่หนึ่ง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สอง

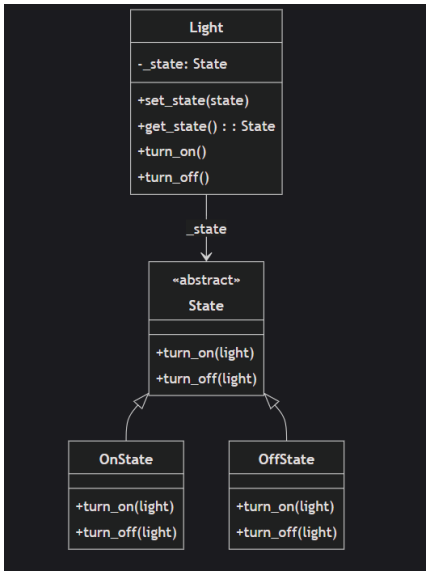
จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สาม

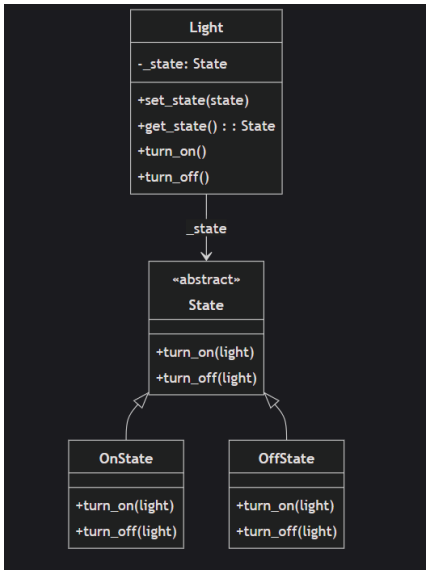
จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบหน่วยการทดสอบ เช่น Branch Coverage

7.2 State Pattern



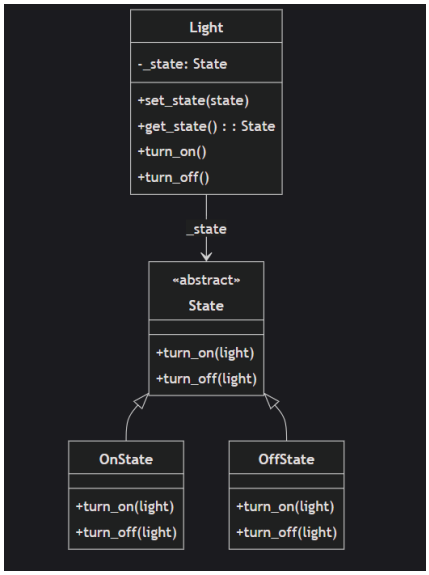
แผนภาพไดอะแกรมรอบที่หนึ่ง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบหน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สอง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage

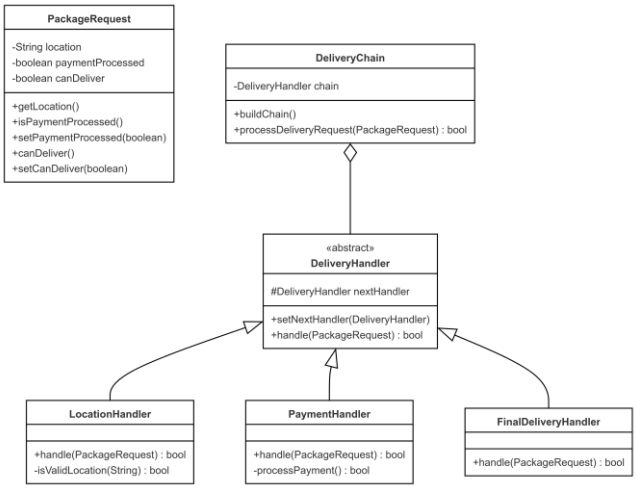


แผนภาพไดอะแกรมรอบที่สาม

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage

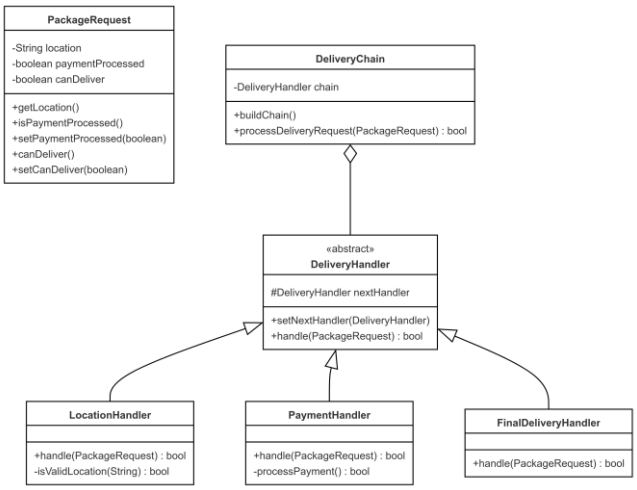
8 Github Copilot (Java)

8.1 Chain of responsibility



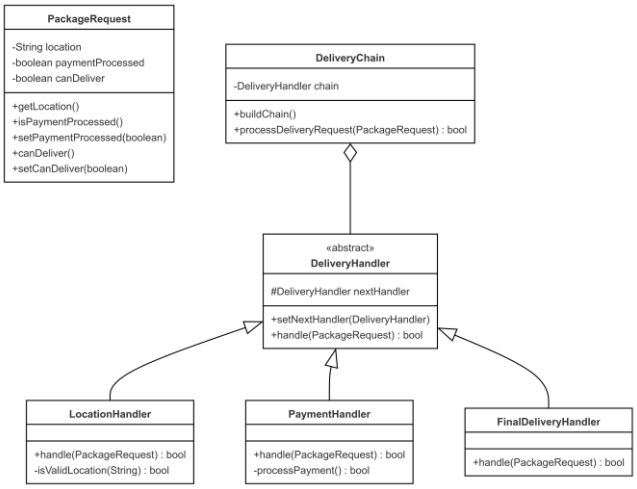
แผนภาพไดอะแกรมรอบที่หนึ่ง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สอง

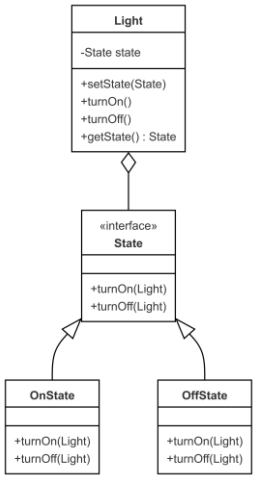
จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สาม

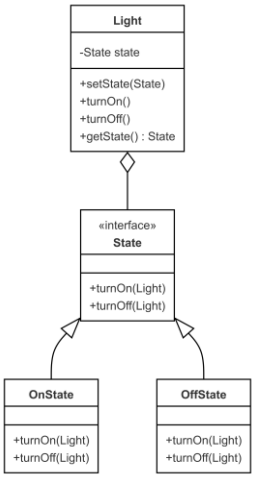
จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบหน่วยการทดสอบ เช่น Branch Coverage

8.2 State Pattern



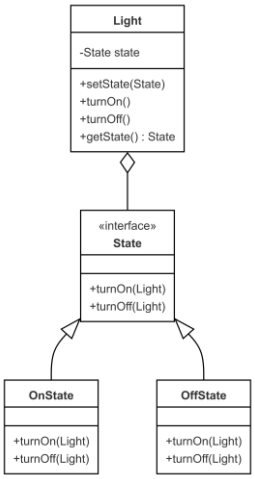
แผนภาพไดอะแกรมรอบที่หนึ่ง

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบหน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สาม

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage



แผนภาพไดอะแกรมรอบที่สาม

จากการประเมินโค้ดได้ถูกสร้างตามข้อกำหนดที่ระบุไว้ได้อย่างถูกต้องครบถ้วน ทั้ง ภาษา รูปแบบการออกแบบ หน่วยการทดสอบ เช่น Branch Coverage

ในการทดสอบภาษา Python ได้แบ่งการทดสอบออกเป็นสามรอบสำหรับแต่ละรูปแบบการออกแบบ โดยมีผู้ออกแบบคำสั่ง คือ คณะผู้จัดทำที่รับผิดชอบคนละเครื่องมือเอไอ เพื่อให้งานดำเนินไปอย่างรวดเร็วมากยิ่งขึ้น

ผลการทดสอบโค้ดที่สร้างโดย ChatGPT แสดงให้เห็นถึงประสิทธิภาพที่ดีเยี่ยม โดยผ่านการทดสอบทั้งหมดในทุกรอบและทั้งสองรูปแบบการออกแบบ แสดงให้เห็นถึงความสามารถของ ChatGPT ในการสร้างโค้ดที่มีคุณภาพและตรงตามข้อกำหนดของทั้ง Chain of Responsibility และ State Pattern

ในทางตรงกันข้าม โค้ดที่สร้างโดย Gemine_Flash พบปัญหาในหลายการทดสอบ โดยเฉพาะอย่างยิ่งในรอบแรกของรูปแบบ Chain of Responsibility รอบแรกและรอบที่สองของรูปแบบ State Pattern อย่างไรก็ตาม มีการปรับปรุงในรอบถัดมา ซึ่งผ่านการทดสอบสำหรับทั้งสองรูปแบบ แสดงให้เห็นถึงการพัฒนาและการเรียนรู้ของระบบ

Gemine_Pro แสดงผลลัพธ์ที่น่าประทับใจ โดยผ่านการทดสอบทั้งหมดในทุกรอบและทั้งสองรูปแบบการออกแบบ เช่นเดียวกับ GitHub Copilot ที่แสดงผลลัพธ์ได้ยอดเยี่ยมเช่นกัน โดยผ่านการทดสอบทั้งหมดในทุกรอบและทั้งสองรูปแบบการออกแบบ แสดงให้เห็นถึงความสามารถในการสร้างโค้ดที่มีคุณภาพและตรงตามข้อกำหนดของทั้ง Chain of Responsibility และ State Pattern

ผลการทดสอบของภาษา Python แสดงให้เห็นว่าเครื่องมือปัญญาประดิษฐ์ส่วนใหญ่สามารถสร้างโค้ดที่มีคุณภาพและตรงตามรูปแบบการออกแบบที่กำหนดได้อย่างมีประสิทธิภาพ โดย ChatGPT, Gemine_Pro และ GitHub Copilot แสดงผลลัพธ์ที่สอดคล้องและน่าเชื่อถือในทุกการทดสอบ ในขณะที่ Gemine_Flash แสดงให้เห็นถึงการพัฒนาและปรับปรุงประสิทธิภาพในรอบสุดท้ายของการทดสอบ

ในส่วนการทดสอบภาษา Java ได้แบ่งการทดสอบออกเป็นสามรอบสำหรับแต่ละรูปแบบการออกแบบ โดยมีผู้ออกแบบคำสั่ง คือ คณะผู้จัดทำที่รับผิดชอบคนละเครื่องมือเอไอ เพื่อให้งานดำเนินไปอย่างรวดเร็วมากยิ่งขึ้น

ผลการทดสอบโค้ด Java ที่สร้างโดย ChatGPT แสดงให้เห็นถึงประสิทธิภาพที่ดี โดยผ่านการทดสอบเกือบทั้งหมด ยกเว้นในกรณีของ State Pattern ในรอบที่สองที่เกิดข้อผิดพลาด อย่างไรก็ตาม ChatGPT สามารถแก้ไขปัญหาและผ่านการทดสอบในรอบที่สามได้สำเร็จ

Gemini Flash แสดงผลลัพธ์ที่ไม่สม่ำเสมอในการทดสอบ Java โดยเฉพาะอย่างยิ่งในส่วนของ Chain of Responsibility ที่เกิดข้อผิดพลาดในทุกรอบการทดสอบ อย่างไรก็ตาม ผลการทดสอบสำหรับ State Pattern นั้นดีขึ้น โดยผ่านการทดสอบทั้งสามรอบ

Gemini Pro แสดงผลลัพธ์ที่น่าผิดหวังในการทดสอบ Java โดยเกิดข้อผิดพลาดในสองรอบแรกของการทดสอบ Chain of Responsibility แต่ผ่านทุกการทดสอบของ State Pattern

GitHub Copilot แสดงผลลัพธ์ที่ยอดเยี่ยมในการทดสอบ Java โดยผ่านการทดสอบทั้งหมดในทุกรอบและทั้งสองรูปแบบการออกแบบ แสดงให้เห็นถึงความสามารถในการสร้างโค้ด Java ที่มีคุณภาพและตรงตามข้อกำหนดของทั้ง Chain of Responsibility และ State Pattern

ผลการทดสอบนี้แสดงให้เห็นว่าเครื่องมือปัญญาประดิษฐ์ส่วนใหญ่สามารถสร้างโค้ด Java ที่มีคุณภาพและตรงตามรูปแบบการออกแบบที่กำหนดได้ แต่ยังมีความแตกต่างในประสิทธิภาพระหว่างเครื่องมือต่างๆ GitHub Copilot แสดงผลลัพธ์ที่โดดเด่นที่สุด ในขณะที่ ChatGPT และ Gemini Pro แสดงให้เห็นถึงความสามารถในการเรียนรู้และปรับปรุงประสิทธิภาพในรอบสุดท้ายของการทดสอบ Gemini Flash แสดงผลลัพธ์ที่ไม่สม่ำเสมอ โดยเฉพาะในส่วนของ Chain of Responsibility หากคิดเป็นร้อยละจำนวนการทดสอบผ่านและไม่ผ่านจะได้ดังนี้

เพื่อทดสอบคุณภาพของโค้ดได้ทำการตรวจสอบความซับซ้อนของโค้ดด้วยเครื่องมือต่างๆ เช่น Junit, Radon ที่สามารถตรวจเช็ค มีเกณฑ์วัดผลดังนี้

Chain of responsbility	
Name and round	Cyclomatic Complexity
ChatGPT- r1	A (2.1)
ChatGPT- r2	A (2.1)
ChatGPT- r3	A (2.7)
Gemini-Pro-r1	A (1.4)
Gemini-Pro-r2	A (1.381)
Gemini-Pro-r3	A (1.375)
Gemini Flash-r1	A (2.367)
Gemini Flash-r2	A (2.367)
Gemini Flash-r3	A (2.367)
Github_copilot-r1	A (2.555)
Github_copilot-r2	A (2.555)
Github_copilot-r3	A (2.555)

State	
Name and round	Cyclomatic Complexity
ChatGPT- r1	A (1.286)
ChatGPT- r2	A (1.286)
ChatGPT- r3	A (1.286)
Gemini-Pro-r1	A (1.286)
Gemini-Pro-r2	A (1.353)
Gemini-Pro-r3	A (1.174)
Gemini Flash-r1	A (1.308)
Gemini Flash-r2	A (1.308)
Gemini Flash-r3	A (1.308)
Github_copilot-r1	A (1.267)
Github_copilot-r2	A (1.267)
Github_copilot-r3	A (1.267)

ภาพแสดงความซับซ้อนของภาษาไพทอนแบบภาพรวม

Chain of responsbility	
Name and round	Cyclomatic Complexity
ChatGPT- r1	10
ChatGPT- r2	10
ChatGPT- r3	10
Gemini-Pro-r1	13
Gemini-Pro-r2	13
Gemini-Pro-r3	13
Gemini Flash-r1	9
Gemini Flash-r2	4
Gemini Flash-r3	5
Github_copilot-r1	12
Github_copilot-r2	12
Github_copilot-r3	12

State	
Name and round	Cyclomatic Complexity
ChatGPT- r1	1
ChatGPT- r2	1
ChatGPT- r3	1
Gemini-Pro-r1	1
Gemini-Pro-r2	1
Gemini-Pro-r3	1
Gemini Flash-r1	1
Gemini Flash-r2	1
Gemini Flash-r3	1
Github_copilot-r1	1
Github_copilot-r2	1
Github_copilot-r3	1

ภาพแสดงความซับซ้อนของภาษาจาวาแบบภาพรวม

ในการทำโครงงานนี้ได้เล็งเห็นถึงความสำคัญของการตรวจสอบ Branch Coverage ซึ่งมีความสำคัญเพราะมันช่วยให้มั่นใจว่าโค้ดในแต่ละสาขาของโปรแกรมได้รับการทดสอบทั้งหมด ไม่ใช่แค่การทำงานโดยรวมของโปรแกรมเท่านั้น ยิ่งมีค่าเปอร์เซ็นต์ที่สูง แสดงว่าโค้ดมีการใช้งานสาขาได้ครอบคลุมมากยิ่งขึ้น

Chain of responsbility	
Name and round	Branch coverage (%)
ChatGPT- r1	100
ChatGPT- r2	100
ChatGPT- r3	100
Gemini-Pro-r1	100
Gemini-Pro-r2	100
Gemini-Pro-r3	100
Gemini Flash-r1	0
Gemini Flash-r2	0
Gemini Flash-r3	100
Github_copilot-r1	100
Github_copilot-r2	100
Github_copilot-r3	100
Avarage	83.33

State	
Name and round	Branch coverage (%)
ChatGPT- r1	100
ChatGPT- r2	100
ChatGPT- r3	100
Gemini-Pro-r1	100
Gemini-Pro-r2	100
Gemini-Pro-r3	100
Gemini Flash-r1	0
Gemini Flash-r2	0
Gemini Flash-r3	100
Github_copilot-r1	100
Github_copilot-r2	100
Github_copilot-r3	100
Avarage	83.33

ภาพแสดงการใช้งานสาขาของภาษาไพทอนแบบภาพรวม

จากผลของ Branch Coverage ทั้งสองแบบ ชี้ให้เห็นถึงความสามารถของเครื่องที่ทำให้มีการใช้งานทุกสาขาเป็นส่วนใหญ่ ถึงแม้ว่าจะมีบางส่วนได้ 0 เปอร์เซ็นต์ หากแต่เป็นเฉพาะในรอบแรกและรอบสอง แต่กระนั้นก็มีการปรับปรุงให้ดีขึ้นโดยที่รอบสามได้ 100 เปอร์เซ็นต์

Chain of responsbility	
Name and round	Branch coverage (%)
ChatGPT- r1	75
ChatGPT- r2	75
ChatGPT- r3	75
Gemini-Pro-r1	83
Gemini-Pro-r2	83
Gemini-Pro-r3	83
Gemini Flash-r1	0
Gemini Flash-r2	0
Gemini Flash-r3	0
Github_copilot-r1	0
Github_copilot-r2	0
Github_copilot-r3	0
Avarage	39.5

State	
Name and round	Branch coverage (%)
ChatGPT- r1	0
ChatGPT- r2	100
ChatGPT- r3	100
Gemini-Pro-r1	100
Gemini-Pro-r2	100
Gemini-Pro-r3	100
Gemini Flash-r1	100
Gemini Flash-r2	100
Gemini Flash-r3	100
Github_copilot-r1	0
Github_copilot-r2	0
Github_copilot-r3	0
Avarage	66.67

ภาพแสดงการใช้งานสาขาของภาษาจาวาแบบภาพรวม

จากผลของ Branch Coverage ทั้งสองแบบ ชี้ให้เห็นความแตกต่างในความครอบคลุมของการทดสอบ State pattern มีความครอบคลุมสูงกว่า โดยเฉพาะในเครื่องมือ Gemini-Pro และ Gemini Flash ที่ได้ 100% ในขณะที่ Chain of Responsibility มีค่าเฉลี่ยที่ต่ำกว่า โดย Github Copilot และ Gemini Flash ใน Chain of Responsibility มีค่า coverage เป็น 0% ซึ่งแสดงถึงการขาดการทดสอบในหลายสาขาของโค้ด การได้ 100% ใน State pattern บ่งบอกถึงการทดสอบที่ครอบคลุมทุกเงื่อนไข

บทที่ 5
อภิปรายผล ประโยชน์ที่ได้รับจากโครงงาน และข้อเสนอแนะ

อภิปรายผล

การศึกษาครั้งนี้ได้เปรียบเทียบประสิทธิภาพของเครื่องมือปัญญาประดิษฐ์ต่างๆ ในการสร้างโค้ดภาษา Python และ Java โดยใช้รูปแบบการออกแบบ Chain of Responsibility และ State Pattern ผลการศึกษาแสดงให้เห็นถึงความแตกต่างที่น่าสนใจระหว่างเครื่องมือและภาษาที่ใช้

1. ความแตกต่างระหว่างภาษา:
    1.1 Python เครื่องมือ AI ส่วนใหญ่แสดงผลลัพธ์ที่ดีกว่าในการสร้างโค้ด Python โดยเฉพาะ ChatGPT, Gemini Pro และ GitHub Copilot ที่ผ่านการทดสอบทั้งหมด
    1.2 Java ผลลัพธ์มีความแตกต่างมากขึ้น โดย GitHub Copilot แสดงประสิทธิภาพสูงสุด ในขณะที่เครื่องมืออื่นๆ มีข้อผิดพลาดในบางกรณี
2. ประสิทธิภาพของเครื่องมือ AI:
    2.1 GitHub Copilot แสดงประสิทธิภาพสูงสุดในทั้งสองภาษา
    2.2 ChatGPT มีประสิทธิภาพดีในภาษา Python แต่พบข้อผิดพลาดบ้างในภาษา Java
    2.3 Gemini Flash และ Gemini Pro แสดงผลลัพธ์ที่ไม่สม่ำเสมอ โดยเฉพาะในภาษา Java
3. ความสามารถในการเรียนรู้และปรับปรุง:
    3.1 หลายเครื่องมือแสดงให้เห็นถึงการปรับปรุงประสิทธิภาพในรอบหลังๆ ของการทดสอบ แสดงถึงความสามารถในการเรียนรู้และปรับปรุงตัวเอง
4. ความแตกต่างระหว่างรูปแบบการออกแบบ:
    4.1 Chain of Responsibility มักพบปัญหามากกว่า State Pattern โดยเฉพาะในภาษา Java
    4.2 State Pattern ดูเหมือนจะง่ายกว่าสำหรับ AI ในการสร้างโค้ดที่ถูกต้อง

ประโยชน์ที่ได้รับจากโครงงาน

1. เข้าใจความสามารถและข้อจำกัดของเครื่องมือ AI ในการสร้างโค้ด
2. ทราบถึงความแตกต่างในประสิทธิภาพระหว่างภาษา Python และ Java สำหรับ AI
3. เห็นถึงความสามารถของ AI ในการใช้รูปแบบการออกแบบซอฟต์แวร์ที่ซับซ้อน
4. ได้แนวทางในการเลือกใช้เครื่องมือ AI ที่เหมาะสมสำหรับงานพัฒนาซอฟต์แวร์
5. เข้าใจถึงความสำคัญของการทดสอบและการปรับปรุงโค้ดที่สร้างโดย AI

ข้อเสนอแนะ

1. ขยายขอบเขตการทดสอบ:
    1.1 เพิ่มจำนวนรอบการทดสอบเพื่อดูแนวโน้มการปรับปรุงของแต่ละเครื่องมือ
    1.2 ทดสอบกับรูปแบบการออกแบบอื่นๆ เพื่อประเมินความสามารถที่หลากหลายมากขึ้น
2. ปรับปรุงวิธีการทดสอบ:
    2.1 พัฒนาเกณฑ์การประเมินที่ละเอียดมากขึ้น นอกเหนือจากการผ่าน/ไม่ผ่าน
    2.2 วิเคราะห์คุณภาพโค้ดในแง่มุมอื่นๆ เช่น ความสะอาดของโค้ด การใช้ทรัพยากร
3. ศึกษาปัจจัยที่ส่งผลต่อประสิทธิภาพ:
    3.1 วิเคราะห์เหตุผลเบื้องหลังความแตกต่างในประสิทธิภาพระหว่างภาษาและเครื่องมือ
    3.2 ศึกษาวิธีการปรับปรุงประสิทธิภาพของ AI ในการสร้างโค้ด Java

4. พัฒนาแนวทางการใช้งาน AI ในการพัฒนาซอฟต์แวร์:
    4.1 สร้างแนวทางปฏิบัติที่ดีในการใช้ AI เพื่อสนับสนุนการพัฒนาซอฟต์แวร์
    4.2 พัฒนาเทคนิคการตรวจสอบและปรับปรุงโค้ดที่สร้างโดย AI
5. ศึกษาผลกระทบระยะยาว:
    5.1 ศึกษาผลกระทบของการใช้ AI ในการพัฒนาซอฟต์แวร์ต่อทักษะของนักพัฒนา
    5.2 ประเมินประสิทธิภาพและความยั่งยืนของโค้ดที่สร้างโดย AI ในระยะยาว

โครงงานได้เปิดมุมมองใหม่เกี่ยวกับการใช้ AI ในการพัฒนาซอฟต์แวร์ และแสดงให้เห็นถึงศักยภาพและข้อจำกัดของเทคโนโลยีนี้ การศึกษาเพิ่มเติมและการพัฒนาอย่างต่อเนื่องจะช่วยปรับปรุงประสิทธิภาพของ AI และนำไปสู่การใช้งานที่มีประสิทธิภาพมากขึ้นในอนาคต

อ้างอิง

1 Chumpolm. (2018). *Software Design Principles and Quality Assurance*. Retrieved from https://chumpolm.wordpress.com/wp-content/uploads/2018/10/chapter071.pdf

2 Refactoring.Guru. (2024). *Design Patterns*. Retrieved from https://refactoring.guru/design-patterns

3 Gokkky. (2020). *Design Pattern*. Medium. Retrieved from https://medium.com/@gokkky/design-pattern-20cb72bd6089