การจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด
โดยการใช้มาตรวัดเชิงวัตถุและอัลกอริทึมละโมบ

นายรัฐพงษ์ วงศ์เปียง

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2556

ORDERING REFACTORING USAGE FOR CODE CHANGING USING OBJECT ORIENTED METRICS AND GREEDY ALGORITHM

Mr. Ratapong Wongpiang

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science Program in Software Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2013

หัวข้อวิทยานิพนธ์	การจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด โดยการใช้มาตรวัดเชิงวัตถุและอัลกอริทึมละโมบ
โดย	นายรัฐพงษ์ วงศ์เปียง
สาขาวิชา	วิศวกรรมซอฟต์แวร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร. พรศิริ หมื่นไชยศรี

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์

(ศาสตราจารย์ ดร. บัณฑิต เอื้ออาภรณ์)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ

(รองศาสตราจารย์ ดร. ทวีตีย์ เสนีวงศ์ ณ อยุธยา)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

(รองศาสตราจารย์ ดร. พรศิริ หมื่นไชยศรี)

.....กรรมการ

(ผู้ช่วยศาสตราจารย์ นครทิพย์ พร้อมพูล)

.....กรรมการภายนอกมหาวิทยาลัย

(ผู้ช่วยศาสตราจารย์ ดร. ทรงศักดิ์ รองวิริยะพานิช)

รัฐพงษ์ วงศ์เปียง : การจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด โดยการใช้มาตรวัดเชิงวัตถุและอัลกอริทึมละโมบ. (ORDERING REFACTORING USAGE FOR CODE CHANGING USING OBJECT ORIENTED METRICS AND GREEDY ALGORITHM) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: รศ. ดร. พรศิริ หมื่นไชยศรี, 150 หน้า.

รีแฟคทอริงเป็นกระบวนการการเปลี่ยนแปลงโครงสร้างภายในของซอฟต์แวร์ โดยไม่ทำให้พฤติกรรมการทำงานของซอฟต์แวร์นั้นเปลี่ยนแปลง ทำให้ซอฟต์แวร์นั้นง่ายต่อการบำรุงรักษาในภายหลัง การปรับแก้ไขซอฟต์แวร์เพื่อเพิ่มความสามารถในการบำรุงรักษาซอฟต์แวร์หนึ่งๆ อาจจำเป็นต้องใช้วิธีรีแฟคทอริงหลายวิธีในการปรับแก้ไข ซึ่งการใช้งานวิธีรีแฟคทอริงในลำดับที่แตกต่างกันในการปรับแก้ไขโค้ดจะทำให้ได้ซอร์ซโค้ดที่แตกต่างกันซึ่งมีผลทำให้ค่าความสามารถในการบำรุงรักษาซอฟต์แวร์นั้นแตกต่างกันออกไป วิทยานิพนธ์นี้จึงนำเสนอแนวคิดในการค้นหาลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดโดยการใช้มาตรวัดเชิงวัตถุและอัลกอริทึมละโมบเพื่อให้ได้ซอร์ซโค้ดที่มีความสามารถในการบำรุงรักษาซอฟต์แวร์ที่เหมาะสมภายหลังจากการปรับแก้ไข อัลกอริทึมละโมบจะช่วยแยกลำดับการใช้งานวิธีรีแฟคทอริงที่เป็นคำตอบออกจากลำดับการใช้งานวิธีรีแฟคทอริงที่เป็นไปได้ทั้งหมดในการแก้ไขโค้ด โดยการทดลองนั้นจะทดสอบกับซอร์ซโค้ดที่ประกอบด้วยร่องรอยที่ผิดพลาดทั้ง 3 ลักษณะ คือ ร่องรอยที่ผิดพลาดแบบเมท็อดที่มีความยาวมาก ร่องรอยที่ผิดพลาดแบบคลาสที่มีขนาดใหญ่ และร่องรอยที่ผิดพลาดแบบฟีเจอร์เอนวี เพื่อเปรียบเทียบความสามารถในการบำรุงรักษาของซอร์ซโค้ดระหว่างซอร์ซโค้ดที่ปรับแก้ไขตามลำดับที่ได้จากวิธีการใช้มาตรวัดเชิงวัตถุและอัลกอริทึมละโมบกับซอร์ซโค้ดที่ปรับแก้ไขโดยไม่พิจารณาลำดับการใช้งานวิธีรีแฟคทอริง ซึ่งผลการทดลองนั้นแสดงให้เห็นว่าซอร์ซโค้ดที่ปรับแก้ไขตามลำดับที่ได้จากวิธีการใช้มาตรวัดเชิงวัตถุและอัลกอริทึมละโมบนั้นจะมีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์มากกว่าซอร์ซโค้ดที่ปรับแก้ไขโดยไม่พิจารณาลำดับการใช้งานวิธีรีแฟคทอริง

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมซอฟต์แวร์ ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก .....

ปีการศึกษา 2556

5470991121 : MAJOR SOFTWARE ENGINEERING
KEYWORDS: REFACTORING / REFACTORING SEQUENCING / SOFTWARE MAINTAINABILITY / GREEDY ALGORITHM

RATAPONG WONGPIANG: ORDERING REFACTORING USAGE FOR CODE CHANGING USING OBJECT ORIENTED METRICS AND GREEDY ALGORITHM. ADVISOR: ASSOC. PROF. DR PORNSIRI MUENCHAISRI, 150 pp.

Refactoring is the process of changing the internal structure of software but it preserves external behavior of software. To improve software maintainability, we apply several refactoring techniques to source code. Applying different sequence of refactoring techniques to different parts of source code results in different code changes and different level of software maintainability. This thesis proposes an approach for ordering refactoring techniques usage for code changing using Greedy Algorithm. To get optimal software maintainability, the thesis creates possible sequences of refactoring techniques usage and apply each refactoring techniques to source code. Greedy Algorithm is used to separate the optimal sequence of refactoring techniques usage from possible sequences of refactoring techniques. In the experiment, the thesis evaluates the approach with source code containing Long Method, Large Class and Feature Envy bad smell by comparing the changed source code result between applying the approach and without ordering refactoring techniques usage. The compared results show that the changed source code by applying the approach can improve software maintainability better than the changed source code without ordering refactoring techniques usage.

Department: Computer Engineering Student's Signature .....
Field of Study: Software Engineering Advisor's Signature .....
Academic Year: 2013

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้ได้รับคำแนะนำและแนวทางในการทำวิจัยจากอาจารย์ที่ปรึกษา รองศาสตราจารย์ ดร.พรศิริ หมื่นไชยศรี ที่คอยช่วยเหลือให้ความรู้ทางด้านที่จำเป็นต่อวิทยานิพนธ์ และคำปรึกษาที่สามารถแก้ไขปัญหาต่างๆ ที่เกิดขึ้นได้ในระหว่างการทำงานวิจัย อีกทั้งยังแนะนำการศึกษาต่อในระดับปริญญาเอกในอนาคต และขอบคุณรองศาสตราจารย์ ดร.ทวิตีย์ เสนีวงศ์ ประธานกรรมการ, ผู้ช่วยศาสตราจารย์ นครทิพย์ พร้อมพูล กรรมการ และผู้ช่วยศาสตราจารย์ ดร. ทรงศักดิ์ รองวิริยะพานิช กรรมการภายนอกมหาวิทยาลัย ที่สละเวลาอันมีค่าเป็นคณะกรรมการสอบวิทยานิพนธ์ และให้คำชี้แนะในการปรับปรุงให้วิทยานิพนธ์ฉบับนี้สมบูรณ์มากยิ่งขึ้น ขอบคุณพี่พนิตา เมนะเนตร ที่ให้คำแนะนำสำหรับการเริ่มศึกษางานวิจัยทางด้านการทำรีแฟคทอริงในการปรับแก้ไขโค้ด

ขอขอบคุณจุฬาลงกรณ์มหาวิทยาลัยที่ให้ความรู้ ประสบการณ์ที่ดี และโอกาสในการศึกษาหาความรู้เพิ่มเติมเพื่อนำไปใช้ในการทำงานและศึกษาในระดับปริญญาเอกในอนาคต ขอบคุณเพื่อนๆ พี่ๆ ทุกท่าน ที่คอยเป็นกำลังใจเสมอตลอดช่วงเวลาที่ได้ศึกษาในรั้วมหาวิทยาลัยแห่งนี้ ได้ใช้เวลาในช่วงที่สนุกร่วมกัน และลำบากร่วมด้วยกัน จะเก็บความทรงจำเหล่านี้ไว้เป็นอย่างดี

และสุดท้าย ขอขอบคุณทุกคนในครอบครัวที่ให้โอกาส, คอยสนับสนุน และให้กำลังใจในการเรียนต่อในระดับปริญญาโท และการทำงานวิจัยเสมอมา

สารบัญ

หน้า

บทคัดย่อภาษาไทย ..... ง

บทคัดย่อภาษาอังกฤษ ..... จ

กิตติกรรมประกาศ ..... ฉ

สารบัญ ..... ช

สารบัญตาราง ..... ฎ

สารบัญภาพ ..... ณ

บทที่ 1 บทนำ 21

1.1 ความสำคัญและที่มาปัญหา 21

1.2 วัตถุประสงค์ของงานวิจัย 22

1.3 ขอบเขตงานวิจัย 22

1.4 ขั้นตอนและวิธีดำเนินงานวิจัย 23

1.5 คุณค่าทางวิชาการ 23

1.6 ผลงานตีพิมพ์จากวิทยานิพนธ์ 24

บทที่ 2 ทฤษฎีและงานวิจัยที่เกี่ยวข้อง 25

2.1 ทฤษฎีที่เกี่ยวข้อง 25

2.1.1 รีแฟคทอริง (Refactoring) 25

2.1.2 ร่องรอยที่ผิดพลาด (Bad Smell) 27

2.1.3 อัลกอริทึมละโมบ (Greedy Algorithm) 33

2.1.4 อัลกอริทึมค้นหาแนวกว้าง (Breadth First Search) 34

2.1.5 อัลกอริทึมปีนเขา (Hill Climbing) 35

2.1.6 อัลกอริทึมเอสตาร์ (A* - A Star) 36

2.1.7 การวัดซอฟต์แวร์เชิงวัตถุ (Object Oriented Software Measurement) 37

2.2 งานวิจัยที่เกี่ยวข้อง 39

2.2.1 Bad-Smell Detection For Refactoring Using Object-Oriented Software Metrics [4] 39

2.2.2 Facilitating Software Refactoring with Appropriate Resolution Order of Bad Smell [9] 40

2.2.3 Analysing Refactoring Dependencies Using Graph Transformation [10] 40

2.2.4 A Constrain Programming Approach to Conflict-aware Scheduling of Prioritized Code Clone Refactoring [11] 41

2.2.5 Searching for Opportunities of Refactoring Sequences : Reducing the Search Space [12] 41

2.2.6 Assessment of Maintainability Metrics for Object-Oriented Software System [13] 42

2.2.7 Size and Frequency of Class Change from a Refactoring Perspective [14] 43

บทที่ 3 วิธีการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด โดยใช้มาตรวัดเชิงวัตถุและอัลกอริทึมละโมบ 44

3.1 ขั้นตอนการค้นหาร่องรอยที่ผิดพลาด..... 45

3.2 ขั้นตอนการประยุกต์ใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด..... 54

3.2.1 เส้นทางที่เป็นทางเลือกสำหรับการใช้งานวิธีรีแฟคทอริง (Candidate refactoring techniques path)..... 54

3.2.2 เส้นทางที่เกิดจากการรวมกันของวิธีรีแฟคทอริงมากกว่า 1 วิธี (Combining refactoring techniques path) 55

3.3 ขั้นตอนการคำนวณหาค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ของซอร์ซโค้ด 59

3.4 ขั้นตอนการเลือกลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด..... 61

3.5 ขั้นตอนการประเมินผลลัพธ์ของลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด 69

บทที่ 4 การประเมินผลลัพธ์ของลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด 71

4.1 ความสามารถในการบำรุงรักษาซอฟต์แวร์ของซอร์ซโค้ดภายหลังจากการปรับแก้ไข..... 72

4.1.1 การเปรียบเทียบค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ของซอร์ซโค้ดระหว่างซอร์ซโค้ดที่ถูกปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบกับซอร์ซโค้ดที่ปรับแก้ไขโดยไม่พิจารณาการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด..... 72

4.1.2 ตรวจสอบซอร์ซโค้ดภายหลังจากการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธี รีแฟคทอริงในการปรับแก้ไขโค้ดนั้นมีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์สูงสุดหรือไม่ 79

4.2 เวลาที่ใช้ในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดด้วยอัลกอริทึมละโมบ87

หน้า

4.3 ความถูกต้องของซอร์ซโค้ดภายหลังจากการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ 93
บทที่ 5 การออกแบบและพัฒนาเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริงโดยใช้มาตรวัดเชิงวัตถุและอัลกอริทึมละโมบ 102
5.1 ส่วนของการออกแบบเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริง 102
5.1.1 สถาปัตยกรรมของเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริง 102
5.1.2 ฟังก์ชันการทำงานหลักของเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริง 107
5.1.3 โครงสร้างคลาสการทำงานหลักของเครื่องมือช่วยในการจัดลำดับการใช้งานวิธี รีแฟคทอริงในส่วนของการจัดลำดับการใช้งานวิธีรีแฟคทอริง 108
5.1.4 แผนภาพซีเควนของเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริง 112
5.2 ส่วนของรายละเอียดเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริง 116
5.2.1 โปรแกรมอีคลิปส์ 116
5.2.2 ส่วนของการจัดลำดับการใช้งานวิธีรีแฟคทอริง 119
บทที่ 6 สรุปผลการวิจัยและข้อเสนอแนะ 125
6.1 สรุปผลการวิจัย 125
6.2 ข้อจำกัด 126
6.3 แนวทางการวิจัยต่อไป 126
รายการอ้างอิง 128
ภาคผนวก 130
ภาคผนวก ก รายละเอียดการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด ด้วยอัลกอริทึมแบบอื่นๆ 131
1. อัลกอริทึมค้นหาแนวกว้าง (Breadth First Search) [16] 131
2. อัลกอริทึมปีนเขา (Hill Climbing) [17] 132
3. อัลกอริทึมเอสตาร์ (A* - A Star) [18] 133
ภาคผนวก ข รายละเอียดซอร์ซโค้ดระบบคำนวณค่าไฟฟ้า 135
1. รายละเอียดซอร์ซโค้ดก่อนการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ 137
2. รายละเอียดซอร์ซโค้ดหลังการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ 138

3. รายละเอียดค่ามาตรวัดของการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ 141
ภาคผนวก ค การติดตั้งเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด 144
1. โปรแกรมอีคลิปส์ 144
2. ปลั๊กอินเจดีโอโดแรนต์ 145
3. ปลั๊กอินเมทริก 146
4. เครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด 147
ประวัติผู้เขียนวิทยานิพนธ์ 150

สารบัญตาราง

หน้า

ตารางที่ 1 แสดงรายละเอียดร่องรอยที่ผิดพลาด [1]..... 28
ตารางที่ 2 แสดงวิธีรีแฟคทอริงและรายละเอียดสำหรับแก้ไขร่องรอยที่ผิดพลาด แบบเมท็อดที่มีความยาวมาก [1]..... 29
ตารางที่ 3 แสดงวิธีรีแฟคทอริงและรายละเอียดสำหรับแก้ไขร่องรอยที่ผิดพลาด แบบคลาสที่มีขนาดใหญ่ [1]..... 31
ตารางที่ 4 แสดงวิธีรีแฟคทอริงและรายละเอียดสำหรับแก้ไขร่องรอยที่ผิดพลาด แบบฟีเจอร์เอนวี [1] 32
ตารางที่ 5 แสดงความสัมพันธ์ระหว่างมาตรวัดซีเคกับความสามารถในการบำรุงรักษาซอฟต์แวร์ [13] 42
ตารางที่ 6 แสดงรายละเอียดของการคำนวณค่ามาตรวัดที่ใช้พิจารณาค่าความสามารถในการบำรุงรักษาซอฟต์แวร์..... 60
ตารางที่ 7 แสดงรายละเอียดค่าของมาตรวัดสำหรับเส้นทาง การปรับแก้ไขเมท็อด Statement ในลำดับแบบที่ P1,P2,P3..... 61
ตารางที่ 8 แสดงค่ามาตรวัดของเมท็อด Statement ของคลาส Customer ก่อนการปรับแก้ไขโค้ดด้วยวิธีรีแฟคทอริง..... 64
ตารางที่ 9 แสดงค่ามาตรวัดของเมท็อด Statement ของคลาส Customer สำหรับการปรับแก้ไขโค้ดรอบที่ 1 65
ตารางที่ 10 แสดงค่ามาตรวัดของเมท็อด Statement ของคลาส Customer สำหรับการปรับแก้ไขโค้ดรอบที่ 2 67
ตารางที่ 11 แสดงค่ามาตรวัดของเมท็อด Statement ของคลาส Customer สำหรับการปรับแก้ไขโค้ดรอบที่ 3 68
ตารางที่ 12 แสดงค่ามาตรวัดของเมท็อด Statement ของคลาส Customer ภายหลังการปรับแก้ไขตามลำดับการใช้งานวิธีรีแฟคทอริง 69
ตารางที่ 13 ลักษณะความแตกต่างระหว่างซอร์ซโค้ดนำเข้า 2 โปรแกรม 71
ตารางที่ 14 แสดงค่ามาตรวัดของเมท็อด Statement ของคลาส Customer ภายหลังการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ..... 73
ตารางที่ 15 แสดงค่ามาตรวัดของเมท็อด Statement ของคลาส Customer ภายหลังการปรับแก้ไขด้วยวิธีรีแฟคทอริงโดยไม่พิจารณาลำดับการใช้งานวิธีรีแฟคทอริง 73

หน้า

ตารางที่ 16 แสดงการเปรียบเทียบค่ามาตรวัดของเมท็อด Statement ของคลาส Customer ภายหลังจากการปรับแก้ได้วยวิธีรีแฟคทอริง ระหว่างการพิจารณาและไม่พิจารณาการจัดลำดับวิธีรีแฟคทอริง..... 74

ตารางที่ 17 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส DisabilitySite ภายหลังการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ..... 76

ตารางที่ 18 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส ResidentialSite ภายหลังการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ..... 76

ตารางที่ 19 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส DisabilitySite ภายหลังการปรับแก้ไขโดยไม่พิจารณาการจัดลำดับการใช้งานวิธีรีแฟคทอริง..... 76

ตารางที่ 20 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส ResidentialSite ภายหลังการปรับแก้ไขโดยไม่พิจารณาการจัดลำดับการใช้งานวิธีรีแฟคทอริง..... 77

ตารางที่ 21 แสดงการเปรียบเทียบค่ามาตรวัดของเมท็อด Charge ของคลาส DisabilitySite ภายหลังจากการปรับแก้ไขด้วยวิธีรีแฟคทอริง ระหว่างการพิจารณาและไม่พิจารณาการจัดลำดับวิธีรีแฟคทอริง..... 77

ตารางที่ 22 แสดงการเปรียบเทียบค่ามาตรวัดของเมท็อด Charge ของคลาส ResidentialSite ภายหลังจากการปรับแก้ไขด้วยวิธีรีแฟคทอริง ระหว่างการพิจารณาและไม่พิจารณาการจัดลำดับวิธีรีแฟคทอริง..... 78

ตารางที่ 23 แสดงรายละเอียดของตำแหน่งที่ต้องปรับแก้ไขและ วิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขเมท็อด Statement ของคลาส Customer..... 80

ตารางที่ 24 แสดงมาตรวัดของแต่ละลำดับการใช้งานวิธีรีแฟคทอริงในการแก้ไขเมท็อด Statement โดยเลือกแก้ไขในตำแหน่งที่ P1,P2,P3 ตามลำดับ..... 81

ตารางที่ 25 แสดงมาตรวัดของแต่ละลำดับการใช้งานวิธีรีแฟคทอริงในการแก้ไขเมท็อด Statement โดยเลือกแก้ไขในตำแหน่งที่ P1,P3,P2 ตามลำดับ..... 81

ตารางที่ 26 แสดงมาตรวัดของแต่ละลำดับการใช้งานวิธีรีแฟคทอริงในการแก้ไขเมท็อด Statement โดยเลือกแก้ไขในตำแหน่งที่ P2,P1,P3 ตามลำดับ..... 82

ตารางที่ 27 แสดงมาตรวัดของแต่ละลำดับการใช้งานวิธีรีแฟคทอริงในการแก้ไขเมท็อด Statement โดยเลือกแก้ไขในตำแหน่งที่ P2,P3,P1 ตามลำดับ..... 82

ตารางที่ 28 แสดงมาตรวัดของแต่ละลำดับการใช้งานวิธีรีแฟคทอริงในการแก้ไขเมท็อด Statement โดยเลือกแก้ไขในตำแหน่งที่ P3,P1,P2 ตามลำดับ..... 83

ตารางที่ 29 แสดงมาตรวัดของแต่ละลำดับการใช้งานวิธีรีแฟคทอริงในการแก้ไขเมท็อด Statement โดยเลือกแก้ไขในตำแหน่งที่ P3,P2,P1 ตามลำดับ..... 83

หน้า

ตารางที่ 30 แสดงรายละเอียดของตำแหน่งที่ต้องปรับแก้ไขและ วิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขเมท็อด Charge ของคลาส DisabilitySite 85
ตารางที่ 31 แสดงมาตรวัดของแต่ละลำดับในการปรับแก้ไขเมท็อด Charge ของคลาส DisabilitySite 85
ตารางที่ 32 แสดงรายละเอียดของตำแหน่งที่ต้องปรับแก้ไขและ วิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขเมท็อด Charge ของคลาส ResidentialSite..... 86
ตารางที่ 33 แสดงมาตรวัดของแต่ละลำดับในการปรับแก้ไขเมท็อด Charge ของคลาส ResidentialSite 86
ตารางที่ 34 แสดงการเปรียบเทียบรายละเอียดต่างๆ ระหว่างอัลกอริทึมทั้ง 4 แบบ ในการจัดลำดับการใช้งานวิธีรีแฟคทอริง 87
ตารางที่ 35 แสดงค่ามาตรวัดของคลาส Customer ภายหลังจากปรับแก้ไขด้วยการจัดลำดับ วิธีรีแฟคทอริงด้วยอัลกอริทึมแบบต่างๆ..... 88
ตารางที่ 36 แสดงจำนวนโหนดทั้งหมดที่ใช้ในการจัดลำดับการใช้งานวิธีรีแฟคทอริง ในการปรับแก้ไขเมท็อด Statement ของคลาส Customer โดยการใช้อัลกอริทึมแบบต่างๆ 89
ตารางที่ 37 แสดงค่ามาตรวัดของคลาส DisabilitySite ภายหลังจากปรับแก้ไขเมท็อด Charge ด้วยการจัดลำดับวิธีรีแฟคทอริงด้วยอัลกอริทึมแบบต่างๆ..... 90
ตารางที่ 38 แสดงจำนวนโหนดทั้งหมดที่ใช้ในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขเมท็อด Charge ของคลาส DisabilitySite โดยการใช้อัลกอริทึมแบบต่างๆ..... 91
ตารางที่ 39 แสดงค่ามาตรวัดของคลาส ResidentialSite ภายหลังจากปรับแก้ไขเมท็อด Charge ด้วยการจัดลำดับวิธีรีแฟคทอริงด้วยอัลกอริทึมแบบต่างๆ 91
ตารางที่ 40 แสดงจำนวนโหนดทั้งหมดที่ใช้ในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขเมท็อด Charge ของคลาส ResidentialSite โดยการใช้อัลกอริทึมแบบต่างๆ..... 92
ตารางที่ 41 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส Customer ก่อนการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง..... 93
ตารางที่ 42 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส Rental ก่อนการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง..... 94
ตารางที่ 43 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส Movie ก่อนการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง..... 94
ตารางที่ 44 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส Customer หลังการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง..... 94

หน้า

ตารางที่ 45 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส Rental หลังการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง 95
ตารางที่ 46 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส Movie หลังการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง 95
ตารางที่ 47 แสดงรายละเอียดกรณีทดสอบส่วนของการแสดงรายละเอียด ของภาพยนตร์ที่ลูกค้าเช่า 96
ตารางที่ 48 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณค่าเช่าภาพยนตร์แต่ละเรื่อง 96
ตารางที่ 49 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณค่าความถี่สะสมในการเช่าภาพยนตร์แต่ละเรื่อง 96
ตารางที่ 50 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณผลรวมค่าเช่าภาพยนตร์ของลูกค้าในการเช่าครั้งหนึ่งๆ 97
ตารางที่ 51 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณผลรวมค่าความถี่สะสมในการเช่าภาพยนตร์ของลูกค้าในการเช่าครั้งหนึ่งๆ 97
ตารางที่ 52 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส DisabilitySite ก่อนการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง 98
ตารางที่ 53 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส DisabilitySite หลังการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง 98
ตารางที่ 54 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส ResdentialSite ก่อนการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง 99
ตารางที่ 55 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส ResidentialSite หลังการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง 99
ตารางที่ 56 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณค่าไฟฟ้าของรายการ ของคลาส DisabilitySite 100
ตารางที่ 57 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณค่ามาตรวัดไฟฟ้า ของการใช้งานรายการล่าสุดของคลาส DisabilitySite 100
ตารางที่ 58 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณค่าอัตราช่วงหน้าร้อน ของการใช้งานรายการล่าสุดของคลาส DisabilitySite 100
ตารางที่ 59 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณค่าไฟฟ้าของรายการ ของคลาส ResidentialSite 100
ตารางที่ 60 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณค่ามาตรวัดไฟฟ้า ของการใช้งานรายการล่าสุดของคลาส ResidentialSite 101

หน้า

ตารางที่ 61 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณค่าอัตราช่วงหน้าร้อน ของการใช้งานรายการล่าสุดของคลาส ResidentialSite 101

ตารางที่ 62 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส DisabilitySite สำหรับการปรับแก้ไขโค้ดรอบที่ 1 141

ตารางที่ 63 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส DisabilitySite สำหรับการปรับแก้ไขโค้ดรอบที่ 2 142

ตารางที่ 64 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส DisabilitySite เมื่อสิ้นสุดการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ 142

ตารางที่ 65 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส ResidentialSite สำหรับการปรับแก้ไขโค้ดรอบที่ 1 142

ตารางที่ 66 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส ResidentialSite สำหรับการปรับแก้ไขโค้ดรอบที่ 2 143

ตารางที่ 67 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส ResidentialSite เมื่อสิ้นสุดการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ 143

สารบัญภาพ

หน้า

ภาพที่ 1 แผนภาพแอคทิวิตีแสดงกระบวนการทำรีแฟคทอริง [6] 26
ภาพที่ 2 ซอร์ซโค้ดที่มีร่องรอยที่ผิดพลาดแบบเมท็อดที่มีความยาวมาก 30
ภาพที่ 3 ซอร์ซโค้ดที่มีร่องรอยที่ผิดพลาดแบบคลาสที่มีขนาดใหญ่ 31
ภาพที่ 4 ซอร์ซโค้ดที่มีร่องรอยที่ผิดพลาดแบบฟีเจอร์เอนวี 32
ภาพที่ 5 ตัวอย่างขั้นตอนการค้นหาเส้นทางโดยการใช้อัลกอริทึมละโมบ [7] 34
ภาพที่ 6 ตัวอย่างขั้นตอนการค้นหาเส้นทางโดยการใช้อัลกอริทึมค้นหาแนวกว้าง [7] 35
ภาพที่ 7 ตัวอย่างขั้นตอนการค้นหาเส้นทางโดยการใช้อัลกอริทึมปีนเขา [7] 36
ภาพที่ 8 แผนภาพแอคทิวิตีแสดงขั้นตอนการจัดลำดับการใช้งานวิธีรีแฟคทอริง ในการปรับแก้ไขโค้ดโดยใช้มาตรวัดเชิงวัตถุและอัลกอริทึมละโมบ 44
ภาพที่ 9 แผนภาพแอคทิวิตีแสดงขั้นตอนการค้นหาร่องรอยที่ผิดพลาดของโปรแกรมอีคลิปส์ ที่ประยุกต์ใช้งานปลั๊กอินเจดีโดแรนต์ 46
ภาพที่ 10 แผนภาพความสัมพันธ์ของคลาสของระบบเช่าภาพยนตร์ 47
ภาพที่ 11 ซอร์ซโค้ดเมท็อด Statement ที่จะทำการปรับแก้ไขโดยวิธีรีแฟคทอริง 48
ภาพที่ 12 ซอร์ซโค้ดเมท็อด Statement ที่ปรับแก้ไขในตำแหน่งที่ 1 ด้วยวิธีรีแฟคทอริงแบบ Extract Method (R1) 48
ภาพที่ 13 ซอร์ซโค้ดเมท็อด Statement ที่ปรับแก้ไขในตำแหน่งที่ 1 ด้วยวิธีรีแฟคทอริงแบบ Move Method (R2) 49
ภาพที่ 14 ซอร์ซโค้ดเมท็อด Statement ที่ปรับแก้ไขในตำแหน่งที่ 2 ด้วยวิธีรีแฟคทอริงแบบ Extract Method (R3) 50
ภาพที่ 15 ซอร์ซโค้ดเมท็อด Statement ที่ปรับแก้ไขในตำแหน่งที่ 2 ด้วยวิธีรีแฟคทอริงแบบ Move Method (R4) 51
ภาพที่ 16 ซอร์ซโค้ดเมท็อด Statement ที่ปรับแก้ไขในตำแหน่งที่ 3 ด้วยวิธีรีแฟคทอริงแบบ Extract Method (R5) 52
ภาพที่ 17 ซอร์ซโค้ดเมท็อด Statement ที่ปรับแก้ไขในตำแหน่งที่ 3 ด้วยวิธีรีแฟคทอริงแบบ Move Method (R6) 53
ภาพที่ 18 แผนภาพต้นไม้แสดงเส้นทางที่เป็นทางเลือกในการใช้งานวิธีรีแฟคทอริง สำหรับปรับแก้ไขเมท็อด A 54

หน้า

ภาพที่ 19 แผนภาพต้นไม้แสดงเส้นทางในการใช้งานวิธีรีแฟคทอริงสำหรับปรับแก้ไขเมท็อด A ที่เกิดจากการรวมกันของวิธีรีแฟคทอริงมากกว่า 1 วิธี 55
ภาพที่ 20 แผนภาพต้นไม้แสดงเส้นทางในการใช้งานวิธีรีแฟคทอริงสำหรับปรับแก้ไขเมท็อด Statement ตามลำดับแบบที่ P1,P2,P3..... 56
ภาพที่ 21 แผนภาพต้นไม้แสดงเส้นทางในการใช้งานวิธีรีแฟคทอริงสำหรับปรับแก้ไขเมท็อด Statement ตามลำดับแบบที่ P1,P3,P2..... 57
ภาพที่ 22 แผนภาพต้นไม้แสดงเส้นทางในการใช้งานวิธีรีแฟคทอริงสำหรับปรับแก้ไขเมท็อด Statement ตามลำดับแบบที่ P2,P1,P3..... 57
ภาพที่ 23 แผนภาพต้นไม้แสดงเส้นทางในการใช้งานวิธีรีแฟคทอริงสำหรับปรับแก้ไขเมท็อด Statement ตามลำดับแบบที่ P2,P3,P1..... 58
ภาพที่ 24 แผนภาพต้นไม้แสดงเส้นทางในการใช้งานวิธีรีแฟคทอริงสำหรับปรับแก้ไขเมท็อด Statement ตามลำดับแบบที่ P3,P1,P2..... 58
ภาพที่ 25 แผนภาพต้นไม้แสดงเส้นทางในการใช้งานวิธีรีแฟคทอริงสำหรับปรับแก้ไขเมท็อด Statement ตามลำดับแบบที่ P3,P2,P1..... 59
ภาพที่ 26 แผนภาพต้นไม้แสดงเส้นทางการเลือกใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด โดยใช้อัลกอริทึมละโมบ 64
ภาพที่ 27 ซอร์ซโค้ดเมท็อด Statement ที่ปรับแก้ไขในตำแหน่งที่ 1 ด้วยวิธีรีแฟคทอริงแบบ R2. 66
ภาพที่ 28 ซอร์ซโค้ดเมท็อด Statement ที่ปรับแก้ไขในตำแหน่งที่ 1 และ 2 ด้วยวิธีรีแฟคทอริงแบบ R2 และ R4 ตามลำดับ 67
ภาพที่ 29 ซอร์ซโค้ดเมท็อด Statement ภายหลังการปรับแก้ไขตามลำดับการใช้งาน วิธีรีแฟคทอริง 70
ภาพที่ 30 สถาปัตยกรรมเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริง 102
ภาพที่ 31 โฟลเดอร์หลักในการจัดเก็บข้อมูลของเครื่องมือช่วยใน การจัดลำดับการใช้งานวิธีรีแฟคทอริง..... 104
ภาพที่ 32 โฟลเดอร์ระบบเช่าภาพยนตร์ที่จัดเก็บซอร์ซโค้ดของระบบ 104
ภาพที่ 33 โฟลเดอร์ระบบเช่าภาพยนตร์แยกตามการปรับแก้ไขด้วยวิธีรีแฟคทอริงแต่ละวิธี 105
ภาพที่ 34 ไฟล์ซอร์ซโค้ดของระบบเช่าภาพยนตร์ที่จัดเก็บภายในโฟลเดอร์movierental_original 105
ภาพที่ 35 ไฟล์รายละเอียดค่ามาตรวัดของซอร์ซโค้ดระบบเช่าภาพยนตร์ ภายหลังจากการปรับแก้ไขด้วยวิธีรีแฟคทอริง..... 106

หน้า

ภาพที่ 36 ตัวอย่างรายละเอียดการจัดเก็บค่ามาตรวัดของซอร์ซโค้ดระบบเช่าภาพยนตร์ 106
ภาพที่ 37 ยูสเคสไดอะแกรมของเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริง 107
ภาพที่ 38 แผนภาพคลาสส่วนของการจัดลำดับการใช้งานวิธีรีแฟคทอริง 108
ภาพที่ 39 รายละเอียดของไฟล์ orderingRefactoringTool.properties 109
ภาพที่ 40 แผนภาพซีเควนของฟังก์ชันระบุร่องรอยที่ผิดพลาดและวิธีรีแฟคทอริง ในการปรับแก้ไขโค้ด 112
ภาพที่ 41 แผนภาพซีเควนของฟังก์ชันคำนวณค่ามาตรวัด 113
ภาพที่ 42 แผนภาพซีเควนของฟังก์ชันจัดลำดับการใช้งานวิธีรีแฟคทอริง 114
ภาพที่ 43 แผนภาพซีเควนของฟังก์ชันดูรายละเอียดค่ามาตรวัดของซอร์ซโค้ดก่อน-หลัง การจัดลำดับการใช้งานวิธีรีแฟคทอริง 115
ภาพที่ 44 แผนภาพซีเควนของฟังก์ชันดูรายละเอียดของซอร์ซโค้ดก่อน-หลัง การใช้งานวิธีรีแฟคทอริง 116
ภาพที่ 45 หน้าจอหลักโปรแกรมอีคลิปส์ 117
ภาพที่ 46 หน้าจอการค้นหาร่องรอยที่ผิดพลาดของโปรแกรมอีคลิปส์ โดยการใช้ปลั๊กอินเจดีโอโดแรนต์ 118
ภาพที่ 47 หน้าจอแสดงการไฮไลต์โค้ดส่วนที่ต้องทำการปรับแก้ไขด้วยวิธีรีแฟคทอริง จากการค้นหาร่องรอยที่ผิดพลาดของโปรแกรมอีคลิปส์ โดยการใช้ปลั๊กอินเจดีโอโดแรนต์ 118
ภาพที่ 48 หน้าจอการคำนวณค่ามาตรวัดของโปรแกรมอีคลิปส์ โดยการใช้ปลั๊กอินเมทริก 119
ภาพที่ 49 หน้าจอการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด 120
ภาพที่ 50 หน้าจอส่วนสำหรับเลือกโปรเจคที่เป็นข้อมูลนำเข้า 121
ภาพที่ 51 หน้าจอส่วนสำหรับแสดงผลรายละเอียดข้อมูลนำเข้า ก่อนการจัดลำดับการใช้งานวิธีรีแฟคทอริง 121
ภาพที่ 52 หน้าจอส่วนสำหรับแสดงรายละเอียดซอร์ซโค้ด ก่อนการจัดลำดับการใช้งานวิธีรีแฟคทอริง 122
ภาพที่ 53 หน้าจอส่วนสำหรับเลือกอัลกอริทึมในการจัดลำดับ การใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด 122
ภาพที่ 54 หน้าจอส่วนสำหรับแสดงผลลัพธ์ภายหลังจากการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดด้วยการใช้อัลกอริทึมละโมบ 123
ภาพที่ 55 หน้าจอส่วนสำหรับแสดงผลลัพธ์ภายหลังจากการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดด้วยการใช้อัลกอริทึมค้นหาแนวกว้าง 123

หน้า

ภาพที่ 56 หน้าจอส่วนสำหรับแสดงผลลัพธ์ค่ามาตรวัดที่ได้จากการ เลือกลำดับการใช้งานวิธีรีแฟคทอริงที่สนใจ 124
ภาพที่ 57 หน้าจอส่วนสำหรับแสดงรายละเอียดซอร์ซโค้ด ภายหลังการจัดลำดับการใช้งานวิธีรีแฟคทอริง 124
ภาพที่ 58 แผนภาพความสัมพันธ์ของคลาสของระบบเช่าภาพยนตร์ 135
ภาพที่ 59 ซอร์ซโค้ดเมท็อด Charge ของคลาส DisabilitySite ก่อนการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง 137
ภาพที่ 60 ซอร์ซโค้ดเมท็อด Charge ของคลาส ResidentialSite ก่อนการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง 138
ภาพที่ 61 ซอร์ซโค้ดเมท็อด Charge ของคลาส DisabilitySite ภายหลังจากปรับแก้ไขที่ตำแหน่งที่ P2 ด้วยวิธีรีแฟคทอริงแบบ R2 138
ภาพที่ 62 ซอร์ซโค้ดเมท็อด Charge ของคลาส DisabilitySite ภายหลังจากปรับแก้ไขที่ตำแหน่งที่ P1 ด้วยวิธีรีแฟคทอริงแบบ R2 และ R1 ตามลำดับ 139
ภาพที่ 63 ซอร์ซโค้ดเมท็อด Charge ของคลาส DisabilitySite ภายหลังจากปรับแก้ไขที่ตำแหน่งที่ P3 ด้วยวิธีรีแฟคทอริงแบบ R2 R1 และ R3 ตามลำดับ 139
ภาพที่ 64 ซอร์ซโค้ดเมท็อด Charge ของคลาส ResidentialSite ภายหลังจากปรับแก้ไขที่ตำแหน่งที่ P5 ด้วยวิธีรีแฟคทอริงแบบ R5 140
ภาพที่ 65 ซอร์ซโค้ดเมท็อด Charge ของคลาส ResidentialSite ภายหลังจากปรับแก้ไขที่ตำแหน่งที่ P4 ด้วยวิธีรีแฟคทอริงแบบ R5 และ R4 ตามลำดับ 140
ภาพที่ 66 ซอร์ซโค้ดเมท็อด Charge ของคลาส ResidentialSite ภายหลังจากปรับแก้ไขที่ตำแหน่งที่ P6 ด้วยวิธีรีแฟคทอริงแบบ R5 R4 และ R6 ตามลำดับ 141
ภาพที่ 67 แสดงซิปไฟล์ของโปรแกรมอีคลิปส์ภายหลังจากดาวโหลดเสร็จสิ้น 144
ภาพที่ 68 แสดงโครงสร้างไฟล์ภายหลังจากการแตกซิบไฟล์ของโปรแกรมอีคลิปส์ 144
ภาพที่ 69 แสดงการเริ่มใช้งานโปรแกรมอีคลิปส์ 145
ภาพที่ 70 แสดงไฟล์จาของปลั๊กอินเจดีโอโดแรนต์ภายหลังจากดาวโหลดเสร็จสิ้น 145
ภาพที่ 71 แสดงการวางไฟล์จาของปลั๊กอินเจดีโอโดแรนต์เพื่อใช้ งานร่วมกับโปรแกรมอีคลิปส์ 146
ภาพที่ 72 แสดงปลั๊กอินเจดีโอโดแรนต์พร้อมใช้งานร่วมกับโปรแกรมอีคลิปส์ 146
ภาพที่ 73 แสดงไฟล์จาของปลั๊กอินเมทริกภายหลังจากดาวโหลดเสร็จสิ้น 146
ภาพที่ 74 แสดงไฟล์จาของปลั๊กอินเมทริกภายหลังวางไว้ภายในโฟลเดอร์ plugin ของโปรแกรมอีคลิปส์ 147

หน้า

ภาพที่ 75 แสดงปลั๊กอินเมทริกพร้อมใช้งานร่วมกับโปรแกรมอีคลิปส์..... 147
ภาพที่ 76 แสดงการวางไฟล์ OrderingRefactoringToolWeb.war ไว้ภายในเซอร์เวอร์อาพาเช่ทอมแคท 148
ภาพที่ 77 แสดงไฟล์ startup.bat สำหรับใช้ในการเริ่มการทำงานของ เซอร์เวอร์อาพาเช่ทอมแคท 148
ภาพที่ 78 แสดงการรันการทำงานของเซอร์เวอร์อาพาเช่ทอมแคท 149
ภาพที่ 79 แสดงระบบช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริง ในการปรับแก้ไขโค้ดพร้อมใช้งาน 149

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาปัญหา

ร่องรอยที่ผิดพลาด (Bad-Smell) เป็นลักษณะการออกแบบหรือการเขียนโค้ดที่ไม่ดีของผู้พัฒนาซอฟต์แวร์ ตัวอย่างลักษณะของซอร์ซโค้ดที่มีร่องรอยที่ผิดพลาด เช่น โค้ดซ้ำซ้อนกันในหลายตำแหน่งภายในโปรแกรม ขนาดของโค้ดต่อหนึ่งฟังก์ชันการทำงานที่มีความยาวมากเกินไป ขนาดของคลาสใหญ่เกินไป เป็นต้น ซอฟต์แวร์ที่มีลักษณะร่องรอยที่ผิดพลาดนั้นถึงแม้ว่าจะสามารถทำงานได้ถูกต้องตามความต้องการของผู้ใช้งาน แต่การปรับแก้ไขหรือเพิ่มเติมการทำงานในภายหลังจากส่งมอบซอฟต์แวร์หรือการทดสอบระบบจากผู้ใช้งานจะทำได้ค่อนข้างยาก เนื่องจากการเขียนโค้ดที่ไม่เป็นระเบียบดังตัวอย่างข้างต้น ทำให้เกิดปัญหาต่างๆ ขึ้นในการพัฒนาซอฟต์แวร์ เช่น ยากต่อการทำความเข้าใจซอร์ซโค้ดการทำงาน ยากต่อการแก้ไขโค้ดเมื่อพบข้อผิดพลาด ยากต่อการนำซอร์ซโค้ดเดิมกลับมาใช้งานใหม่ ยากต่อการเพิ่มเติมต่อขยายการทำงานใหม่ เป็นต้น จากปัญหาดังกล่าวนั้นเป็นลักษณะที่ทำให้ซอฟต์แวร์ยากต่อการบำรุงรักษา (Maintainability) ดังนั้นเพื่อให้ซอฟต์แวร์ง่ายต่อการบำรุงรักษาจึงต้องมีการกำจัดร่องรอยที่ผิดพลาดที่เกิดขึ้นในการเขียนโค้ด โดยภายหลังจากที่กำจัดร่องรอยที่ผิดพลาดออกแล้วการทำงานของซอฟต์แวร์ยังคงเหมือนเดิมกับก่อนการกำจัดร่องรอยที่ผิดพลาด ซึ่งเทคนิคที่นำมาช่วยในการกำจัดร่องรอยที่ผิดพลาดของการเขียนโค้ด คือ รีแฟคทอริง (Refactoring)

รีแฟคทอริงเป็นการเปลี่ยนแปลงโครงสร้างภายในของซอฟต์แวร์ โดยไม่ทำให้พฤติกรรมการทำงานของซอฟต์แวร์นั้นเปลี่ยนแปลง ทำให้ซอฟต์แวร์นั้นง่ายต่อการบำรุงรักษาในภายหลัง เช่น ง่ายต่อการทำความเข้าใจซอร์ซโค้ดการทำงานของโปรแกรม การเปลี่ยนแปลงการทำงานของซอฟต์แวร์เพื่อรองรับการทำงานใหม่ทำได้ง่าย ใช้เวลาน้อยในการปรับแก้ไขโปรแกรม การค้นหาข้อผิดพลาดของโปรแกรมทำได้ง่าย เป็นต้น การรีแฟคทอริงโค้ดนั้นผู้พัฒนาซอฟต์แวร์จำเป็นต้องเข้าใจการเลือกใช้งานวิธีรีแฟคทอริงก่อน ซึ่งการเลือกใช้งานวิธีรีแฟคทอริงนั้นขึ้นอยู่กับลักษณะของซอร์ซโค้ดที่มีปัญหา โดยมาร์ติน ฟาวเลอร์ (Martin Fowler) ได้ระบุลักษณะร่องรอยที่ผิดพลาดในลักษณะต่างๆ พร้อมทั้งได้แนะนำวิธีรีแฟคทอริงเพื่อช่วยเป็นแนวทางในการแก้ไขร่องรอยที่ผิดพลาดในแต่ละลักษณะที่ได้ระบุไว้ [1] ในปัจจุบันมีงานวิจัยเป็นจำนวนมากได้ศึกษาวิธีในการค้นหาร่องรอยที่ผิดพลาดในระดับของซอร์ซโค้ด [2-4] และได้พัฒนาออกมาเป็นเครื่องมือ [5] เพื่อความสะดวกแก่ผู้พัฒนาระบบในการปรับแก้ไขโค้ด ซึ่งการปรับแก้ไขโค้ดด้วยวิธีรีแฟคทอริงต่างๆ จะมีผลต่อคุณลักษณะของซอร์ซโค้ดภายหลังจากการปรับแก้ไข เช่น ความยาวของซอร์ซโค้ด (Size) ความซับซ้อนของซอร์ซโค้ด (Complexity) เข้าคู่กันระหว่างวัตถุ (Coupling Between Object) ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส (Lack of Cohesion in Methods) เป็นต้น บางซอฟต์แวร์อาจจำเป็นต้องใช้วิธีรีแฟคทอริงหลายวิธีในการปรับแก้ไขซอร์ซโค้ดหลายตำแหน่ง ลำดับในการเลือกใช้งานวิธีรีแฟคทอริงจึงเป็นอีกหนึ่งปัจจัยสำหรับผู้พัฒนาซอฟต์แวร์ในการปรับแก้ไขโค้ด ถ้าผู้พัฒนาซอฟต์แวร์สามารถเลือกลำดับการใช้งานวิธีรีแฟคทอริงได้อย่างถูกต้อง ก็จะทำให้ได้ซอร์ซโค้ดภายหลัง

จากการปรับแก้ไขที่มีค่าความสามารถในการรักษาซอฟต์แวร์ที่มากที่สุด ดังนั้นวิทยานิพนธ์นี้จึงได้นำเสนอวิธีในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด โดยอาศัยมาตรวัดเชิงวัตถุ (Object Oriented Metrics) ในการคำนวณหาค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ และอัลกอริทึมละโมบ (Greedy Algorithm) ในการค้นหาลำดับการใช้งานวิธีรีแฟคทอริงจากลำดับการใช้งานวิธีรีแฟคทอริงที่เป็นไปได้ทั้งหมด (search space) เพื่อหาลำดับการใช้งานวิธีรีแฟคทอริงที่ทำให้ได้ซอร์ซโค้ดภายหลังที่ทำการแก้ไขที่มีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์มากที่สุด และใช้เวลาในการค้นหา (search time) น้อยที่สุด ในส่วนของการประเมินผลวิธีการนั้นจะทำการเปรียบค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ของซอร์ซโค้ดที่ทำการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงกับค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ของซอร์ซโค้ดที่ปรับแก้ไขโดยไม่พิจารณาการจัดลำดับการใช้งานวิธีรีแฟคทอริง ผลลัพธ์ที่ได้จากการเปรียบเทียบนั้นค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ของซอร์ซโค้ดที่ปรับแก้ไขโดยวิธีการจัดลำดับการใช้งานวิธีรีแฟคทอริงควรจะมีค่ามากกว่าค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ของซอร์ซโค้ดที่ปรับแก้ไขโดยไม่พิจารณาการจัดลำดับการใช้งานวิธีรีแฟคทอริง

1.2 วัตถุประสงค์ของงานวิจัย

วิทยานิพนธ์นี้มีวัตถุประสงค์เพื่อออกแบบวิธีการค้นหาและจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดเพื่อให้ได้ซอร์ซโค้ดภายหลังจากการปรับแก้ไขนั้นมีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ที่มากที่สุด และพัฒนาเครื่องมือในที่ช่วยในการค้นหาและจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด

1.3 ขอบเขตงานวิจัย

1. มาตรวัดเชิงวัตถุที่จะนำมาพิจารณาเพื่อหาค่าความสามารถในการบำรุงรักษาซอฟต์แวร์นั้นจะพิจารณาเพียง 3 ตัวเท่านั้น คือ การเข้าคู่กันระหว่างวัตถุ (Coupling Between Objects) ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส (Lack of Cohesion in Method) และมาตรวัดผลรวมค่าความซับซ้อนต่อคลาส (Weighted Method per Class)
2. ลักษณะของซอร์ซโค้ดที่นำมาใช้ในการทดลองนั้นเป็นซอร์ซโค้ดที่มีร่องรอยที่ผิดพลาด 3 ลักษณะ คือ เมท็อดที่มีความยาวมาก (Long Method) คลาสที่มีขนาดใหญ่ (Large Class) และฟีเจอร์เอนวี (Feature Envy)
3. สนใจเฉพาะส่วนของการค้นหาและจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดเท่านั้น โดยในส่วนของการค้นหาร่องรอยที่ผิดพลาดในซอร์ซโค้ดและวิธีรีแฟคทอริงที่ใช้ในการแก้ไขนั้น จะนำเครื่องมือมาช่วยในการค้นหา
4. ในการพิจารณาค่าความสามารถในการบำรุงรักษาซอฟต์แวร์นั้น จะแยกพิจารณาตามมาตรวัดของแต่ละตัว เนื่องจากมาตรวัดทั้ง 3 ตัวที่นำมาใช้ในการทดลองนั้นมีสเกลที่แตกต่างกันจึงไม่สามารถนำมาคำนวณรวมกันเป็นค่าเดียวได้

5. การค้นหาร่องรอยที่ผิดพลาดในซอร์ซโค้ดนั้น จะทำเพียงแค่ครั้งเดียวเท่านั้น คือ ก่อนทำการปรับแก้ไขโค้ดด้วยวิธีรีแฟคทอริง ภายหลังที่ซอร์ซโค้ดต้นฉบับถูกปรับแก้ไขแล้ว จะไม่ทำการค้นหาร่องรอยที่ผิดพลาดอีก เพื่อป้องกันกรณีที่เกิดร่องรอยที่ผิดพลาดขึ้นเรื่อยๆ โดยไม่มีที่สิ้นสุด
6. ใช้ซอร์ซโค้ดจากโอเพนซอร์ซ (Open Source) 2 โปรแกรมในการทดลอง
7. ซอร์ซโค้ดต้นฉบับที่นำมาใช้เป็นข้อมูลนำเข้าต้องเป็นโปรแกรมภาษาจาวาที่คอมไพล์ผ่าน
8. เครื่องมือที่พัฒนาขึ้นสำหรับใช้ในการทดลองการค้นหาและจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดนั้น ใช้โปรแกรมอีคลิปส์ที่ประยุกต์ใช้งานร่วมกับปลั๊กอิน 2 ตัว คือ ปลั๊กอินเจดีโอโดแรนต์และปลั๊กอินเมทริก โดยปลั๊กอินเจดีโอโดแรนต์ทำหน้าที่ในการค้นหาตำแหน่งของร่องรอยที่ผิดพลาดในซอร์ซโค้ด พร้อมทั้งระบุวิธีรีแฟคทอริงที่ใช้งานในการปรับแก้ไขโค้ด และปลั๊กอินเมทริกทำหน้าที่ในการคำนวณหาค่าของมาตรวัดของซอร์ซโค้ดภายหลังจากการปรับแก้ไข

1.4 ขั้นตอนและวิธีดำเนินงานวิจัย

1. ศึกษาวิธีรีแฟคทอริงจากหนังสือและงานวิจัยต่างๆ เพื่อทำความเข้าใจเป้าหมายของวิธีรีแฟคทอริงแต่ละวิธีในการปรับแก้ไขโค้ด
2. ศึกษาร่องรอยที่ผิดพลาดถึงปัญหาและผลกระทบที่เกิดขึ้นจากร่องรอยที่ผิดพลาด เพื่อนำไปประยุกต์ใช้กับวิธีรีแฟคทอริง
3. ศึกษาและเลือกมาตรวัดเชิงวัตถุเพื่อใช้ในการวัดค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ของซอรซ์โค้ดที่ปรับแก้ไขด้วยวิธีรีแฟคทอริง
4. ศึกษาอัลกอริทึมละโมบ เพื่อใช้ในการออกแบบวิธีการจัดลำดับกลุ่มวิธีรีแฟคทอริง
5. ศึกษาเครื่องมือและงานวิจัยสำหรับค้นหาวิธีรีแฟคทอริงจากซอร์ซโค้ด
6. ศึกษาลักษณะของซอร์ซโค้ดที่มีร่องรอยที่ผิดพลาดที่ใช้เป็นข้อมูลนำเข้า
7. ออกแบบการทดลองเพื่อค้นหาลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด
8. ทดลองและเก็บรวบรวมข้อมูล
9. ประเมินผลและสรุปผลจากการทดลองในการเรียงลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด
10. สรุปผลและเรียบเรียงวิทยานิพนธ์

1.5 คุณค่าทางวิชาการ

1. วิธีการที่พัฒนาขึ้นสามารถปรับปรุงการใช้งานในวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดให้มีประสิทธิภาพมากขึ้น ทำให้คุณภาพของซอร์ซโค้ดภายหลังจากการปรับแก้ไขมีความสามารถในการบำรุงรักษาซอฟต์แวร์มากขึ้น

2. วิธีการที่พัฒนาขึ้นสามารถช่วยผู้พัฒนาซอฟต์แวร์ในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด เพื่อความสะดวกในการปรับแก้ไขโค้ด
3. ได้เครื่องมือช่วยในการค้นหาและจัดลำดับวิธีการรีแฟคทอริงในการปรับแก้โค้ดสำหรับโปรแกรมภาษาจาวา

1.6 ผลงานตีพิมพ์จากวิทยานิพนธ์

1. หัวเรื่อง “Selecting Sequence of Refactoring Techniques Usage for Code Changing Using Greedy Algorithm.” โดย รัฐพงษ์ วงศ์เปียง และ พรศิริ หมื่นไชยศรี ในบันทึกการประชุม “The 2013 IEEE 4th International Conference on Electronics Information and Communication (ICEIEC 2013)” ซึ่งจัดขึ้น ณ ปักกิ่ง ประเทศจีน ระหว่างวันที่ 15-17 พฤศจิกายน 2556
2. หัวเรื่อง “Comparing Heuristic Search Methods for Selecting Sequence of Refactoring Techniques Usage for Code Changing.” โดย รัฐพงษ์ วงศ์เปียง และ พรศิริ หมื่นไชยศรี ในบันทึกการประชุม “The 2014 IAENG International Conference on Software Engineering” กลุ่มการประชุม “The International MultiConference of Engineers and Computer Scientists 2014 (IMECS2014)” ซึ่งจัดขึ้น ณ เกาลูน ประเทศฮองกง ระหว่างวันที่ 12-14 มีนาคม 2557

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ทฤษฎีที่เกี่ยวข้อง

2.1.1 รีแฟคทอริง (Refactoring)

รีแฟคทอริง หมายถึง กระบวนการเปลี่ยนแปลงโครงสร้างของซอฟต์แวร์ โดยที่การเปลี่ยนแปลงนั้นไม่กระทบกับพฤติกรรมภายนอก (External behavior) เพื่อทำให้ซอฟต์แวร์นั้นง่ายต่อการบำรุงรักษา ง่ายต่อการทำความเข้าใจการทำงานของซอฟต์แวร์ และสามารถนำส่วนของซอฟต์แวร์หรือคอมโพเนนต์ (Component) นำกลับมาใช้งานใหม่ (reuse) ได้ง่าย

มาร์ติน ฟาวเลอร์ (Martin Fowler) นำเสนอวิธีรีแฟคทอริงในการเปลี่ยนแปลงซอร์ซโค้ดทั้งหมด 72 แบบ โดยแบบออกเป็น 7 กลุ่มตามลักษณะการเปลี่ยนแปลง ได้แก่

1. กลุ่ม Composing Method

เป็นวิธีรีแฟคทอริงสำหรับการเปลี่ยนแปลงเมท็อด ตัวอย่างวิธีรีแฟคทอริง ได้แก่ Extract Method, Inline Temp, Replace Temp with Query, Substitute Algorithm เป็นต้น

2. กลุ่ม Moving Feature between Objects

เป็นวิธีรีแฟคทอริงสำหรับเปลี่ยนแปลงหน้าที่ความรับผิดชอบจากคลาสหนึ่งไปยังคลาสอื่น ตัวอย่างวิธีรีแฟคทอริงได้แก่ Move Method, Move Field เป็นต้น

3. กลุ่ม Organizing Data

เป็นวิธีรีแฟคทอริงสำหรับการจัดการข้อมูล ตัวอย่างวิธีรีแฟคทอริง ได้แก่ Self Encapsulate Field, Replace Data Value with Object, Change Value to Reference, Change Reference to Value, Replace Array with Object, Encapsulate Field เป็นต้น

4. กลุ่ม Simplify Conditional Expressions

เป็นวิธีรีแฟคทอริงสำหรับเปลี่ยนแปลงส่วนเงื่อนไข ตัวอย่างวิธีรีแฟคทอริง ได้แก่ Decompose Conditional, Consolidate Conditional Expression, Replace Conditional with Polymorphism, Introduction Null Objects เป็นต้น

5. กลุ่ม Making Method Calls Simpler

เป็นวิธีรีแฟคทอริงสำหรับทำให้เรียกใช้งานเมท็อดได้ง่ายขึ้น ตัวอย่างวิธีรีแฟคทอริง ได้แก่ Rename Method, Add Parameter, Remove Parameter, Preserve Whole Object, Introduce Parameter Object เป็นต้น

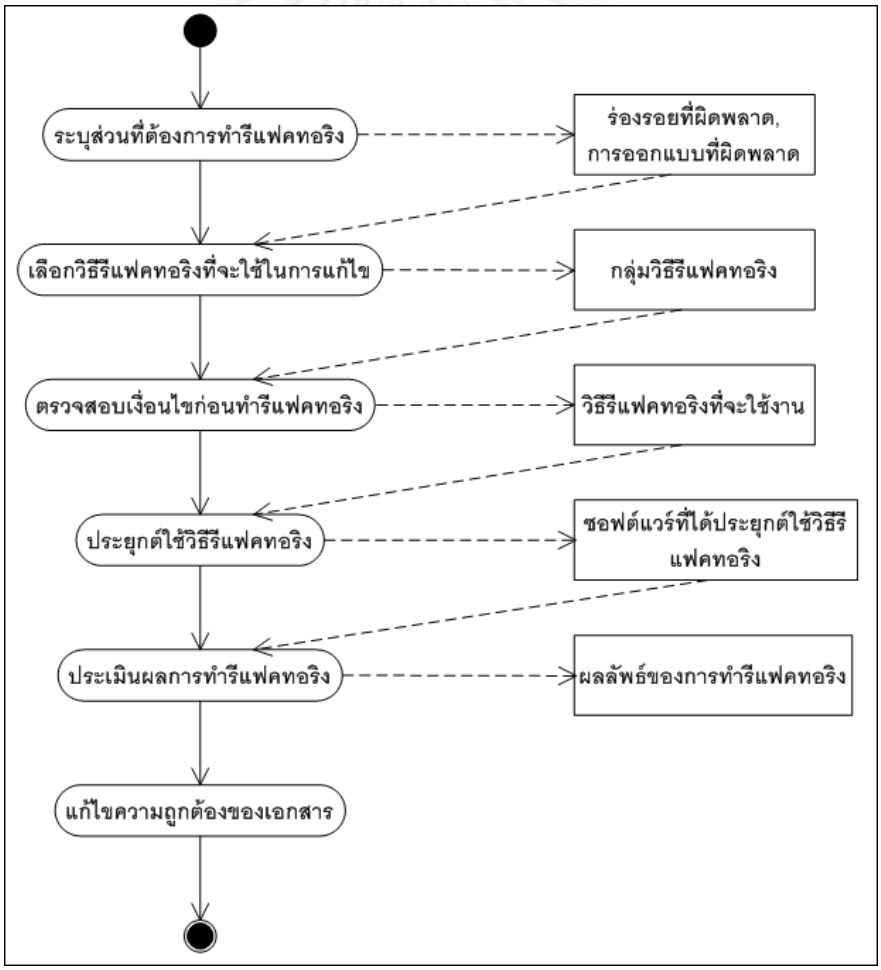
6. กลุ่ม Dealing with Generalization

เป็นวิธีรีแฟคทอริงสำหรับความสัมพันธ์เป็นลำดับชั้น ตัวอย่างวิธีรีแฟคทอริง ได้แก่ Pull Up Field, Pull Up Method, Push Down Method, Push Down Field, Extract Superclass, Collapse Hierarchy เป็นต้น

7. กลุ่ม Big Refactoring

เป็นวิธีรีแฟคทอริงสำหรับการเปลี่ยนแปลงหลายตำแหน่ง ดังนี้ Tease Apart Inheritance, Convert Procedure Design to Objects, Separate Domain from Presentation, Extract Hierarchy

กระบวนการทำรีแฟคทอริง [6] แสดงด้วยแผนภาพแอคทิวิตี้ ดังภาพที่ 1 โดยกระบวนการทำรีแฟคทอริงประกอบด้วย 6 ขั้นตอน มีรายละเอียด ดังนี้



ภาพที่ 1 แผนภาพแอคทิวิตีแสดงกระบวนการทำรีแฟคทอริง [6]

1. ระบุส่วนที่ต้องการทำรีแฟคทอริง

เป็นขั้นตอนในการค้นหาส่วนที่ต้องการทำรีแฟคทอริง ซึ่งพิจารณาจากร่องรอยที่ผิดพลาด (Bad Smell) หรือการออกแบบที่ผิดพลาด (Design Defect) ก็ได้

2. เลือกวิธีรีแฟคทอริงที่ใช้ในการแก้ไข

เป็นขั้นตอนในการเลือกวิธีรีแฟคทอริงที่นำมาใช้ในการปรับแก้ไขโค้ดการทำงานของซอฟต์แวร์

3. ตรวจสอบเงื่อนไขก่อนทำรีแฟคทอริง

เป็นขั้นตอนในตรวจสอบวิธีรีแฟคทอริงที่ใช้นั้นตรงตามเงื่อนไขในการใช้งานหรือไม่ เมื่อใช้งานแล้วซอฟต์แวร์นั้นยังคงต้องทำงานได้เหมือนเดิม ซึ่งเป็นการรับประกันก่อนการทำรีแฟคทอริง

4. ประยุกต์ใช้วิธีรีแฟคทอริง

เป็นขั้นตอนในการใช้งานวิธีรีแฟคทอริงที่ได้เลือกไว้จากขั้นตอนก่อนหน้านี้ ซึ่งการประยุกต์ใช้วิธีรีแฟคทอริงนั้นมี 2 แบบ คือ ผู้ใช้งานแก้ไขโค้ดเอง (Manual) และโปรแกรมแก้ไขให้ (Auto)

5. ประเมินผลการทำรีแฟคทอริง

เป็นขั้นตอนประเมินผลเปรียบเทียบค่าตัววัด เช่น ความมีประสิทธิภาพ (Performance) การบำรุงรักษา (Maintenance) ผลกระทบที่เกิดขึ้น (Impact) เป็นต้น ก่อนและภายหลังการทำรีแฟคทอริงเพื่อประเมินความคุ้มค่าในการทำ

6. แก้ไขความถูกต้องของเอกสาร

ภายหลังจากการทำรีแฟคทอริงเสร็จสิ้นแล้ว ขั้นตอนนี้เป็นขั้นตอนในการแก้ไขเอกสารที่เกี่ยวข้องหรือมีผลกระทบต่อการทำรีแฟคทอริง ให้ถูกต้องและสอดคล้องกับซอฟต์แวร์ภายหลังจากการทำรีแฟคทอริง

2.1.2 ร่องรอยที่ผิดพลาด (Bad Smell)

ร่องรอยที่ผิดพลาด [1] หมายถึง ลักษณะของซอร์ซโค้ดที่อาจก่อให้เกิดข้อผิดพลาดขึ้นได้ โดยเกิดจากความผิดพลาดในกระบวนการออกแบบซอฟต์แวร์ หรือการเขียนโค้ดที่ไม่เป็นระเบียบ ซึ่งลักษณะดังกล่าวนั้นจะทำให้ระบบทำงานไม่ถูกต้อง ยากต่อการทำความเข้าใจ และยากต่อการปรับแก้ไขภายหลัง ในการแก้ไขปัญหาเกี่ยวกับร่องรอยที่ผิดพลาดนั้นสามารถแก้ไขได้หลายวิธี เช่น การออกแบบโครงสร้างของซอฟต์แวร์ใหม่ การใช้วิธีรีแฟคทอริงในการแก้ไขปัญหา โดยมาร์ติน ฟาวเลอร์ได้นำเสนอลักษณะของร่องรอยที่ผิดพลาดทั้งหมด 22 แบบ ดังตารางที่ 1

ตารางที่ 1 แสดงรายละเอียดร่องรอยที่ผิดพลาด [1]

ร่องรอยที่ผิดพลาด	คุณลักษณะ
Duplicated Code	โปรแกรมที่มีโค้ดซ้ำซ้อนหลายตำแหน่ง
Long Method	เมท็อดที่มีความยาวมากจนเกินไป
Large Class	คลาสมีขนาดใหญ่ที่สามารถทำงานได้หลายอย่าง จนทำให้คลาสมีขนาดใหญ่มากจนเกินไป
Long parameter list	การรับพารามิเตอร์ที่มีขนาดยาวจนเกินไปทำให้ทำความเข้าใจได้ยาก
Divergent Change	โครงสร้างของคลาสถูกออกแบบมาให้สามารถทำการแก้ไขได้ โดยไม่กระทบกับคลาสที่เกี่ยวข้อง
Shotgun Surgery	ลักษณะของคลาสที่เมื่อมีการเปลี่ยนแปลงแล้ว จะต้องแก้ไขการทำงานของคลาสหลายคลาส จึงทำให้ยากต่อการตรวจสอบถึงในตำแหน่งที่ต้องแก้ไข
Feature Envy	ลักษณะของคลาสที่มีการเรียกใช้เมท็อดหรือตัวแปรของคลาสอื่นๆ มากเกินไป
Data Clumps	ลักษณะของข้อมูลตัวเดียวกันแต่มีกระจัดกระจายอยู่หลายคลาส
Primitive Obsession	ลักษณะการใช้ตัวแปรพื้นฐาน(Primitive Data) ให้ถูกวิธี โดยคำนึงถึงว่าใช้เพื่อสิ่งใดหรือควรสร้างเป็นคลาสแทนหรือไม่
Switch Statements	การใช้สวิชต์(Switch) จะทำให้เกิดการซ้ำกันของโปรแกรม ซึ่งจะสามารถหาจุดที่มีสวิชต์ได้ในหลายๆจุดในโปรแกรม ดังนั้นถ้ามีการเพิ่มลักษณะใหม่ลงมาในสวิชต์ก็จะทำให้ยากต่อการแก้ไข โดยจะใช้วิธีการสร้างเป็นคลาสเพื่อช่วงในการแก้ปัญหาดังกล่าว
Parallel Inheritance Hierarchical	เมื่อมีการสร้างคลาสลูกของคลาสหนึ่ง จะต้องตามเพิ่มคลาสลูกนั้นให้กับอีกคลาสหนึ่งด้วย
Lazy Class	มีลักษณะของคลาสที่ไม่ได้ใช้งานเป็นจำนวนมาก
Speculative Generality : Often	เมท็อดที่ถูกออกแบบมาให้ใช้งานอย่างใดอย่างหนึ่งแต่ไม่ได้ถูกเรียกใช้งานจริงเลย
Temporary field	การใช้ตัวแปรชั่วคราวมากเกินไป อาจทำให้เกิดความสับสนได้
Message Chains	การที่คลาสหนึ่งเรียกอ็อบเจ็คแล้วอ็อบเจ็คนั้นก็ไปเรียกอ็อบเจ็คถัดไปเรื่อยๆ เป็นลูกโซ่
Middle man	มีการเรียกใช้คลาสตัวแทน(Delegation)

Inappropriate Intimacy	การที่คลาสฯ หนึ่งสามารถเข้าถึงในส่วนที่ไม่สามารถเข้าถึงได้ของอีกคลาส
Alternative classes that with different interface	คลาสที่มีการทำงานเหมือนกันหลายๆ คลาสแต่มีชื่อต่างกัน
Incomplete library class	ไลบารี่ไม่สมบูรณ์ ทำให้ต้องมีการแก้ไขข้อมูลเพื่อให้การทำงานสมบูรณ์
Data Class	คลาสที่มีตัวแปรและเมท็อดของคลาส สำหรับทำงานที่สามารถเชื่อมต่อได้ แต่ยังไม่มีพฤติกรรมที่แท้จริง จึงต้องเปลี่ยนเมท็อดเหล่านั้นให้ระดับการเข้าถึงเป็นแบบส่วนตัว (private)
Refused Bequest	ลักษณะของคลาสลูกที่สืบทอดคุณสมบัติของคลาสแม่ แต่ไม่ได้ใช้งานคุณสมบัติที่สืบทอดดังกล่าว
Comments	ลักษณะของโปรแกรมที่มีการคอมเมนต์มากเกินไป

โดยร่องรอยที่ผิดพลาดสำหรับใช้ในการทดลองนั้นมี 3 ลักษณะ ได้แก่ ร่องรอยที่ผิดพลาดแบบเมท็อดที่มีความยาวมาก (Long Method) ร่องรอยที่ผิดพลาดแบบคลาสที่มีขนาดใหญ่ (Large Class) และร่องรอยที่ผิดพลาดแบบฟีเจอร์เอนวี (Feature Envy) โดยรายละเอียดของแต่ละร่องรอยที่ผิดพลาด มีดังนี้

1. ร่องรอยที่ผิดพลาดแบบเมท็อดที่มีความยาวมาก

ลักษณะของเมท็อดที่มีจำนวนบรรทัดของซอร์ซโค้ดการทำงานค่อนข้างยาว หรือมีจำนวนพารามิเตอร์และจำนวนตัวแปรชั่วคราว (Temporary variable) มาก ตัวอย่างซอร์ซโค้ดที่มีลักษณะร่องรอยที่ผิดพลาดแบบเมท็อดที่มีความยาวมากแสดงได้ดังภาพที่ 2 โดยวิธีรีแฟคทอริงที่ใช้ในการแก้ไขนั้นแสดงดังตารางที่ 2

ตารางที่ 2 แสดงวิธีรีแฟคทอริงและรายละเอียดสำหรับแก้ไขร่องรอยที่ผิดพลาดแบบเมท็อดที่มีความยาวมาก [1]

วิธีรีแฟคทอริง	รายละเอียด
Extract Method	แยกโค้ดการทำงานออกไปเป็นเมท็อดใหม่ที่มีการทำงานเฉพาะทาง
Replace Temp with Query	แทนที่การใช้งานตัวแปรชั่วคราวด้วยการเรียกใช้งานเมท็อดในการทำงานแทน
Introduce Parameter Object	แทนที่การใช้งานหลายตัวแปรในการส่งเป็นค่าพารามิเตอร์ด้วยการรวมตัวแปรเหล่านั้นเป็นตัวแปรอ็อบเจ็คแทน
Preserve Whole Object	ส่งตัวแปรอ็อบเจ็คแทนการส่งด้วยตัวแปรพื้นฐาน(Primitive Type)

Decompose conditional	แตกเงื่อนไขของ if else ที่ซับซ้อน แยกออกเป็นเงื่อนไขที่ลดความซับซ้อนลง อ่านแล้วเข้าใจง่าย

```
public String statement(){
    double totalAmount = 0;
    int frequentRenterPoints = 0;
    Enumeration rentals = _rentals.elements();

    String result = "Rental Record for " + getName() + "\n";

    while (rentals.hasMoreElements()) {
        double thisAmount = 0;
        Rental each = (Rental) rentals.nextElement();
        //determine amounts for each line
        switch (each.getMovie().getPriceCode()) {
        case Movie.REGULAR:
            thisAmount += 2;
            if (each.getDaysRented() > 2)
                thisAmount += (each.getDaysRented() - 2) * 1.5;
            break;
        case Movie.NEW_RELEASE:
            thisAmount += each.getDaysRented() * 3;
            break;
        case Movie.CHILDRENS:
            thisAmount += 1.5;
            if (each.getDaysRented() > 3)
                thisAmount += (each.getDaysRented() - 3) * 1.5;
            break;
        }
        // add frequent renter points
        frequentRenterPoints ++;
        // add bonus for a two day new release rental
        if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE)
                &&
                each.getDaysRented() > 1) frequentRenterPoints ++;
        //show figures for this rental
        result += "\t" + each.getMovie().getTitle()+ "\t" +
                String.valueOf(thisAmount) + "\n";
        totalAmount += thisAmount;
    }
    //add footer lines
    result += "Amount owed is " + String.valueOf(totalAmount) +
            "\n";
    result += "You earned " + String.valueOf(frequentRenterPoints)
            +
            " frequent renter points";
    return result;

}
```

ภาพที่ 2 ซอร์ซโค้ดที่มีร่องรอยที่ผิดพลาดแบบเมท็อดที่มีความยาวมาก

จากซอร์ซโค้ดตัวอย่างดังภาพที่ 2 เมท็อด Statement ทำหน้าที่ในการคำนวณค่าเช่าภาพยนตร์ (totalAmount) และค่าความถี่สะสมของการเช่าภาพยนตร์ของลูกค้า (frequentRenterPointers) เมท็อดนี้มีความยาวของจำนวนบรรทัดค่อนข้างมากและมีการใช้งานตัวแปรชั่วคราวหลายตำแหน่งภายในซอร์ซโค้ด (thisAmount และ frequentRenterPointers) เมื่อมีการปรับเปลี่ยนซอร์ซโค้ดการทำงานที่เมท็อดดังกล่าวจะทำให้ยากต่อการทำความเข้าใจและปรับแก้ไข

2. ร่องรอยที่ผิดพลาดแบบคลาสที่มีขนาดใหญ่

ลักษณะของคลาสที่มีจำนวนเมท็อดและตัวแปรอินสแตนซ์ (Instance variable) ของคลาสเป็นจำนวนมาก ซึ่งเมท็อดและตัวแปรอินสแตนซ์นั้นอาจจะไม่มีความเกี่ยวข้องกับคลาสดังกล่าวโดยตรง มีเพียงไว้เพื่อให้คลาสอื่นเรียกใช้งานเท่านั้น หรือเป็นคลาสที่มีฟังก์ชันการทำงานที่มากเกินไป ตัวอย่างซอร์ซโค้ดที่มีลักษณะที่ผิดพลาดแบบคลาสที่มีขนาดใหญ่แสดงได้ดังภาพที่ 3 และวิธีรีแฟคทอริงที่ใช้ในการแก้ไขนั้นแสดงได้ดังตารางที่ 3 ได้แก่

ตารางที่ 3 แสดงวิธีรีแฟคทอริงและรายละเอียดสำหรับแก้ไขร่องรอยที่ผิดพลาดแบบคลาสที่มีขนาดใหญ่ [1]

วิธีรีแฟคทอริง	รายละเอียด
Extract Class	แยกเมท็อดหรือตัวแปรของคลาส ออกไปอยู่ในคลาสใหม่ที่แยกออกมา
Extract Subclass	แยกเมท็อดหรือตัวแปรของคลาส ออกไปอยู่ในคลาสใหม่ที่เป็นคลาสลูก (subclass)
Pull up Field	ย้ายตัวแปรของคลาสลูกไปไว้ที่คลาสแม่ (super class) แทน
Pull up Method	ย้ายเมท็อดของคลาสลูกไปไว้ที่คลาสแม่แทน
Push Down Method	ย้ายเมท็อดของคลาสแม่ไปไว้ที่คลาสลูกแทน
Push Down Field	ย้ายตัวแปรของคลาสแม่ไปไว้ที่คลาสลูกแทน

```
public class TimeSeries extends Series implements Cloneable, Serializable {

    private static final long serialVersionUID = -5032960206869675528L;
    protected static final String DEFAULT_DOMAIN_DESCRIPTION = "Time";
    protected static final String DEFAULT_RANGE_DESCRIPTION = "Value";
    private String domain;
    private String range;
    protected Class timePeriodClass;
    protected List data;
    private int maximumItemCount;
    private long maximumItemAge;
    private double minY;
    private double maxY;

    public TimeSeries(Comparable name) {
    public TimeSeries(Comparable name, String domain, String range) {
    private double maxIgnoreNaN(double a, double b) {
    public TimeSeries(Comparable name, Class timePeriodClass) {
    public TimeSeries(Comparable name, String domain, String range,
    public TimeSeries createCopy(RegularTimePeriod start, RegularTimePeriod end)
    public void add(TimeSeriesDataItem item, boolean notify) {
    private double minIgnoreNaN(double a, double b) {
    public String getDomainDescription() {
    public void setDomainDescription(String description) {
    public String getRangeDescription() {
    public void setRangeDescription(String description) {
    public int getItemCount() {
    public int getIndex(RegularTimePeriod period) {
    public List getItems() {
    public TimeSeries addAndOrUpdate(TimeSeries series) {
    public void setMaximumItemCount(int maximum) {
    public int getMaximumItemCount() {
    public Collection getTimePeriods() {
    public Collection getTimePeriodsUniqueToOtherSeries(TimeSeries series) {
    public void update(int index, Number value) {
    public void removeAgedItems(long latest, boolean notify) {
    public void clear() {
    public long getMaximumItemAge() {
    public void setMaximumItemAge(long periods) {
```

ภาพที่ 3 ซอร์ซโค้ดที่มีร่องรอยที่ผิดพลาดแบบคลาสที่มีขนาดใหญ่

จากซอร์ซโค้ดตัวอย่างดังภาพที่ 3 คลาส TimeSeries เป็นคลาสที่ทำหน้าที่คำนวณเกี่ยวกับเวลาต่างๆ ที่ใช้งานภายในโปรแกรมโอเพนซอร์ซ JFreeChart มีจำนวนเมท็อดสำหรับเรียกใช้งานเป็นจำนวนมาก คลาสนี้จึงมีร่องรอยที่ผิดพลาดแบบคลาสที่มีขนาดใหญ่

3. ร่องรอยที่ผิดพลาดแบบฟีเจอร์เอนวี

ลักษณะของคลาสที่มีเมท็อดหรือตัวแปรของคลาสถูกเรียกใช้งานโดยคลาสอื่นมากกว่าคลาสที่เป็นเจ้าของเมท็อดหรือตัวแปรนั้น ตัวอย่างซอร์ซโค้ดที่มีร่องรอยที่ผิดพลาดแบบฟีเจอร์เอนวี แสดงได้ดังภาพที่ 4 โดยวิธีรีแฟคทอริงที่ใช้ในการแก้ไขนั้นแสดงได้ดังตารางที่ 4

ตารางที่ 4 แสดงวิธีรีแฟคทอริงและรายละเอียดสำหรับแก้ไขร่องรอยที่ผิดพลาดแบบฟีเจอร์เอนวี [1]

วิธีรีแฟคทอริง	รายละเอียด
Move Method	ย้ายเมท็อดจากคลาสหนึ่ง ไปไว้ที่คลาสอื่นแทน
Move Field	ย้ายตัวแปรจากคลาสหนึ่ง ไปไว้ที่คลาสอื่นแทน
Extract Method	แยกโค้ดการทำงานออกไปเป็นเมท็อดใหม่ที่มีการทำงานเฉพาะทาง

```
public class Phone {

    private String areaCode;
    private String prefix;
    private String number;

    public String getAreaCode() {
        return areaCode;
    }
    public void setAreaCode(String areaCode) {
        this.areaCode = areaCode;
    }
    public String getPrefix() {
        return prefix;
    }
    public void setPrefix(String prefix) {
        this.prefix = prefix;
    }
    public String getNumber() {
        return number;
    }
    public void setNumber(String number) {
        this.number = number;
    }

public class Customer {

    private Phone mobilePhone;

    public String getMobilePhoneNumber(){
        return "(" + mobilePhone.getAreaCode() + ")"
            + mobilePhone.getPrefix() + "-" + mobilePhone.getNumber();

    }
```

ภาพที่ 4 ซอร์ซโค้ดที่มีร่องรอยที่ผิดพลาดแบบฟีเจอร์เอนวี

จากซอร์ซโค้ดตัวอย่างดังตารางที่ 4 ประกอบด้วย 2 คลาส คือ คลาส Phone ทำหน้าที่เก็บรายละเอียดของเบอร์โทรศัพท์ และคลาส Customer ทำหน้าที่แสดง

รายละเอียดของเบอร์โทรศัพท์ โดยการเรียกใช้งานเมท็อด getMobilePhoneNumber ในการแสดงรายละเอียดของเบอร์โทรศัพท์ทั้ง 3 ส่วน คือ areaCode, prefix และ number เป็นลักษณะการเขียนโค้ดที่นำเอาคลาส Customer มาทำหน้าที่แสดงข้อมูลหมายเลขโทรศัพท์แทนคลาส Phone โดยการนำผลลัพธ์ของการเรียกใช้งาน 3 เมท็อดของคลาส Phone คือ getAreaCode, getPrefix และ getNumber มารวมกันเป็นรายละเอียดของหมายเลขโทรศัพท์ ซึ่งการเขียนเมท็อดในลักษณะดังกล่าวเป็นลักษณะการเขียนโค้ดที่มีร่องรอยที่ผิดพลาดแบบฟีเจอร์เอนวี โดยคลาส Customer แย่งหน้าที่ความรับผิดชอบของคลาส Phone ไป แนวทางการแก้ไขคือควรย้ายเมท็อดที่ใช้ในการประกอบข้อมูลเบอร์โทรศัพท์ไปไว้ที่คลาส Phone แทน แล้วให้คลาส Customer ทำหน้าที่แค่เรียกใช้งานแทน

2.1.3 อัลกอริทึมละโมบ (Greedy Algorithm)

อัลกอริทึมละโมบเป็นอัลกอริทึม (Algorithm) [7] สำหรับแก้ไขปัญหาการค้นหาแบบมีข้อมูล (Heuristic Search) ที่จะเลือกเส้นทางที่ดีที่สุดก่อน เพื่อค้นหาเป้าหมายอย่างรวดเร็ว โดยพิจารณาเส้นทางที่สามารถมองเห็นได้ทุกตำแหน่งในขณะนั้น และเลือกเส้นทางที่ใช้ทรัพยากรน้อยที่สุด ทำเช่นนี้ไปเรื่อยๆ จนถึงเป้าหมายที่ต้องการ การทำงานของอัลกอริทึมละโมบมีรายละเอียดดังนี้

1. สร้างโหนดเริ่มต้น

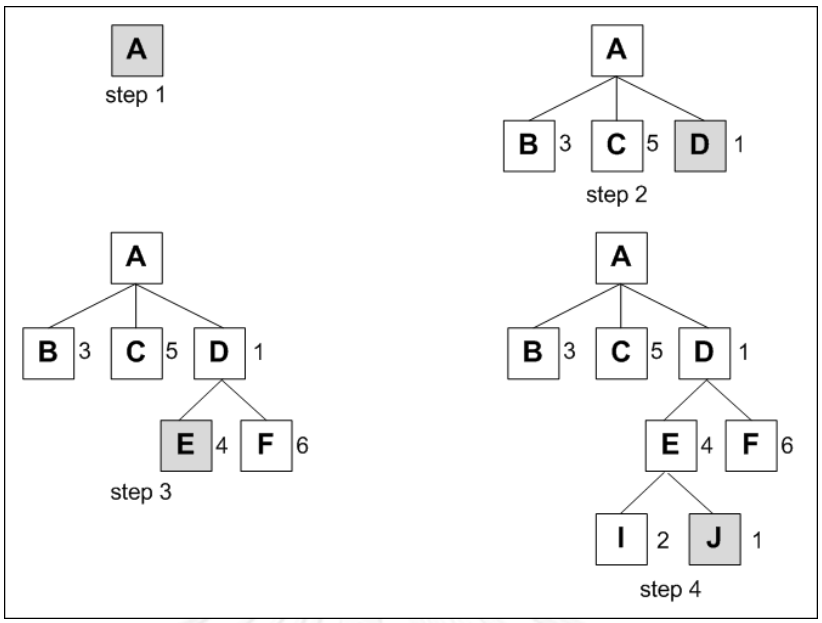
2. วนรอบการทำงาน โดยตรวจสอบว่าถึงโหนดเป้าหมาย (goal state) หรือไม่พบโหนดต่อไปอีก จะหยุดการสร้างเส้นทางทันที แต่ถ้ายังไม่ถึงโหนดเป้าหมายให้ทำตามเงื่อนไขดังนี้

    2.1 สร้างโหนดลูกทั้งหมดที่เป็นไปได้จากโหนดแม่

    2.2 เลือกโหนดลูกที่ให้เส้นทางที่ดีที่สุด

    2.3 กำหนดให้โหนดลูกที่ได้รับเลือกให้เป็นโหนดแม่สำหรับการเลือกในรอบถัดไป

จากภาพที่ 5 แสดงตัวอย่างขั้นตอนการค้นหาเส้นทางโดยการใช้อัลกอริทึมละโมบในการค้นหาเส้นทางที่สั้นที่สุด โดยเริ่มจากการสร้างโหนด A ที่เป็นโหนดเริ่มต้น จากนั้นจึงสร้างโหนดลูก B C และ D ที่มีระยะทางแต่ละโหนดเรียงจากซ้ายไปขวาเป็น 3 5 และ 1 หน่วยตามลำดับ ในการเดินทางไปแต่ละโหนดนั้นอัลกอริทึมละโมบจะพิจารณาเลือกโหนดที่มีระยะทางน้อยที่สุดเป็นเส้นทางที่จะเดินในแต่ละชั้น โดยในชั้นแรกโหนด D ที่มีระยะทางน้อยที่สุดในชั้นที่ 2 จะถูกเลือก โหนด D จึงเป็นโหนดที่จะสร้างทางเดินสำหรับชั้นที่ 3 ต่อไป ทางเดินที่สามารถเดินต่อไปจากโหนด D ได้มีด้วยกัน 2 ทางคือ โหนด E และ โหนด F ซึ่งมีระยะทางแต่ละโหนดเป็น 4 และ 6 หน่วยต่อลำดับ โหนด E มีระยะทางน้อยกว่าโหนด F จึงถูกเลือกเป็นเส้นทางที่จะเดินต่อไปสำหรับชั้นที่ 3 จากโหนด E จะมีเส้นทางให้เลือกเดินอยู่ 2 เส้นทาง คือ โหนด I และ J ซึ่งโหนด J มีระยะเดินเป็น 1 หน่วยซึ่งน้อยกว่าโหนด I ที่มีระยะทาง 2 หน่วย จึงเลือกเดินเส้นทางโหนด J ซึ่งโหนด J ไม่มีเส้นทางเดินต่อไปแล้ว จึงหยุดการค้นหาเส้นทาง ดังนั้นเส้นทางการเดินทางจากโหนดเริ่มต้นไปยังโหนดลูกชั้นที่ 3 คือ โหนด A ไปยัง โหนด D ไปยัง โหนด E และ โหนด J ตามระยะทางที่สั้นที่สุดในแต่ละระดับชั้น



ภาพที่ 5 ตัวอย่างขั้นตอนการค้นหาเส้นทางโดยการใช้อัลกอริทึมละโมบ [7]

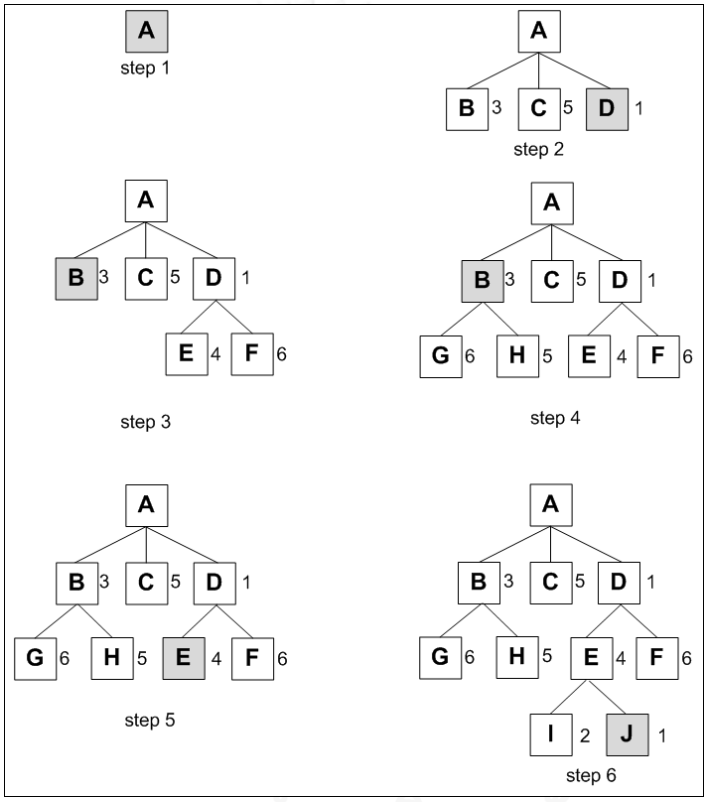
2.1.4 อัลกอริทึมค้นหาแนวกว้าง (Breadth First Search)

เป็นอัลกอริทึมที่จะเลือกเส้นทางที่ดีที่สุด โดยจะนำเอาโหนดเพื่อนบ้านหรือโหนดใกล้เคียงของการพิจารณารอบก่อนหน้านี้มาเป็นเกณฑ์ในการตัดสินใจด้วย อัลกอริทึมนี้มีคุณสมบัติในการเดินย้อนกลับเส้นทางเดิมได้ถ้าเส้นทางใหม่ที่จะเดินมีผลลัพธ์ที่แย่กว่าเส้นเดิมก่อนหน้านี้ การทำงานของอัลกอริทึมค้นหาแนวกว้างมีรายละเอียดดังนี้

1. สร้างโหนดเริ่มต้น
2. สำรวจโหนดลูกถัดไป โดยพิจารณาเงื่อนไข
    2.1 ถ้าพบโหนดที่ต้องการ ให้หยุดการค้นหาและคืนค่าโหนดดังกล่าว
    2.2 ถ้าไม่พบโหนดที่ต้องการ ให้พิจารณาโหนดลูกของโหนดเพื่อนบ้านหรือโหนดใกล้เคียง
3. ถ้าสำรวจจนครบทุกโหนดแล้ว ยังไม่พบโหนดที่ต้องการ ให้หยุดการค้นหา
4. ถ้ายังสำรวจไม่ครบ ให้ย้อนกลับไปทำข้อ 2 ใหม่อีกครั้ง

จากภาพที่ 6 แสดงตัวอย่างขั้นตอนการค้นหาเส้นทางโดยการใช้อัลกอริทึมค้นหาแนวกว้างในการค้นหาเส้นทางที่สั้นที่สุด โดยเริ่มจากการสร้างโหนด A ที่เป็นโหนดเริ่มต้น จากนั้นจึงสร้างโหนดลูก B C และ D ที่มีระยะทางแต่ละโหนดเรียงจากซ้ายไปขวาเป็น 3 5 และ 1 หน่วยตามลำดับในการเดินทางไปแต่ละโหนดนั้นอัลกอริทึมค้นหาแนวกว้างจะพิจารณาเลือกโหนดที่มีระยะทางน้อยที่สุดเป็นเส้นทางที่จะเดินในแต่ละชั้น โดยในชั้นแรกโหนด D ที่มีระยะทางน้อยที่สุดในชั้นที่ 2 จะถูกเลือก โหนด D จึงเป็นโหนดที่จะสร้างทางเดินสำหรับชั้นที่ 3 ต่อไป โดยในส่วนของการพิจารณาโหนด

ในชั้นที่ 3 นั้นจะนำเอาโหนดเพื่อนบ้านที่ไม่ได้ถูกเลือกในโหนดชั้นที่ 2 คือ โหนด B และ C มาพิจารณาร่วมกับโหนดลูกของ D ในชั้นที่ 3 คือ โหนด E และ F ซึ่งโหนด B มีระยะทางสั้นที่สุด คือ 3 หน่วยจึงถูกเลือกในการพิจารณารอบที่ 3 วนรอบการขั้นหาไปจนกระทั้งสิ้นสุดเส้นทางของโหนดชั้นที่ 3 ที่เป็นคำตอบจึงหยุดการค้นหา ดังนั้นเส้นทางการเดินทางจากโหนดเริ่มต้นไปยังโหนดลูกชั้นที่ 3 คือ โหนด A ไปยัง โหนด D ไปยัง โหนด B ไปยังโหนด E และ โหนด J ตามระยะทางที่สั้นที่สุดในแต่ละระดับชั้น



ภาพที่ 6 ตัวอย่างขั้นตอนการค้นหาเส้นทางโดยการใช้อัลกอริทึมค้นหาแนวกว้าง [7]

2.1.5 อัลกอริทึมปีนเขา (Hill Climbing)

เป็นอัลกอริทึมที่จะเลือกเส้นทางที่ดีที่สุดไปเรื่อยๆ ในทุกรอบของการค้นหา โดยขั้นการค้นหานั้นจะพิจารณาเลือกเดินไปยังเส้นทางที่ให้ผลลัพธ์ที่ดีที่สุดและต้องดีกว่าเส้นทางปัจจุบันที่ได้เคยเลือกเดินไปแล้ว แต่ถ้าไม่พบเส้นทางที่ดีกว่าเส้นทางเดิมก็จะหยุดค้นหาคำตอบทันที การทำงานของอัลกอริทึมปีนเขามีรายละเอียดดังนี้

1. สร้างโหนดเริ่มต้น

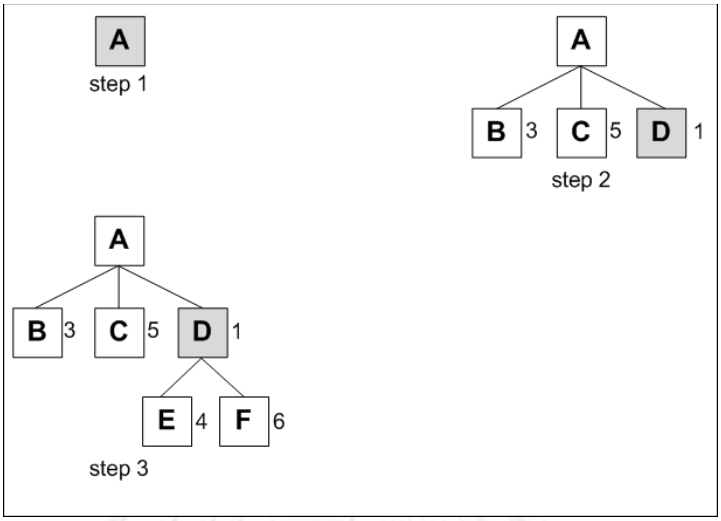
2. วนรอบการทำงาน โดยตรวจสอบว่าถึงโหนดเป้าหมายหรือไม่พบโหนดต่อไปอีก จะหยุดการสร้างทางทันที แต่ถ้าไม่ถึงโหนดเป้าหมายให้ทำตามเงื่อนไขดังนี้

2.1 สร้างโหนดลูกทั้งหมดที่เป็นไปได้จากโหนดเริ่มต้น

2.2 เลือกโหนดลูกที่ให้เส้นทางที่ดีที่สุดและต้องดีกว่าโหนดปัจจุบัน

2.3 ถ้าไม่พบโหนดที่ดีกว่า จะหยุดการค้นหาทันที

3. ถ้าสำรวจจนครบทุกโหนดแล้ว ยังไม่พบโหนดที่ต้องการ ให้หยุดการค้นหา



ภาพที่ 7 ตัวอย่างขั้นตอนการค้นหาเส้นทางโดยการใช้อัลกอริทึมปีนเขา [7]

จากภาพที่ 7 แสดงตัวอย่างขั้นตอนการค้นหาเส้นทางโดยการใช้อัลกอริทึมปีนเขาในการค้นหาเส้นทางที่สั้นที่สุด โดยเริ่มจากการสร้างโหนด A ที่เป็นโหนดเริ่มต้น จากนั้นจึงสร้างโหนดลูก B C และ D ที่มีระยะทางแต่ละโหนดเรียงจากซ้ายไปขวาเป็น 3 5 และ 1 หน่วยตามลำดับ ในการเดินทางไปแต่ละโหนดนั้นอัลกอริทึมค้นหาแนวกว้างจะพิจารณาเลือกโหนดที่มีระยะทางน้อยที่สุดเป็นเส้นทางที่จะเดินในแต่ละชั้น โดยในชั้นแรกโหนด D ที่มีระยะทางน้อยที่สุดในชั้นที่ 2 จะถูกเลือกโหนด D จึงเป็นโหนดที่จะสร้างทางเดินสำหรับชั้นที่ 3 ต่อไป จากโหนด D มีเส้นทางให้สร้าง 2 โหนด คือโหนด E และ โหนด F ที่มีระยะทางเป็น 4 หน่วยและ 6 หน่วยตามลำดับ ซึ่งโหนดทั้ง 2 มีเส้นทางที่ยาวกว่าโหนด D จึงหยุดการค้นหา ดังนั้นเส้นทางการเดินทางจากโหนดเริ่มต้นไปยังโหนดลูก คือ โหนด A ไปและ โหนด D ตามลำดับ

2.1.6 อัลกอริทึมเอสตาร์ (A* - A Star)

เป็นอัลกอริทึมที่ใช้ค้นหาเส้นทางที่ดีที่สุดและใช้ทรัพยากรน้อยที่สุดจากโหนดเริ่มต้นไปยังโหนดที่เป็นเป้าหมาย โดยการใช้ฟังก์ชันฮิวริสติกแบบค่าของระยะทาง (f(x)) ซึ่งเกิดจากผลรวมของสองฟังก์ชัน คือ ค่าระยะทางจากโหนดเริ่มต้นไปยังโหนดปัจจุบัน (g(x)) และค่าระยะทางจากโหนดปัจจุบันไปยังโหนดที่เป็นเป้าหมาย (h(x)) ซึ่งรายละเอียดฟังก์ชันฮิวรีสติกมีดังนี้

$$f(x) = g(x) + h(x)$$

- f(x) เป็นฟังก์ชันฮิวริสติกที่ระบุถึงทรัพยากรที่ใช้ไปยังโหนดเป้าหมาย
- g(x) เป็นค่าทรัพยากรที่ใช้ในการไปยังโหนดปัจจุบัน
- h(x) เป็นค่าทรัพยากรที่ใช้โดยนับจากโหนดปัจจุบันไปยังโหนดเป้าหมาย

2.1.7 การวัดซอฟต์แวร์เชิงวัตถุ (Object Oriented Software Measurement)

การวัด (Measurement) คือ การกำหนดค่าตัวเลขหรือตัวอักษรให้กับคุณลักษณะของสิ่งที่สนใจ (Entity) เพื่ออธิบายคุณลักษณะ (Attribute) นั้นๆ การวัดจำเป็นต้องมีมาตรวัดเพื่อใช้เป็นตัวระบุลักษณะที่จะทำการวัด มาตรวัดสามารถแบ่งได้เป็น 2 ประเภท คือ

1. มาตรวัดซอฟต์แวร์แบบดั้งเดิม (Traditional Metrics)

เช่น มาตรวัดไซโคลเมติกของแมคเคบ (Cyclomatic complexity – V(G)) มาตรวัดจำนวนบรรทัด (Line of code - LOC) เป็นต้น

2. มาตรวัดซอฟต์แวร์เชิงวัตถุ (Object Oriented Metrics) ของชิมดาเบิ้ลและเคมเมอร์เรอร์ (Chidamber and Kemerer) [8]

เช่น มาตรวัดระดับความลึกของการสืบทอดคุณสมบัติของคลาส (Depth of Inheritance of Tree in Class) มาตรวัดการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส (Lack of Cohesion in Methods - LCOM) มาตรวัดการเข้าคู่กันระหว่างวัตถุ (Coupling Between Objects - CBO) เป็นต้น

2.1 การเข้าคู่กันระหว่างวัตถุ [8]

คำนิยาม – การเข้าคู่กันระหว่างวัตถุของคลาสหนึ่งๆ จะนับจากจำนวนของคลาสอื่นที่มีความสัมพันธ์กัน

ทฤษฎีพื้นฐาน – การเข้าคู่กันระหว่างวัตถุเป็นความสัมพันธ์ระหว่างอ็อบเจ็คของคลาสหนึ่งที่มีความสัมพันธ์กับอ็อบเจ็คของอีกคลาสหนึ่ง ซึ่งอาจจะมีความสัมพันธ์มากกว่า 1 อ็อบเจ็คก็ได้ โดยความสัมพันธ์ระหว่างอ็อบเจ็คนั้นมีด้วยกัน 2 ลักษณะ คือ ลักษณะที่อ็อบเจ็คเรียกใช้งานระหว่างกัน และลักษณะที่อ็อบเจ็คเรียกใช้งานตัวแปรอินสแตนซ์ของอีกอ็อบเจ็คหนึ่ง

มุมมอง

- การเข้าคู่กันระหว่างวัตถุมีผลต่อการออกแบบและการนำกลับมาใช้ใหม่ โดยคลาสที่มีความอิสระต่อคลาสอื่นๆ จะง่ายต่อการนำกลับมาใช้งานใหม่ ตรงกันข้ามถ้าคลาสนั้นมีความเกี่ยวข้องกับคลาสอื่นๆ หลายคลาส การแก้ไขหรือการนำกลับมาใช้งานใหม่ก็จะทำได้ยาก

- ค่าที่ได้ภายหลังจากใช้มาตรวัดการเข้าคู่กันระหว่างวัตถุ ควรมีค่าน้อย ยิ่งค่าน้อย คลาสนั้นก็จะมีความเป็นอิสระต่อคลาสอื่นมาก การแก้ไขหรือการนำกลับมาใช้งานใหม่ก็จะง่ายขึ้น

การคำนวณ

- ใช้การนับจำนวนสิ่งที่มีความเกี่ยวข้องกับคลาสหรืออ็อบเจ็คอื่น ได้แก่ จำนวนการเรียกใช้งานเมท็อดอื่น (Method Invocation) ตัวแปรอ้างอิง (Variable Reference) ชนิดตัวแปรในการคืนค่า (Return Types) ชนิดของพารามิเตอร์ (Formal Parameter)

การเข้าถึงค่าตัวแปร (Field Accesses) การสืบทอดคุณสมบัติ (Inheritance) โพลีมอพิซึม (Polymorphism) กรณียกเว้น (Exception)

- ค่าที่ได้ภายหลังจากใช้มาตรวัดการเข้าคู่กันระหว่างวัตถุ ควรมีค่าน้อย ยิ่งค่าน้อย คลาสนั้นก็จะมีความเป็นอิสระต่อคลาสอื่นมาก การแก้ไขหรือการนำกลับมาใช้งานใหม่ก็จะง่ายขึ้น

2.2 ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส [8]

คำนิยาม – ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส นับจากจำนวนของตัวแปรอินสแตนซ์ที่ไม่ได้ใช้งานร่วมกัน (disjoin sets) ที่ถูกเรียกใช้งานในเมท็อดของคลาสเดียวกัน

กำหนดให้คลาส C1 มีจำนวนเมท็อดเท่ากับ n และ M1, M2, ..., Mn แทนเมท็อดที่มีอยู่ในคลาส C1 และ { Ij } เป็นเซตของตัวแปรอินสแตนซ์ที่ถูกเรียกใช้งานในเมท็อด Mi ถ้าในคลาส C1 มีจำนวนเมท็อดเท่ากับ n จะมีเซตของตัวแปรอินสแตนซ์ทั้งหมดเป็น { I1 }, ..., { In } และกำหนดให้ P และ Q เป็นเซตของ I ที่มีรายละเอียดดังนี้ P = {(Ii ,Ij) | Ii$\cap$Ij = Ø } และQ = {(Ii ,Ij) | Ii$\cap$Ij ≠ Ø } ถ้ามีจำนวนเซตทั้งหมดเป็น n โดยที่มี { I1 }, ..., { In } เป็น Ø ด้วย ดังนั้นสูตรการคำนวณหาระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส คือ

LCOM = |P| - |Q|, ถ้า |P| > |Q| หรือ

= จำนวนเซตของตัวแปรที่ไม่ได้ใช้ร่วมกัน – จำนวนเซตของตัวแปรที่ใช้ร่วมกัน

ทฤษฎีพื้นฐาน – ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสจะนับจากจำนวนเมท็อดที่ไม่ได้ใช้ตัวแปรอินสแตนซ์ร่วมกัน (เซตของตัวแปรที่เรียกใช้งานแต่ละเมท็อดอินเตอร์เซกกันแล้วได้เซตว่าง)

มุมมอง

- ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสถูกนำมาใช้ในการพิจารณาเรื่องคุณสมบัติด้านการปกปิดข้อมูล (Encapsulation)

- ถ้าระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดมีค่ามากจะทำให้ความซับซ้อนของระบบนั้นมีค่ามากขึ้นตาม

- ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสบ่งบอกว่าคลาสนั้นควรแยกออกเป็นคลาสย่อย (subclasses) หรือไม่

- ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสควรมีค่าน้อย จะทำให้คลาสนั้นง่ายต่อการทดสอบ

การคำนวณ

- ตัวอย่างการคำนวณแสดงได้ดังนี้ ให้คลาส C มี 3 เมท็อด คือ M1, M2 , M3 กำหนดให้ I เป็นเซตของตัวแปรอินสแตนซ์ที่ใช้เมท็อด โดยที่ I1 เป็นเซตของตัวแปรอินสแตนซ์ที่ใช้ในเมท็อด M1 มีสมาชิกเป็น {a,b,c,d,e} และ I2 เป็นเซตของตัวแปรอินสแตนซ์ที่ใช้ในเมท็อด M2 มีสมาชิกเป็น {a,b,e} และ I3 เป็นเซตของตัวแปรอินสแตนซ์ที่ใช้ในเมท็อด M3 มี

สมาชิกเป็น {x,y,z} เมื่อนำ { I1} ⋂{ I2} ได้ค่าเป็นเซต {a,b,e} และ { I1} ⋂{ I3} กับ { I2} ⋂{ I3} ได้ค่าเป็นเซตว่าง จะได้ค่าจำนวนเซตของตัวแปรที่ไม่ได้ใช้งานร่วมกันเท่ากับ 2 หน่วย และค่าจำนวนเซตของตัวแปรที่ใช้งานร่วมกันเท่ากับ 1 หน่วย ดังนั้นระดับของการขาดกันเกาะกันเป็นก้อนของเมท็อดภายในคลาสมีค่าเท่ากับ 1 หน่วย

2.3 มาตรวัดผลรวมค่าความซับซ้อนต่อคลาส [8]

คำนิยาม – มาตรวัดผลรวมค่าความซับซ้อนต่อคลาสของคลาสหนึ่งๆ จะนับจากผลรวมของค่าความซับซ้อนของแต่ละเมท็อดต่อคลาสคือ

$$WMC = \sum_{i=1}^{n} c_1$$

โดยค่าความซับซ้อนภายในเมท็อดนั้นสามารถหาได้จากค่าความซับซ้อนของแมคเคบ (McCabe ’s Cyclomatic Complexity) ซึ่งมีสูตรการคำนวณดังต่อไปนี้

M = E – N + 2P

E = จำนวนด้าน

N = จำนวนโหนด

และ P = 1 สำหรับโปรแกรมที่มีการทำงานย่อย

หรือสามารถคำนวณได้จาก

M = V(G) =จำนวนเงื่อนไขการตัดสินใจ (if , else if, switch, for/while) + 1

ทฤษฎีพื้นฐาน – มาตรวัดผลรวมค่าความซับซ้อนต่อคลาสจะมีความสัมพันธ์โดยตรงกับความซับซ้อนของเมท็อด

มุมมอง

- จำนวนเมท็อดและความซับซ้อนของเมท็อด ใช้ในการประมาณเวลาและแรงงานในการพัฒนาและค่าความสามารถในการบำรุงรักษาของคลาสได้
- ถ้ามีค่ามาตรวัดผลรวมค่าความซับซ้อนต่อคลาสมาก คลาสนั้นจะมีความซับซ้อนมาก

2.2 งานวิจัยที่เกี่ยวข้อง

2.2.1 Bad-Smell Detection For Refactoring Using Object-Oriented Software Metrics [4]

งานวิจัยนี้นำเสนอวิธีการตรวจสอบร่องรอยที่ผิดพลาดสำหรับซอร์ซโค้ด 6 ลักษณะ คือ Feature Envy, Large Class, Lazy Class, Long Method, Long Parameter Lists และ Switch Statement โดยใช้มาตรวัดซอฟต์แวร์เชิงวัตถุ ได้แก่ Number of instance method in a class (NIM), Number of instance variable in a class (NIV) เป็นต้น และได้ออกแบบมาตรวัดใหม่เพื่อ

ใช้ในการตรวจสอบร่องรอยที่ผิดพลาดทั้ง 6 ลักษณะเพิ่มเติม คือ Number of called method (NCDM), Number of called attributes (NCDA), Number of caller attributes (NCRA), Set of method used by instance variable in a class (SMIV), Number of temporary variables in a method (NOT) และ Number of switch statement in a method (NOSS) เพื่อช่วยในการตรวจสอบหาร่องรอยที่ผิดพลาดที่มีอยู่ในซอร์ซโค้ดที่เป็นข้อมูลนำเข้า พร้อมทั้งระบุตำแหน่งของร่องรอยที่ผิดพลาดนั้นๆ พร้อมทั้งแนะนำวิธีรีแฟคทอริงที่ใช้ในการแก้ไขร่องรอยที่ผิดพลาดแต่ละประเภทและมีการประเมินความสามารถของมาตรวัดร่องรอยที่ผิดพลาดระหว่างซอร์ซโค้ดก่อนปรับแก้ไขด้วยวิธีรีแฟคทอริงและภายหลังจากปรับแก้ไขด้วยวิธีรีแฟคทอริง

จากงานวิจัยนี้แสดงให้เห็นว่าสามารถใช้มาตรวัดทางซอฟต์แวร์ช่วยในการตรวจสอบร่องรอยที่ผิดพลาดแต่ละประเภทได้ และนำเสนอวิธีรีแฟคทอริงในการแก้ไขร่องรอยที่ผิดพลาด แต่งานวิจัยนี้ไม่ได้อธิบายวิธีในการเลือกใช้งานวิธีรีแฟคทอริง เพียงแต่นำเสนอร่องรอยที่ผิดพลาดแต่ละลักษณะควรใช้วิธีใดบ้างในการแก้ไข

2.2.2 Facilitating Software Refactoring with Appropriate Resolution Order of Bad Smell [9]

งานวิจัยนี้นำเสนอการปรับแก้ไขโค้ดโดยพิจารณาจากความสัมพันธ์ระหว่างร่องรอยที่ผิดพลาดลักษณะต่างๆ เป็นหลัก การกระทบกันระหว่างร่องรอยที่ผิดพลาดเมื่อมีการแก้ไขซอร์ซโค้ด จากนั้นจึงนำมาจัดลำดับการแก้ไขร่องรอยที่ผิดพลาดว่าควรจะปรับแก้ไขร่องรอยที่ผิดพลาดใดก่อน-หลัง ลักษณะร่องรอยที่ผิดพลาดที่ใช้ในการทดลอง ได้แก่ Duplication Code, Divergent Change, Long Method, Large Class, Long Parameter List, Feature Envy, Useless Field, Useless Method, Useless Class และ Primitive Obsession

จากงานวิจัยนี้แสดงให้เห็นความสัมพันธ์ระหว่างร่องรอยที่ผิดพลาดเพื่อใช้เป็นเกณฑ์ในการเลือกปรับแก้ไขโค้ด แต่งานวิจัยนี้ไม่ได้ศึกษาในระดับของลำดับการใช้งานวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขร่องรอยที่ผิดพลาดลักษณะต่างๆ ว่าควรจะเลือกใช้และมีลำดับในการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดตามแต่ละร่องรอยที่ผิดพลาดอย่างไร

2.2.3 Analysing Refactoring Dependencies Using Graph Transformation [10]

งานวิจัยนี้นำเสนอการใช้งานวิธีรีแฟคทอริงด้วยรูปแบบของการเปลี่ยนรูปกราฟ (Graph Transformation) เพื่อวิเคราะห์ความสัมพันธ์ระหว่างวิธีรีแฟคทอริง 2 วิธีในรูปแบบต่างๆ โดยการค้นหาความสัมพันธ์นั้นผู้วิจัยใช้วิธีการวิเคราะห์เปรียบเทียบคู่ (Critical pair analysis) และวิธีการวิเคราะห์การขึ้นตรงต่อกันของวิธีรีแฟคทอริง (Sequentail dependency analysis) โดยรูปแบบความสัมพันธ์นั้นมีลักษณะดังนี้

1. ความสัมพันธ์แบบแยกจากกัน (Mutual Exclusions) เป็นความสัมพันธ์ที่เมื่อใช้วิธีรีแฟคทอริงวิธีใดวิธีหนึ่งแล้ว จะไม่สามารถใช้งานอีกวิธีหนึ่งได้

2. ความสัมพันธ์แบบไม่สมมาตรกัน (Asymmetrics Conflicts) เป็นความสัมพันธ์ที่สลับการใช้งานก่อน-หลังของวิธีรีแฟคทอริงทั้ง 2 วิธีในการปรับแก้ไขโค้ด จะให้ผลลัพธ์ที่ไม่เหมือนกัน

3. ความสัมพันธ์แบบขึ้นตรงต่อกัน (Sequential Dependencies) เป็นความสัมพันธ์แบบมีลำดับที่แน่นอนของการใช้วิธีรีแฟคทอริง 2 วิธี

จากงานวิจัยนี้แสดงให้เห็นว่าสามารถใช้วิธีการเปลี่ยนรูปกราฟในการนำเสนอการใช้งานวิธีรีแฟคทอริงระหว่างวิธีรีแฟคทอริงที่มีความสัมพันธ์แบบต่างๆ ได้ ซึ่งงานวิจัยนี้ศึกษาเฉพาะการจำแนกความสัมพันธ์ระหว่างวิธีรีแฟคทอริงแต่ไม่ได้นำเสนอว่าใช้เกณฑ์ใดในการเลือกวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดในกรณีที่วิธีรีแฟคทอริง 2 วิธีนั้นไม่มีความสัมพันธ์ตรงกันลักษณะความสัมพันธ์ที่ได้จำแนกไว้แบบใดแบบหนึ่ง

2.2.4 A Constrain Programming Approach to Conflict-aware Scheduling of Prioritized Code Clone Refactoring [11]

งานวิจัยนี้นำเสนอวิธีการปรับปรุงคุณภาพการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดโดยการสร้างโมเดลความพยายาม (Effort Model) ที่ใช้ตัววัดคุณภาพเชิงวัตถุแบบคิวมูท (QMOOD) [7] ในการประมาณค่าความพยายามสำหรับการใช้วิธีรีแฟคทอริงในการปรับแก้ไขโค้ดซ้ำ (Clone Code) และได้จำแนกความสัมพันธ์ระหว่างวิธีรีแฟคทอริงออกเป็นลักษณะ ได้แก่ ความสัมพันธ์แบบมีลำดับ (Sequential dependency) ความสัมพันธ์แบบแยกจากกัน (Mutual inclusion) เป็นต้น เพื่อใช้เป็นเกณฑ์ในการกำหนดน้ำหนักสำหรับคำนวณหาค่าความพยายาม จากนั้นจึงสร้างสูตรคำนวณในการกำหนดการใช้งานวิธีรีแฟคทอริง โดยขจัดงานที่ทำให้เกิดความขัดแย้งกันหรือวิธีรีแฟคทอริงที่ขัดแย้งกันออกไปตามหลักแนวคิดการเขียนโปรแกรมแบบมีเงื่อนไข (Constrain Programming) เพื่อให้ได้ซอร์ซโค้ดภายหลังจากปรับแก้ไขด้วยวิธีรีแฟคทอริงที่มีคุณภาพมากที่สุดและใช้ความพยายามน้อยที่สุด

จากงานวิจัยนี้แสดงให้เห็นแค่การขจัดความขัดแย้งระหว่างวิธีรีแฟคทอริงเป็นหลัก เพื่อทำให้การปรับแก้ไขโค้ดการทำงานด้วยวิธีรีแฟคทอริงนั้นเสียค่าความพยายามน้อยที่สุด แต่ไม่ได้ระบุลำดับการใช้งานวิธีรีแฟคทอริงที่เหลือหลังจากที่ได้ขจัดความขัดแย้งแล้วว่ามีวิธีการเลือกใช้งานอย่างไร

2.2.5 Searching for Opportunities of Refactoring Sequences : Reducing the Search Space [12]

งานวิจัยนี้นำเสนอแนวคิดในการลดจำนวนการใช้งานวิธีรีแฟคทอริงในบรรดากลุ่มวิธีรีแฟคทอริงที่เป็นไปได้ในการแก้ไขปัญหา โดยพิจารณาจากความสัมพันธ์ 5 ลักษณะในการจัดลำดับ ได้แก่ เท่าเทียมกัน (Equivalent) สับเปลี่ยน (Commutative) ตรงกันข้าม (Inverse) ต้องห้าม (Forbidden) และคู่ขนาน (Parallel) เพื่อหาวิธีรีแฟคทอริงที่เป็นไปได้สำหรับการจัดลำดับวิธีรีแฟคทอริง โดยใช้กราฟดีเทอมินีสติก ไฟไนท์ ออโตมาตา (Deterministic Finite Automata -

DFA) ในการนำเสนอและใช้ลักษณะความสัมพันธ์ต่างๆ เป็นเกณฑ์ในการลดขั้นตอนการหา (Search space) วิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขโค้ด

จากงานวิจัยนี้ผู้วิจัยศึกษาในส่วนของการขจัดวิธีรีแฟคทอริงให้เหลือน้อยที่สุดก่อนที่จะนำไปใช้ปรับแก้ไขโค้ด แต่ภายหลังจากที่ได้ขจัดจนเหลือวิธีรีแฟคทอริงสำหรับใช้งานจริงแล้ว งานวิจัยนี้ไม่ได้นำเสนอลำดับการเลือกใช้งานวิธีรีแฟคทอริงที่เหลือในการปรับแก้ไขโค้ดว่าควรจะเลือกวิธีรีแฟคทอริงใดก่อน-หลังในการปรับแก้ไขโค้ด

2.2.6 Assessment of Maintainability Metrics for Object-Oriented Software System [13]

งานวิจัยนี้ศึกษาหาความสัมพันธ์ระหว่างมาตรวัดทั้ง 6 ชนิดของซีเค (CK Metrics) ได้แก่ ผลรวมค่าความซับซ้อนต่อคลาส (Weighted Method per Class) มาตรวัดความรับผิดชอบของคลาส (Response sets for a class - RFC) ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส (Lack of Cohesion in Methods - LCOM) มาตรวัดการเข้าคู่กันระหว่างวัตถุ (Coupling Between Objects - CBO) มาตรวัดระดับความลึกของการสืบทอดคุณสมบัติของคลาส (Depth of Inheritance of Tree in class - DIT) และมาตรวัดจำนวนคลาสลูก (Depth Number of Children of Class - NOC) นั้นมีผลกระทบกับคุณลักษณะด้านใดบ้างของค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ (Maintainability) เช่น ความซับซ้อนของระบบ (Complexity) ความสามารถในการนำส่วนของซอฟต์แวร์กลับมาใช้งานใหม่ (Reusability) ความสามารถในการทดสอบระบบ (Testability) ความสามารถในการทำความเข้าใจระบบ (Understandability) เป็นต้น และได้สรุปความสัมพันธ์แสดงได้ดังตารางที่ 5 มีรายละเอียดดังต่อไปนี้

ตารางที่ 5 แสดงความสัมพันธ์ระหว่างมาตรวัดซีเคกับความสามารถในการบำรุงรักษาซอฟต์แวร์ [13]

มาตรวัดซีเค	คุณลักษณะของความสามารถในการบำรุงรักษาซอฟต์แวร์ที่เกี่ยวข้อง	ความสัมพันธ์ระหว่างค่าความสามารถในการบำรุงรักษาซอฟต์แวร์กับมาตรวัด
ผลรวมค่าความซับซ้อนต่อคลาส	- ความสามารถในการทดสอบ - ความซับซ้อนของระบบ	แปรผกผัน
มาตรวัดความรับผิดชอบของคลาส	- ความสามารถในการทำความเข้าใจระบบ - ความสามารถในการทดสอบระบบ	แปรผกผัน
ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	- ความสามารถในการนำกลับมาใช้งานใหม่	แปรผกผัน

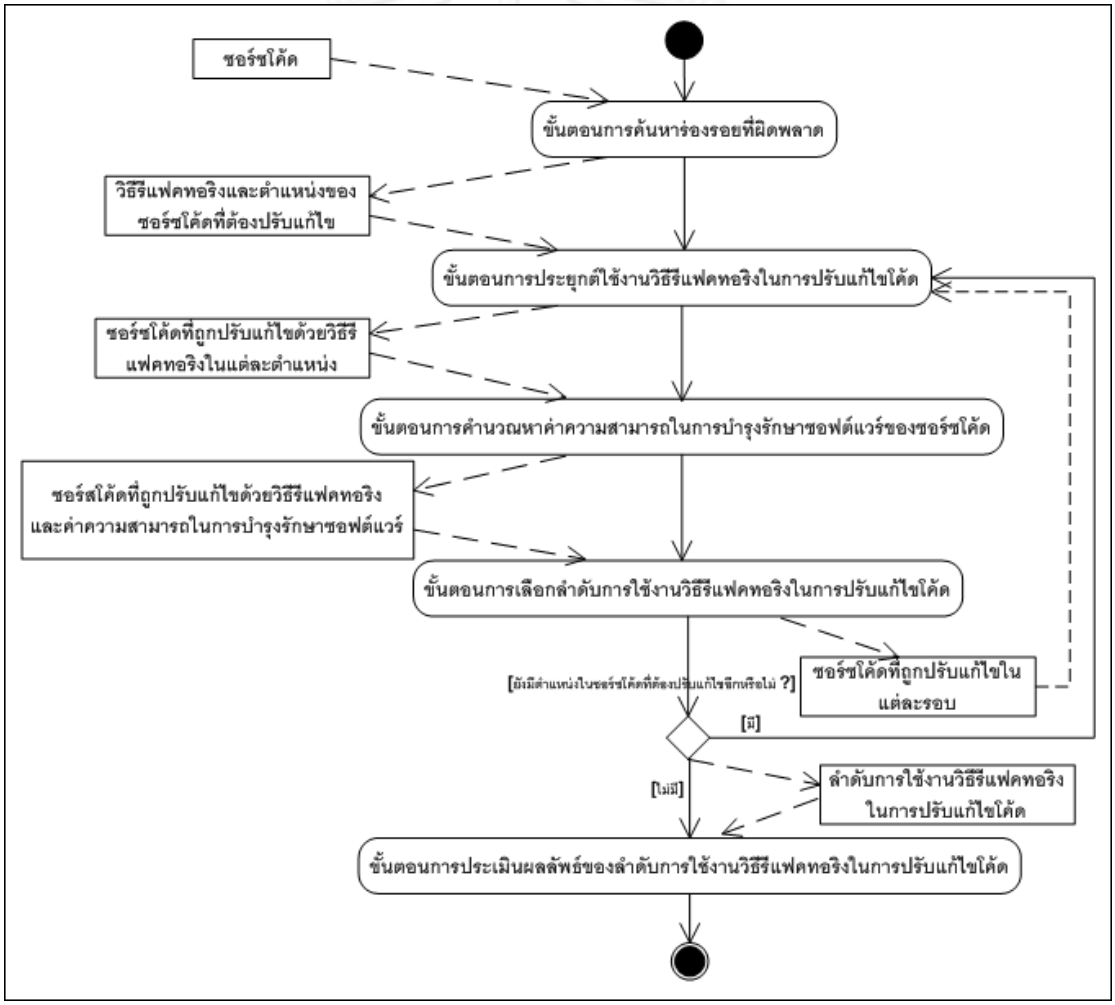
มาตรวัดการเข้าคู่กันระหว่างวัตถุ	- ความสามารถในการนำกลับมาใช้งานใหม่	แปรผกผัน
มาตรวัดระดับความลึกของการสืบทอดคุณสมบัติของคลาส	- ความสามารถในการนำกลับมาใช้งานใหม่	แปรผกผัน
มาตรวัดจำนวนคลาสลูก	- ความซับซ้อนของระบบ	แปรผกผัน

2.2.7 Size and Frequency of Class Change from a Refactoring Perspective [14]

งานวิจัยนี้ศึกษาการค้นหาคลาสที่นำมาทำการปรับแก้ไขด้วยวิธีรีแฟคทอริงโดยใช้เกณฑ์ของจำนวนเวอร์ชัน (version) ของคลาสที่มีการเปลี่ยนแปลงและจำนวนบรรทัดของคลาส (lines of code) ที่ได้เพิ่มเติมเข้าไป โดยทำการทดลองกับซอร์ซโค้ดของไลบารี่ของจาวาคลาส 3 ชนิด คือ เอดับเบิ้ลยูที (AWT) ไอโอ (IO) และ แลงค์ (Lang) ที่มีเวอร์ชันการพัฒนาในรอบ 3 ปี เพื่อตรวจสอบคุณลักษณะคลาสที่มีจำนวนเวอร์ชันการเปลี่ยนแปลงบ่อยครั้งและจำนวนบรรทัดที่ได้เพิ่มเติมเข้าไปนำมาสร้างเป็นกราฟเพื่อดูแนวโน้มของคลาสที่จะต้องนำมาแก้ไข ผู้วิจัยได้สรุปว่าคลาสที่มีความถี่ในการเปลี่ยนแปลงสูงและคลาสที่มีขนาดใหญ่หรือจำนวนบรรทัดมาก จะมีโอกาสที่เกิดร่องรอยที่ผิดพลาดมากกว่าคลาสที่มีความถี่ในการเปลี่ยนแปลงต่ำหรือคลาสที่มีขนาดเล็ก

บทที่ 3
วิธีการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด โดยใช้มาตรวัดเชิงวัตถุและอัลกอริทึมละโมบ

วิธีการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด โดยใช้มาตรวัดเชิงวัตถุและอัลกอริทึมละโมบแบ่งออกเป็น 5 ขั้นตอนหลักดังภาพที่ 8 ได้แก่ ขั้นตอนการค้นหาร่องรอยที่ผิดพลาด ขั้นตอนการประยุกต์ใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด ขั้นตอนการคำนวณหาค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ของซอร์ซโค้ด ขั้นตอนการเลือกลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด และขั้นตอนการประเมินผลลัพธ์ของลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด



ภาพที่ 8 แผนภาพแอคทิวิตีแสดงขั้นตอนการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด โดยใช้มาตรวัดเชิงวัตถุและอัลกอริทึมละโมบ

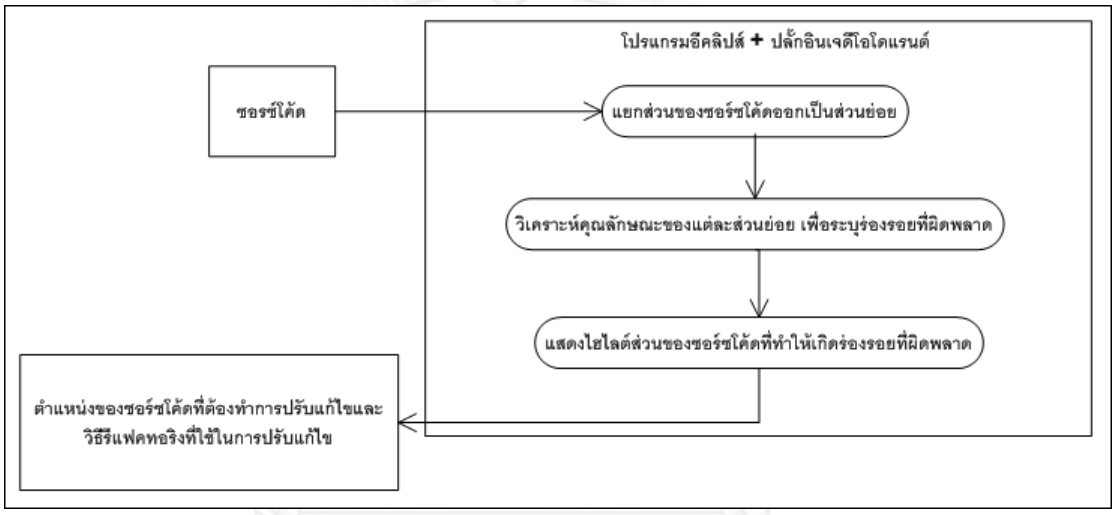
จากภาพที่ 8 วิธีการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดนั้นจะเริ่มจากการค้นหาร่องรอยที่ผิดพลาดในซอร์ซโค้ดของข้อมูลนำเข้า เพื่อหาตำแหน่งที่เป็นสาเหตุของร่องรอยที่ผิดพลาดและวิธีรีแฟคทอริงที่ใช้ในการขจัดร่องรอยที่ผิดพลาด โดยขั้นตอนนี้จะใช้เครื่องมือนำมาช่วยในการค้นหา จากนั้นจึงนำวิธีรีแฟคทอริงแต่ละวิธีมาปรับแก้ไขโค้ดเพื่อขจัดร่องรอยที่ผิดพลาดในแต่ละตำแหน่ง โดยในการปรับแก้ไขโค้ดนั้นจะเลือกใช้วิธีรีแฟคทอริงเพียง 1 วิธี ในการปรับแก้ไขตำแหน่งที่เป็นสาเหตุของร่องรอยที่ผิดพลาดเพียง 1 ตำแหน่งต่อ 1 รอบของการปรับแก้ไขเท่านั้น เพื่อหารูปแบบของซอร์ซโค้ดที่เป็นไปได้ทั้งหมดในการปรับแก้ไขโค้ดในแต่ละรอบ โดยมองการปรับแก้ไขโค้ดในแต่ละตำแหน่งด้วยวิธีรีแฟคทอริงต่างๆ เป็นเส้นทางเดิน (path) สำหรับการค้นหาลำดับในการปรับแก้ไขโค้ด หลังจากที่ได้เส้นทางเดินของแต่ละตำแหน่งที่ปรับแก้ไขแล้วนั้น จะนำแต่ละเส้นทางมาคำนวณหาค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ เพื่อใช้เป็นเกณฑ์ในการตัดสินใจเลือกวิธีรีแฟคทอริงในการแก้ไขซอร์ซโค้ดในแต่ละรอบ

เมื่อได้ค่าความสามารถในการบำรุงรักษาของทุกเส้นทางเดินในการแก้ไขโค้ดแล้ว ในขั้นตอนการเลือกลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดจะทำการเลือกเส้นทางหรือวิธีรีแฟคทอริงที่ใช้ปรับแก้ไขโค้ดแล้วได้ซอร์ซโค้ดที่มีค่าความสามารถในการบำรุงรักษามากที่สุดเป็นเส้นทางที่จะเลือกเดินในแต่ละรอบตามหลักการของอัลกอริทึมละโมบ เพื่อนำซอร์ซโค้ดที่ปรับแก้ไขด้วยวิธีรีแฟคทอริงที่ได้เลือกนั้นมาเป็นซอร์ซโค้ดต้นแบบสำหรับการหาลำดับการใช้งานวิธีรีแฟคทอริงในรอบถัดไป การค้นหาลำดับจะเป็นลักษณะวนรอบการทำงานเช่นนี้ไปเรื่อยๆ จนกระทั่งขจัดร่องรอยที่ผิดพลาดจนหมดหรือปรับแก้ไขโค้ดจนครบทุกตำแหน่งแล้ว จึงหยุดการวนรอบการทำงาน สุดท้ายจะได้ลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดที่เมื่อนำไปใช้ปรับแก้ไขจะได้ซอร์ซโค้ดที่มีค่าความสามารถในการบำรุงรักษามากที่สุด ในส่วนของขั้นตอนการประเมินผลลัพธ์ของลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดนั้นจะทำการเปรียบเทียบผลลัพธ์ของซอร์ซโค้ดที่ปรับแก้ไขโดยการจัดลำดับการใช้งานวิธีรีแฟคทอริงกับซอร์ซโค้ดที่ปรับแก้ไขโดยที่ไม่พิจารณาการจัดลำดับการใช้งานวิธีรีแฟคทอริงหรือการสุ่มเลือกวิธีรีแฟคทอริงในการแก้ไข เพื่อเป็นการยืนยันว่าการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดจะทำให้ได้ซอร์ซโค้ดภายหลังจากการปรับแก้ไขที่มีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์มากกว่าซอร์ซโค้ดที่ปรับแก้ไขโค้ดโดยไม่พิจารณาการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไข และเปรียบเทียบผลลัพธ์ที่ได้กับอัลกอริทึมอื่นๆ ได้แก่ อัลกอริทึมค้นหาแนวกว้าง อัลกอริทึมปีนเขา และอัลกอริทึมเอสตาร์ เพื่อเป็นการยืนยันว่าการใช้งานอัลกอริทึมละโมบจะได้ผลลัพธ์ที่ดีกว่าอัลกอริทึมอื่นๆ

3.1 ขั้นตอนการค้นหาร่องรอยที่ผิดพลาด

เป็นขั้นตอนการค้นหาตำแหน่งของร่องรอยที่ผิดพลาดที่มีอยู่ในซอร์ซโค้ดนำเข้าและวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขร่องรอยที่ผิดพลาดนั้น เพื่อนำมาสร้างเส้นทางที่เป็นไปได้ในการในการปรับแก้ไขโค้ดด้วยวิธีรีแฟคทอริงในขั้นตอนการประยุกต์ใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดต่อไป โดยลักษณะของร่องรอยที่ผิดพลาดที่ต้องการค้นหามี 3 ลักษณะ ได้แก่ ร่องรอยที่ผิดพลาดแบบเมท็อดที่มีความยาวมาก ร่องรอยที่ผิดพลาดแบบคลาสที่มีขนาดใหญ่ และร่องรอยที่ผิดพลาดแบบฟีเจอร์เอนวี ซึ่งร่องรอยที่ผิดพลาดแบบเมท็อดที่มีความยาวมากและร่องรอยที่ผิดพลาดแบบคลาสที่มี

ขนาดใหญ่เป็นลักษณะของซอร์ซโค้ดที่มีการเขียนคำสั่งการทำงานเป็นจำนวนมาก มีโอกาสเปลี่ยนแปลงโค้ดการทำงานเพื่อเพิ่มเติมการทำงานใหม่หรือรองรับการทำงานใหม่ในอนาคต และลักษณะร่องรอยที่ผิดพลาดดังกล่าวก็มีโอกาสที่จะก่อให้เกิดข้อผิดพลาดของการทำงานที่เกิดจากการเปลี่ยนแปลงนั้นๆได้ (change-prone) ในส่วนของร่องรอยที่ผิดพลาดแบบฟีเจอร์เอนวีนั้นเป็นลักษณะของพฤติกรรมของคลาสหนึ่งที่มีการเรียกใช้งานเมท็อดหรือตัวแปรของคลาสอื่นมากกว่าคลาสที่เป็นเจ้าของ เป็นการออกแบบการวางเมท็อดที่มีคุณสมบัติไม่ตรงกับคุณลักษณะการทำงานของคลาส ซึ่งจะทำให้การแก้ไขนั้นทำได้ยาก ควรจะย้ายเมท็อดหรือตัวแปรของคลาสให้ตรงกับคุณลักษณะของคลาสที่เป็นเจ้าของการทำงาน การขจัดร่องรอยที่ผิดพลาดทั้ง 3 ลักษณะออกจากซอร์ซโค้ดจะทำให้ซอร์ซโค้ดนั้นง่ายต่อการปรับแก้ไขเพิ่มเติมการทำงานในภายหลัง และการหาข้อผิดพลาดจะทำได้ง่ายขึ้น

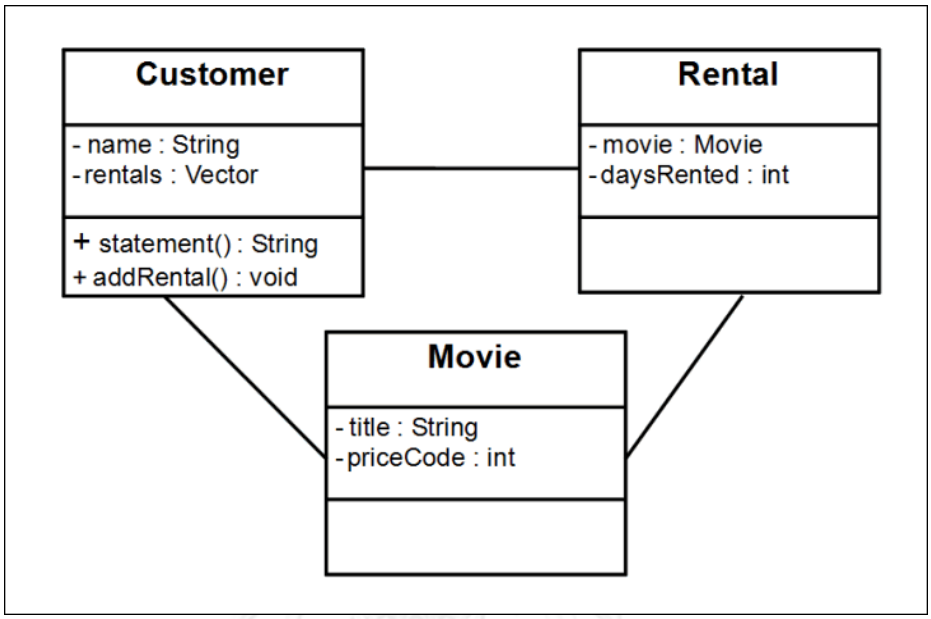


ภาพที่ 9 แผนภาพแอคทิวิตีแสดงขั้นตอนการค้นหาร่องรอยที่ผิดพลาดของโปรแกรมอีคลิปส์ที่ประยุกต์ใช้งานปลั๊กอินเจดีโดแรนต์

จากภาพที่ 9 ในส่วนของการค้นหาร่องรอยที่ผิดพลาดในซอร์ซโค้ดนั้นใช้โปรแกรมอีคลิปส์ (Eclipse) ที่ประยุกต์ใช้งานปลั๊กอิน (Plug in) เจดีโอโดแรนต์ (JDeodorant) [5] ที่มีคุณสมบัติในการค้นหาร่องรอยที่ผิดพลาดพร้อมทั้งระบุวิธีรีแฟคทอริงและตำแหน่งที่ต้องทำการแก้ไขเพื่อขจัดร่องรอยที่ผิดพลาด โดยเจดีโอโดแรนต์สามารถตรวจจับร่องรอยที่ผิดพลาดแบบเมท็อดที่มีความยาวมาก ร่องรอยที่ผิดพลาดแบบคลาสที่มีขนาดใหญ่ และร่องรอยที่ผิดพลาดแบบฟีเจอร์เอนวี โดยขั้นตอนในการค้นหาร่องรอยที่ผิดพลาดของปลั๊กอินเจดีโอโดแรนต์นั้นประกอบด้วย 3 ขั้นตอนหลัก ได้แก่ ขั้นตอนแยกส่วนของซอร์ซโค้ดออกเป็นส่วนย่อย ขั้นตอนวิเคราะห์คุณลักษณะของแต่ละส่วนย่อยเพื่อระบุร่องรอยที่ผิดพลาด และขั้นตอนแสดงไฮไลต์ส่วนของซอร์ซโค้ดที่ทำให้เกิดร่องรอยที่ผิดพลาด ภายหลังจากเสร็จสิ้นขั้นตอนการค้นหาจะได้ตำแหน่งของซอร์ซโค้ดที่ต้องทำการปรับแก้ไขและวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไข

โดยตัวอย่างซอร์ซโค้ดที่นำมาเป็นข้อมูลนำเข้ามีร่องรอยที่ผิดพลาดแบบเมท็อดที่มีความยาวมาก ร่องรอยที่ผิดพลาดแบบคลาสที่มีขนาดใหญ่ และร่องรอยที่ผิดพลาดแบบฟีเจอร์เอนวีดังภาพที่

11 โดยตัวอย่างซอร์ซโค้ดข้อมูลนำเข้าและใช้ในขั้นตอนถัดไปในการจัดลำดับการใช้งานวิธีรีแฟคทอริงนั้นเป็นซอร์ซโค้ดการทำงานของระบบเช่าภาพยนตร์ [1] ประกอบด้วยคลาสหลัก 3 คลาส ได้แก่ คลาส Customer, คลาส Rental และคลาส Movie แผนภาพความสัมพันธ์ของคลาสของระบบเช่าภาพยนตร์ แสดงได้ดังภาพที่ 10



ภาพที่ 10 แผนภาพความสัมพันธ์ของคลาสของระบบเช่าภาพยนตร์

จากแผนภาพที่ 10 คลาส Customer เป็นคลาสที่เก็บข้อมูลของลูกค้า ประกอบด้วยเมท็อด Statement สำหรับคำนวณค่าเช่าภาพยนตร์ในแต่ละครั้งของลูกค้า โดยการนำรายละเอียดข้อมูลการเช่าภาพยนตร์จากคลาส Rental มาคำนวณหาค่าเช่าภาพยนตร์ (totalAmount) และแต้มความถี่สะสมในการเช่าภาพยนตร์ (frequentRenterPoint) ในแต่ละครั้ง โดยแสดงผลลัพธ์การคำนวณออกทางหน้าจอ (console) คลาส Rental เป็นคลาสที่เก็บข้อมูลการเช่าภาพยนตร์ของลูกค้า และคลาส Movie เป็นคลาสที่เก็บข้อมูลรายละเอียดของภาพยนตร์สำหรับให้ลูกค้าเช่า เมื่อนำตัวอย่างซอร์ซโค้ดของระบบเช่าภาพยนตร์มาหาร่องรอยที่ผิดพลาดนั้นจะพบร่องรอยที่ผิดพลาดในเมท็อด Statement ของคลาส Customer จะพบว่าเมท็อด Statement นั้นมีหน้าที่การทำงานมากเกินกว่าที่เมท็อดหนึ่งๆ ควรจะทำงาน กล่าวคือ เมท็อด Statement มีโค้ดส่วนของการคำนวณหาค่าเช่าภาพยนตร์และแต้มความถี่สะสมในการเช่าภาพยนตร์ซึ่งการทำงานดังกล่าวควรจะเป็นหน้าที่ของคลาส Rental รับผิดชอบการทำงาน จึงควรปรับแก้ไขการทำงานของเมท็อด Statement ให้เหลือเหน้าที่เฉพาะการแสดงผลลัพธ์การคำนวณค่าต่างๆ ภายทางหน้าจอเท่านั้น โดยรายละเอียดของร่องรอยที่ผิดพลาดและวิธีรีแฟคทอริงที่ใช้แก้ไขร่องรอยที่ผิดพลาดมีรายละเอียดดังนี้

- ตำแหน่งที่ 1 (บรรทัดที่ 8-23) เป็นส่วนของการคำนวณค่าเช่าภาพยนตร์ของลูกค้า ซึ่งรายละเอียดส่วนนี้มีจำนวนบรรทัดค่อนข้างมาก และมีการเรียกใช้คลาส Rental มาช่วยในการคำนวณหาค่าเช่าภาพยนตร์ ทำให้เกิดลักษณะของร่องรอยที่ผิดพลาดแบบฟีเจอร์เอ็นวี ซอร์ซโค้ดตำแหน่งนี้สามารถปรับแก้ไขได้ด้วยวิธีรีแฟคทอริงแบบ Extract Method ในการแยกซอร์ซโค้ดส่วน

ของการคำนวณค่าเช่าภาพยนตร์ไปเป็นเมท็อดใหม่ภายใต้คลาส Customer แทนด้วยสัญลักษณ์ R1 แสดงได้ดังภาพที่ 12 หรือสามารถใช้วิธีรีแฟคทอริงแบบ Move Method ในการย้ายซอร์ซโค้ดส่วนของการคำนวณค่าเช่าภาพยนตร์ไปเป็นเมท็อดใหม่ภายใต้คลาส Rental แทนด้วยสัญลักษณ์ R2 แสดงได้ดังภาพที่ 13

```
01 public class Customer {
02         public String statement(){
03                 double totalAmount = 0;
04                 int frequentRenterPoints = 0;
05                 Enumeration rentals = _rentals.elements();
06                 String result = "Rental Record for " + getName() + "\n";
07                 while (rentals.hasMoreElements()) {
08                         double thisAmount = 0;
09                         Rental each = (Rental) rentals.nextElement();
10                         switch (each.getMovie().getPriceCode()) {
11                         case Movie.REGULAR:
12                                 thisAmount += 2;
13                                 if (each.getDaysRented() > 2)
14                                         thisAmount += (each.getDaysRented() - 2) * 1.5;
15                                 break;
16                         case Movie.NEW_RELEASE:
17                                 thisAmount += each.getDaysRented() * 3;
18                                 break;
19                         case Movie.CHILDRENS:
20                                 thisAmount += 1.5;
21                                 if (each.getDaysRented() > 3)
22                                         thisAmount += (each.getDaysRented() - 3) * 1.5;
23                                 break;
24                         }
25                         frequentRenterPoints ++;
26                         if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) && each.getDaysRented() > 1) frequentRenterPoints ++;
27                         result += "\t" + each.getMovie().getTitle()+ "\t" + String.valueOf(thisAmount) + "\n";
28                         totalAmount += thisAmount;
29                 }
30                 result += "Amount owed is " + String.valueOf(totalAmount) + "\n";
31                 result += "You earned " + String.valueOf(frequentRenterPoints) + " frequent renter points";
32                 return result;
33         }
```

ภาพที่ 11 ซอร์ซโค้ดเมท็อด Statement ที่จะทำการปรับแก้ไขโดยวิธีรีแฟคทอริง

```
01 public class Customer {
02         public String statement(){
03                 double totalAmount = 0;
04                 int frequentRenterPoints = 0;
05                 Enumeration rentals = _rentals.elements();
06                 String result = "Rental Record for " + getName() + "\n";
07                 while (rentals.hasMoreElements()) {
08                         double thisAmount = 0;
09                         Rental each = (Rental) rentals.nextElement();
10                         thisAmount = amountFor(each);
11
12                         frequentRenterPoints ++;
13
14                         if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) && each.getDaysRented() > 1) frequentRenterPoints ++;
15                         result += "\t" + each.getMovie().getTitle()+ "\t" + String.valueOf(thisAmount) + "\n";
16                         totalAmount += thisAmount;
17                 }
18
19                 result += "Amount owed is " + String.valueOf(totalAmount) + "\n";
20                 result += "You earned " + String.valueOf(frequentRenterPoints) + " frequent renter points";
21                 return result;
22         }
23
24         private int amountFor(Rental each) {
25                 int thisAmount = 0;
26                 switch (each.getMovie().getPriceCode()) {
27                 case Movie.REGULAR:
28                         thisAmount += 2;
29                 if (each.getDaysRented() > 2)
30                         thisAmount += (each.getDaysRented() - 2) * 1.5;
31                         break;
32                 case Movie.NEW_RELEASE:
33                         thisAmount += each.getDaysRented() * 3;
34                         break;
35                 case Movie.CHILDRENS:
36                         thisAmount += 1.5;
37                         if (each.getDaysRented() > 3)
38                                 thisAmount += (each.getDaysRented() - 3) * 1.5;
39                         break;
40                 }
41                 return thisAmount;
42         }
43}
```

ภาพที่ 12 ซอร์ซโค้ดเมท็อด Statement ที่ปรับแก้ไขในตำแหน่งที่ 1 ด้วยวิธีรีแฟคทอริงแบบ Extract Method (R1)

```
01 public class Customer {
02        public String statement(){
03                double totalAmount = 0;
04                int frequentRenterPoints = 0;
05                Enumeration rentals = _rentals.elements();
06                String result = "Rental Record for " + getName() + "\n";
07                while (rentals.hasMoreElements()) {
08                        double thisAmount = 0;
09                        Rental each = (Rental) rentals.nextElement();
10                        thisAmount = each.getCharge();
11
12                        frequentRenterPoints ++;
13
14                        if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) && each.getDaysRented() > 1) frequentRenterPoints ++;
15
16                        result += "\t" + each.getMovie().getTitle()+ "\t" + String.valueOf(thisAmount) + "\n";
17                        totalAmount += thisAmount;
18                }
19                //add footer lines
20                result += "Amount owed is " + String.valueOf(totalAmount) + "\n";
21                result += "You earned " + String.valueOf(frequentRenterPoints) + " frequent renter points";
22                return result;
23
24        }
25}
26
27 public class Rental {
28        public double getCharge() {
29                double result = 0;
30                switch (getMovie().getPriceCode()) {
31                case Movie.REGULAR:
32                        result += 2;
33                        if (getDaysRented() > 2)
34                                result += (getDaysRented() - 2) * 1.5;
35                        break;
36                case Movie.NEW_RELEASE:
37                        result += getDaysRented() * 3;
38                        break;
39                case Movie.CHILDRENS:
40                        result += 1.5;
41                        if (getDaysRented() > 3)
42                                result += (getDaysRented() - 3) * 1.5;
43                        break;
44                }
45                return result;
46        }
47}
```

ภาพที่ 13 ซอร์ซโค้ดเมท็อด Statement ที่ปรับแก้ไขในตำแหน่งที่ 1 ด้วยวิธีรีแฟคทอริงแบบ Move Method (R2)

- ตำแหน่งที่ 2 (บรรทัดที่ 25-26) เป็นส่วนของการคำนวณค่าแต้มความถี่สะสมในการเช่าภาพยนตร์ของลูกค้า ซอร์ซโค้ดส่วนนี้มีการเรียกใช้งานคลาส Rental มาช่วยในการคำนวณหาค่าความถี่สะสมในการเช่าภาพยนตร์ ทำให้เกิดลักษณะของร่องรอยที่ผิดพลาดแบบฟีเจอร์เอ็นวี ซอร์ซโค้ดตำแหน่งนี้สามารถปรับแก้ไขด้วยวิธีรีแฟคทอริงแบบ Extract Method ในการแยกซอร์ซโค้ดส่วนของการคำนวณค่าแต้มความถี่สะสมในการเช่าภาพยนตร์ไปเป็นเมท็อดใหม่ภายใต้คลาส Customer แทนด้วยสัญลักษณ์ R3 แสดงได้ดังภาพที่ 14 หรือสามารถใช้วิธีรีแฟคทอริงแบบ Move Method ในการย้ายซอร์ซโค้ดส่วนของการคำนวณแต้มความถี่สะสมไปเป็นเมท็อดใหม่ภายใต้คลาส Rental แทนด้วยสัญลักษณ์ R4 แสดงได้ดังภาพที่ 15

```
01 public class Customer {
02        public String statement(){
03                double totalAmount = 0;
04                Enumeration rentals = _rentals.elements();
05
06                String result = "Rental Record for " + getName() + "\n";
07
08                while (rentals.hasMoreElements()) {
09                        double thisAmount = 0;
10                        Rental each = (Rental) rentals.nextElement();
11
12                        switch (each.getMovie().getPriceCode()) {
13                        case Movie.REGULAR:
14                                thisAmount += 2;
15                                if (each.getDaysRented() > 2)
16                                        thisAmount += (each.getDaysRented() - 2) * 1.5;
17                                break;
18                        case Movie.NEW_RELEASE:
19                                thisAmount += each.getDaysRented() * 3;
20                                break;
21                        case Movie.CHILDRENS:
22                                thisAmount += 1.5;
23                                if (each.getDaysRented() > 3)
24                                        thisAmount += (each.getDaysRented() - 3) * 1.5;
25                                break;
26                        }
27
28                        result += "\t" + each.getMovie().getTitle()+ "\t" + String.valueOf(thisAmount) + "\n";
29                        totalAmount += thisAmount;
30                }
31
32                result += "Amount owed is " + String.valueOf(totalAmount) + "\n";
33                result += "You earned " + String.valueOf(getTotalFrequentRenterPoints()) + " frequent renter points";
34
35                return result;
36
37        }
38
39        private int getTotalFrequentRenterPoints(){
40                int frequentRenterPoints = 0;
41                Enumeration rentals = _rentals.elements();
42                while (rentals.hasMoreElements()) {
43                        Rental each = (Rental) rentals.nextElement();
44
45                        frequentRenterPoints ++;
46
47                        if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) && each.getDaysRented() > 1)
48                                frequentRenterPoints ++;
49
50                }
51                return frequentRenterPoints;
52        }
53}
```


ภาพที่ 14 ซอร์ซโค้ดเมท็อด Statement ที่ปรับแก้ไขในตำแหน่งที่ 2 ด้วยวิธีรีแฟคทอริงแบบ Extract Method (R3)

```
01 public class Customer {
02        public String statement(){
03                double totalAmount = 0;
04                Enumeration rentals = _rentals.elements();
05
06                String result = "Rental Record for " + getName() + "\n";
07
08                while (rentals.hasMoreElements()) {
09                        double thisAmount = 0;
10                        Rental each = (Rental) rentals.nextElement();
11
12                        switch (each.getMovie().getPriceCode()) {
13                        case Movie.REGULAR:
14                                thisAmount += 2;
15                                if (each.getDaysRented() > 2)
16                                        thisAmount += (each.getDaysRented() - 2) * 1.5;
17                                break;
18                        case Movie.NEW_RELEASE:
19                                thisAmount += each.getDaysRented() * 3;
20                                break;
21                        case Movie.CHILDRENS:
22                                thisAmount += 1.5;
23                                if (each.getDaysRented() > 3)
24                                        thisAmount += (each.getDaysRented() - 3) * 1.5;
25                                break;
26                        }
27
28                        result += "\t" + each.getMovie().getTitle()+ "\t" + String.valueOf(thisAmount) + "\n";
29                        totalAmount += thisAmount;
30                }
31
32                result += "Amount owed is " + String.valueOf(totalAmount) + "\n";
33                result += "You earned " + String.valueOf(getTotalFrequentRenterPoints()) + " frequent renter
points";
34
35                return result;
36
37        }
38
39        private int getTotalFrequentRenterPoints(){
40                int result = 0;
41                Enumeration rentals = _rentals.elements();
42                while (rentals.hasMoreElements()) {
43                        Rental each = (Rental) rentals.nextElement();
44                        result += each.getFrequentRenterPoints();
45                }
46                return result;
47        }
48
49        public class Rental {
50                int getFrequentRenterPoints() {
51                        if ((getMovie().getPriceCode() == Movie.NEW_RELEASE) &&
52                                getDaysRented() > 1)
53                                return 2;
54                        else
55                                return 1;
56                }
57        }
```


ภาพที่ 15 ซอร์ซโค้ดเมท็อด Statement ที่ปรับแก้ไขในตำแหน่งที่ 2 ด้วยวิธีรีแฟคทอริงแบบ Move Method (R4)

- ตำแหน่งที่ 3 (บรรทัดที่ 28) เป็นส่วนของการคำนวณค่าเช่าภาพยนตร์รวมในการเช่าภาพยนตร์ครั้งหนึ่งๆ ของลูกค้า ซอร์ซโค้ดส่วนนี้จะมีการทำงานร่วมกันซอร์ซโค้ดส่วนของการคำนวณค่าเช่าภาพยนตร์ของลูกค้าแต่ละเรื่องมารวมกันเป็นยอดรวมของการเช่าภาพยนตร์ ซอร์ซโค้ดส่วนนี้สามารถแยกการทำงานออกจากเมท็อด Statement ได้โดยการใช้วิธีรีแฟคทอริงแบบ Extract Method แยกซอร์ซโค้ดส่วนนี้ไปเป็นเมท็อดใหม่ภายใต้คลาส Customer แทนด้วยสัญลักษณ์ R5 แสดงได้ดังภาพที่ 16 หรือใช้วิธีรีแฟคทอริงแบบ Move Method ในการย้ายซอร์ซโค้ดส่วนของการ

คำนวณค่าเช่าภาพยนตร์รวมไปเป็นเมท็อดใหม่ภายใต้คลาส Rental แทนด้วยสัญลักษณ์ R6 แสดงได้ดังภาพที่ 17

```
01 public class Customer {
02        public String statement(){
03                int frequentRenterPoints = 0;
04                Enumeration rentals = _rentals.elements();
05
06                String result = "Rental Record for " + getName() + "\n";
07
08                while (rentals.hasMoreElements()) {
09                        double thisAmount = 0;
10                        Rental each = (Rental) rentals.nextElement();
11
12                        switch (each.getMovie().getPriceCode()) {
13                        case Movie.REGULAR:
14                                thisAmount += 2;
15                                if (each.getDaysRented() > 2)
16                                        thisAmount += (each.getDaysRented() - 2) * 1.5;
17                                break;
18                        case Movie.NEW_RELEASE:
19                                thisAmount += each.getDaysRented() * 3;
20                                break;
21                        case Movie.CHILDRENS:
22                                thisAmount += 1.5;
23                                if (each.getDaysRented() > 3)
24                                        thisAmount += (each.getDaysRented() - 3) * 1.5;
25                                break;
26
27                        }
28
29                        frequentRenterPoints ++;
30
31                        if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) && each.getDaysRented() > 1) frequentRenterPoints ++;
32
33                        result += "\t" + each.getMovie().getTitle()+ "\t" + String.valueOf(thisAmount) + "\n";
34
35                }
36
37                result += "Amount owed is " + getTotalCharge() + "\n";
38                result += "You earned " + String.valueOf(frequentRenterPoints) + " frequent renter points";
39                return result;
40
41        }
42
43        private double getTotalCharge() {
44                Enumeration rentals = _rentals.elements();
45                double result = 0;
46
47                while (rentals.hasMoreElements()) {
48                        Rental each = (Rental) rentals.nextElement();
49
50                        switch (each.getMovie().getPriceCode()) {
51                        case Movie.REGULAR:
52                                result += 2;
53                                if (each.getDaysRented() > 2)
54                                        result += (each.getDaysRented() - 2) * 1.5;
55                                break;
56                        case Movie.NEW_RELEASE:
57                                result += each.getDaysRented() * 3;
58                                break;
59                        case Movie.CHILDRENS:
60                                result += 1.5;
61                                if (each.getDaysRented() > 3)
62                                        result += (each.getDaysRented() - 3) * 1.5;
63                                break;
64                        }
65
66                }
67                return result;
68        }
69}
```

ภาพที่ 16 ซอร์ซโค้ดเมท็อด Statement ที่ปรับแก้ไขในตำแหน่งที่ 3 ด้วยวิธีรีแฟคทอริงแบบ Extract Method (R5)

```
01 public class Customer {
02        public String statement(){
03                int frequentRenterPoints = 0;
04                Enumeration rentals = _rentals.elements();
05
06                String result = "Rental Record for " + getName() + "\n";
07
08                while (rentals.hasMoreElements()) {
09                        double thisAmount = 0;
10                        Rental each = (Rental) rentals.nextElement();
11
12                        switch (each.getMovie().getPriceCode()) {
13                        case Movie.REGULAR:
14                                thisAmount += 2;
15                                if (each.getDaysRented() > 2)
16                                        thisAmount += (each.getDaysRented() - 2) * 1.5;
17                                break;
18                        case Movie.NEW_RELEASE:
19                                thisAmount += each.getDaysRented() * 3;
20                                break;
21                        case Movie.CHILDRENS:
22                                thisAmount += 1.5;
23                                if (each.getDaysRented() > 3)
24                                        thisAmount += (each.getDaysRented() - 3) * 1.5;
25                                break;
26
27                        }
28
29                        frequentRenterPoints ++;
30
31                        if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) && each.getDaysRented() > 1) frequentRenterPoints ++;
32                                result += "\t" + each.getMovie().getTitle()+ "\t" + String.valueOf(thisAmount) + "\n";
33
34                }
35
36                result += "Amount owed is " + getTotalCharge() + "\n";
37                result += "You earned " + String.valueOf(frequentRenterPoints) + " frequent renter points";
38                return result;
39
40        }
41
42        private double getTotalCharge() {
43                double result = 0;
44                Enumeration rentals = _rentals.elements();
45                while (rentals.hasMoreElements()) {
46                        Rental each = (Rental) rentals.nextElement();
47                        result += each.getCharge();
48                }
49                return result;
50        }
51}
52
53 public class Rental {
54        public double getCharge() {
55                double result = 0;
56                switch (getMovie().getPriceCode()) {
57                case Movie.REGULAR:
58                        result += 2;
59                        if (getDaysRented() > 2)
60                                result += (getDaysRented() - 2) * 1.5;
61                        break;
62                case Movie.NEW_RELEASE:
63                        result += getDaysRented() * 3;
64                        break;
65                case Movie.CHILDRENS:
66                        result += 1.5;
67                        if (getDaysRented() > 3)
68                                result += (getDaysRented() - 3) * 1.5;
69                        break;
70                }
71                return result;
72        }
73 }
```

ภาพที่ 17 ซอร์ซโค้ดเมท็อด Statement ที่ปรับแก้ไขในตำแหน่งที่ 3 ด้วยวิธีรีแฟคทอริงแบบ Move Method (R6)

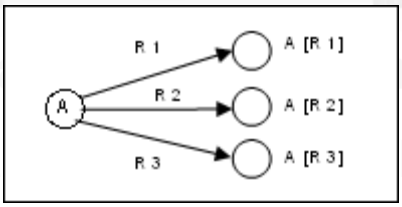
ภายหลังจากสิ้นสุดการค้นหาร่องรอยที่ผิดพลาดในซอร์ซโค้ดนำเข้า จะได้ตำแหน่งที่ต้องปรับแก้ไขเพื่อกำจัดร่องรอยที่ผิดพลาดและวิธีรีแฟคทอริงที่ใช้ในการแก้ไขโค้ด โดยจะนำทั้ง 2 อย่างดังกล่าวไปใช้งานในขั้นตอนการประยุกต์ใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดต่อไป

3.2 ขั้นตอนการประยุกต์ใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด

เป็นขั้นตอนในการนำตำแหน่งที่ต้องทำการปรับแก้ไขและวิธีรีแฟคทอริงที่ได้จากขั้นตอนการค้นหาร่องรอยที่ผิดพลาด นำมาสร้างเป็นเส้นทางในการใช้งานวิธีรีแฟคทอริง (Refactoring techniques usage path) เส้นทางในการใช้งานวิธีรีแฟคทอริง คือ เส้นทางที่บ่งบอกถึงการเปลี่ยนแปลงของซอร์ซโค้ดหนึ่งไปเป็นอีกซอร์ซโค้ดหนึ่งโดยการใช้งานวิธีรีแฟคทอริง จุดประสงค์ที่ต้องสร้างเส้นทางในการใช้งานวิธีรีแฟคทอริงเพื่อที่ต้องการหารูปแบบที่เป็นไปได้ในการปรับแก้ไขโค้ดหรือขอบเขตในการปรับแก้ไขโค้ดด้วยวิธีรีแฟคทอริง โดยเส้นทางในการใช้งานวิธีรีแฟคทอริงนั้นมี 2 ลักษณะ คือ เส้นทางที่เป็นทางเลือกสำหรับการใช้งานวิธีรีแฟคทอริง (Candidate refactoring techniques path) และเส้นทางที่เกิดจากการรวมกันของวิธีรีแฟคทอริงมากกว่า 1 วิธี (Combining refactoring techniques path) แต่ละลักษณะมีรายละเอียดต่อไปนี้

3.2.1 เส้นทางที่เป็นทางเลือกสำหรับการใช้งานวิธีรีแฟคทอริง (Candidate refactoring techniques path)

เป็นเส้นทางที่เกิดจากซอร์ซโค้ดที่สามารถปรับแก้ไขได้ด้วยวิธีรีแฟคทอริงมากกว่า 1 วิธี ซึ่งในการปรับแก้ไขนั้นจะสามารถเลือกใช้งานวิธีรีแฟคทอริงได้เพียงแค่ 1 วิธีในการปรับแก้ไข ถ้าเลือกวิธีใดวิธีหนึ่งแล้วจะไม่สามารถเลือกอีกวิธีหนึ่งได้ ตัวอย่างเช่น กำหนดให้เมท็อด A สามารถปรับแก้ไขได้ด้วยวิธีรีแฟคทอริงทั้ง 3 แบบ ได้แก่ วิธีรีแฟคทอริงแบบ Extract Method เพื่อลดจำนวนบรรทัดของเมท็อดลง (กำหนดให้แทนด้วยสัญลักษณ์ R1) วิธีรีแฟคทอริงแบบ Move Method เพื่อย้ายเมท็อดการทำงานไปยังคลาสอื่นแทน (กำหนดให้แทนด้วยสัญลักษณ์ R2) และวิธีรีแฟคทอริงแบบ Pull up method เพื่อย้ายการทำงานไปยังคลาสพ่อ (super class) แทน (กำหนดให้แทนด้วยสัญลักษณ์ R3) เส้นทางในการใช้งานวิธีรีแฟคทอริงสำหรับปรับแก้ไขเมท็อดA นั้นแสดงได้ดังภาพที่ 18



ภาพที่ 18 แผนภาพต้นไม้แสดงเส้นทางที่เป็นทางเลือกในการใช้งานวิธีรีแฟคทอริงสำหรับปรับแก้ไขเมท็อด A

จากภาพที่ 18 แผนภาพต้นไม้แสดงเส้นทางในการใช้งานวิธีรีแฟคทอริงนั้นประกอบด้วยสัญลักษณ์ดังต่อไปนี้

- โหนด (วงกลม) หมายถึง ซอร์ซโค้ดที่จะถูกปรับแก้ไขด้วยวิธีรีแฟคทอริง

- Ri หมายถึง วิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไข

- ลูกศร หมายถึง เส้นทางที่เป็นไปได้สำหรับการปรับแก้ไขโค้ดด้วยวิธีรีแฟคทอริงในแบบต่างๆ

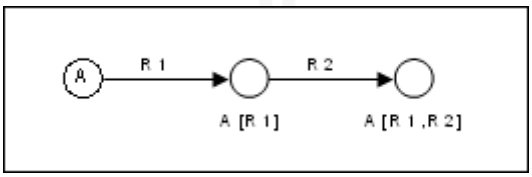
จำนวนเส้นทางที่เป็นไปได้ทั้งหมดสำหรับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดนั้น สามารถค้นหาได้จากสูตรดังต่อไปนี้

จำนวนเส้นทางที่เป็นไปได้ทั้งหมดสำหรับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด
= จำนวนวิธีรีแฟคทอริงที่สามารถปรับแก้ไขโค้ดได้

จากภาพที่ 18 จะเห็นได้ว่าเมท็อด A สามารถเลือกปรับแก้ไขด้วยวิธีรีแฟคทอริงได้ 3 แบบ คือ R1 หรือ R2 หรือ R3 โดยภายหลังจากที่ปรับแก้ไขด้วยวิธีรีแฟคทอริงแล้วจะได้ซอร์ซโค้ดในรูปแบบที่แตกต่างกัน คือ ซอร์ซโค้ด A[R1] ที่เกิดจากการปรับแก้ไขโค้ดโดยวิธีแฟคทอริง R1 ซอร์ซโค้ด A[R2] ที่เกิดจากการปรับแก้ไขโค้ดวิธีรีแฟคทอริง R2 และซอร์ซโค้ด A[R3] ที่เกิดจากการปรับแก้ไขโค้ดโดยวิธีรีแฟคทอริง R3 ดังนั้นจำนวนเส้นทางที่เป็นไปได้ทั้งหมดสำหรับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขเมท็อด A จึงมีค่าเท่ากับ 3 เส้นทาง

3.2.2 เส้นทางที่เกิดจากการรวมกันของวิธีรีแฟคทอริงมากกว่า 1 วิธี (Combining refactoring techniques path)

เป็นเส้นทางที่เกิดจากซอร์ซโค้ดที่ถูกปรับแก้ไขด้วยวิธีรีแฟคทอริงมาก่อนหน้านี้แล้วถูกนำมาปรับแก้ไขด้วยวิธีรีแฟคทอริงด้วยวิธีอื่นหรือปรับแก้ไขที่ตำแหน่งอื่นต่อไป ทำให้ซอร์ซโค้ดนั้นๆ ถูกปรับแก้ไขด้วยวิธีแฟคทอริงมากกว่า 1 วิธีตามลำดับการปรับแก้ไข ตัวอย่างเช่น เมท็อด A ถูกปรับแก้ไขด้วยวิธีรีแฟคทอริงแบบ Move Method (กำหนดให้แทนด้วยสัญลักษณ์ R1) มาก่อนหน้านี้แล้ว ทำให้ซอร์ซโค้ดภายหลังจากการปรับแก้ไขกลายเป็น A[R1] จากนั้นจึงนำซอร์ซโค้ดที่ถูกปรับแก้ไขด้วยวิธีรีแฟคทอริงแบบ Move Method มาปรับแก้ไขต่อด้วยวิธีรีแฟคทอริงแบบ Extract Method (กำหนดให้แทนด้วยสัญลักษณ์ R2) ที่ตำแหน่งอื่น ทำให้ภายหลังจากสิ้นสุดการปรับแก้ไขด้วยวิธีรีแฟคทอริงแบบ Extract Method ทำให้ได้ซอร์ซโค้ดที่ถูกปรับแก้ไขเป็น A[R1,R2] โดยเส้นทางในการใช้งานวิธีรีแฟคทอริงสำหรับปรับแก้ไขเมท็อด A นั้นแสดงได้ดังภาพที่ 19



ภาพที่ 19 แผนภาพต้นไม้แสดงเส้นทางในการใช้งานวิธีรีแฟคทอริงสำหรับปรับแก้ไขเมท็อด A ที่เกิดจากการรวมกันของวิธีรีแฟคทอริงมากกว่า 1 วิธี

จากภาพที่ 19 จะเห็นว่าเมท็อด A ถูกปรับแก้ไขด้วยวิธีรีแฟคทอริง 2 วิธี คือ วิธีรีแฟคทอริงแบบ Move Method ก่อน จากนั้นจึงนำซอร์ซโค้ดภายหลังจากที่ปรับแก้ไขแล้วมาปรับแก้ไขด้วยวิธีรีแฟคทอริงแบบ Extract Method ที่ตำแหน่งอื่นต่อ ทำให้เมท็อด A ถูกปรับแก้ไขด้วยวิธีรีแฟคทอริง 2 วิธีติดต่อกัน ในทางกลับกันเมท็อด A ถ้าปรับแก้ไขโดยวิธีรีแฟคทอริง

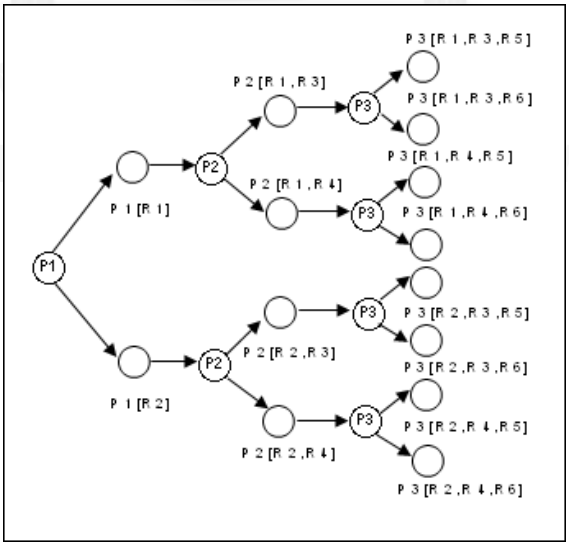
แบบ Extract Method ก่อนจากนั้นมาปรับแก้ไขด้วยวิธีรีแฟคทอริงแบบ Move Method ก็จะทำให้ได้ซอร์ซโค้ดที่ถูกปรับแก้ไขเป็น A[R2,R1] แทน ดังนั้นจำนวนเส้นทางที่เป็นไปได้ทั้งหมดสำหรับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขเมท็อด A จึงมีค่าเท่ากับ 2 เส้นทาง โดยจำนวนเส้นทางที่เป็นไปได้ทั้งหมดสำหรับการใช้งานร่วมกันระหว่างวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดนั้น สามารถค้นหาได้จากสูตรดังต่อไปนี้

จำนวนเส้นทางที่เป็นไปได้ทั้งหมดสำหรับการใช้งานร่วมกันระหว่างวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด = คอมบิเนชัน (Combination) ของจำนวนวิธีรีแฟคทอริงที่ใช้งานร่วมกัน

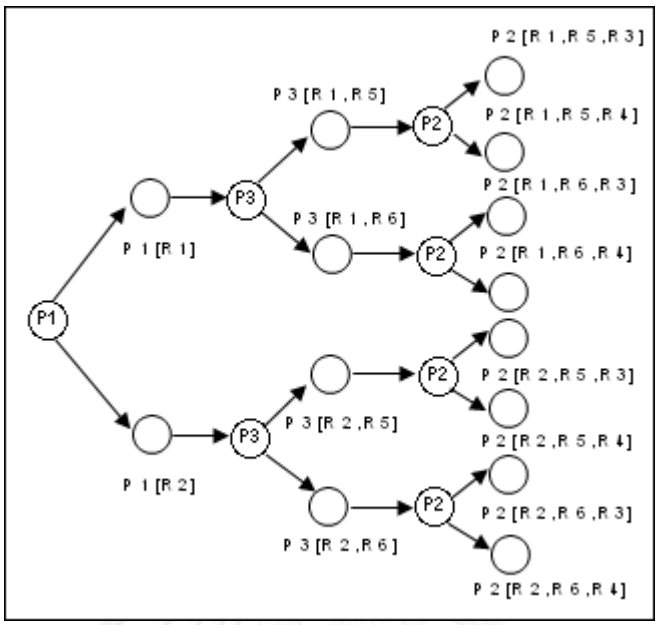
= n x (n-1) x (n-2) x ... x (n-(n-2)) x (n-(n-1))

กำหนดให้ n แทนจำนวนวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไข

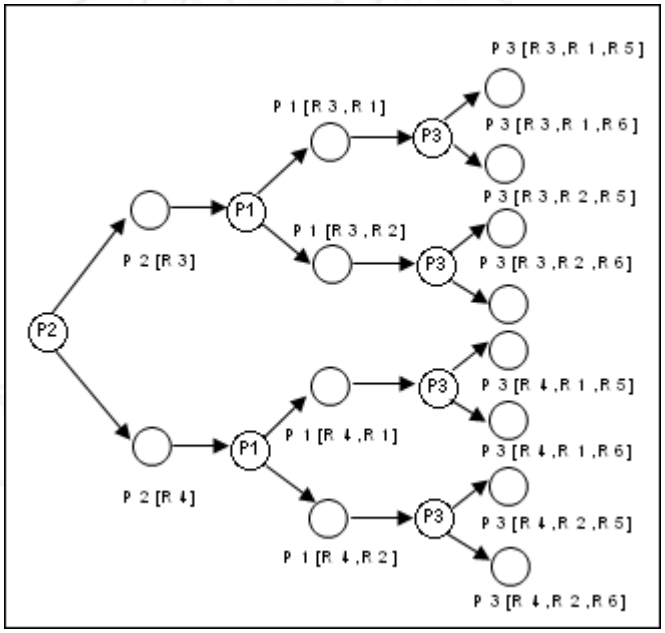
จากตัวอย่างเมท็อด Statement ที่เป็นซอร์ซโค้ดนำเข้ามีตำแหน่งที่สามารถทำการปรับแก้ไขด้วยวิธีรีแฟคทอริงทั้งหมด 3 ตำแหน่ง (จากขั้นตอนการค้นหาร่องรอยที่ผิดพลาด) โดยกำหนดให้ตำแหน่งที่ 1 แทนด้วยสัญลักษณ์ P1 ตำแหน่งที่ 2 แทนด้วยสัญลักษณ์ P2 และตำแหน่งที่ 3 แทนด้วยสัญลักษณ์ P3 แต่ละตำแหน่งสามารถเลือกปรับแก้ไขได้ด้วยวิธีรีแฟคทอริง 2 แบบ คือ ตำแหน่งที่ 1 สามารถเลือกวิธีรีแฟคทอริง R1 หรือ R2 ในการปรับแก้ไข ตำแหน่งที่ 2 สามารถเลือกวิธีรีแฟคทอริง R3 หรือ R4 ในการปรับแก้ไข และตำแหน่งที่ 3 สามารถเลือกวิธีรีแฟคทอริง R5 หรือ R6 ในการปรับแก้ไข ลำดับตำแหน่งที่เป็นไปได้ในการเลือกปรับแก้ไขนั้นมีทั้งหมด 6 แบบ คือ (P1,P2,P3), (P1,P3,P2), (P2,P1,P3), (P2,P3,P1), (P3,P1,P2) และ (P3,P2,P1) ซึ่งแต่ละแบบจะมีเส้นทางที่เป็นไปได้ทั้งหมดสำหรับการปรับแก้ไขโค้ดด้วยวิธีรีแฟคทอริงทั้งหมด 8 เส้นทาง ดังนั้นจำนวนเส้นทางที่เป็นไปได้ทั้งหมดสำหรับการปรับแก้ไขในทุกลำดับมีจำนวนทั้งหมด 48 เส้นทาง โดยตัวอย่างแผนภาพต้นไม้แสดงเส้นทางการปรับแก้ไขเมท็อด Statement นั้นได้แสดงได้ดังภาพที่ 20 – ภาพที่ 25



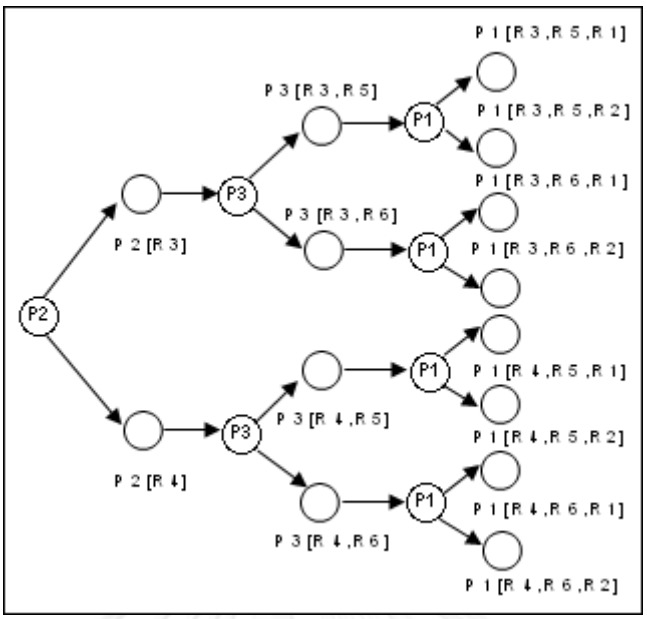
ภาพที่ 20 แผนภาพต้นไม้แสดงเส้นทางในการใช้งานวิธีรีแฟคทอริงสำหรับปรับแก้ไขเมท็อด Statement ตามลำดับแบบที่ P1,P2,P3



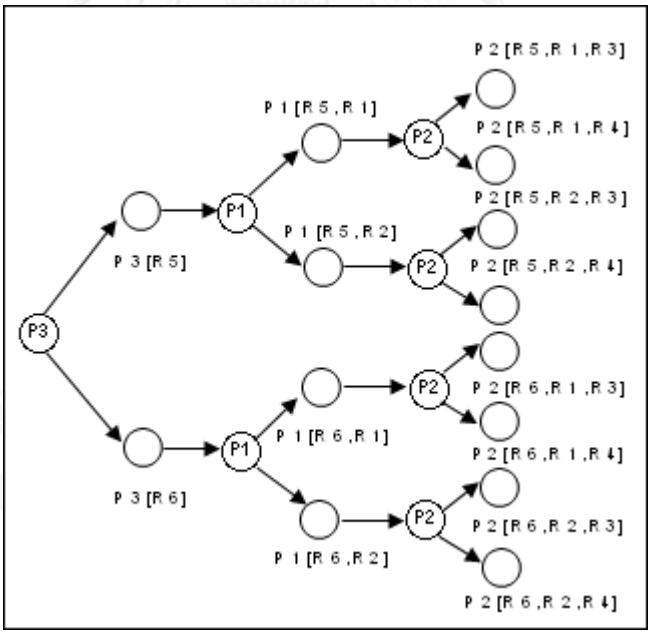
ภาพที่ 21 แผนภาพต้นไม้แสดงเส้นทางในการใช้งานวิธีรีแฟคทอริงสำหรับปรับแก้ไขเมท็อด Statement ตามลำดับแบบที่ P1,P3,P2



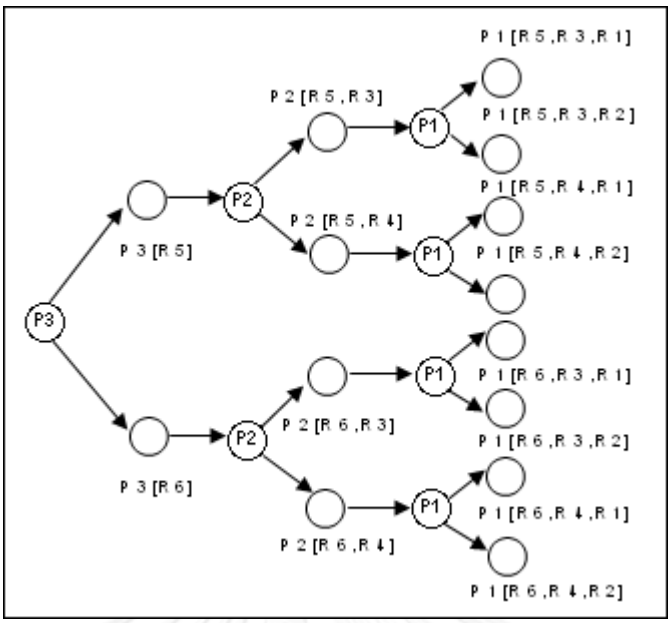
ภาพที่ 22 แผนภาพต้นไม้แสดงเส้นทางในการใช้งานวิธีรีแฟคทอริงสำหรับปรับแก้ไขเมท็อด Statement ตามลำดับแบบที่ P2,P1,P3



ภาพที่ 23 แผนภาพต้นไม้แสดงเส้นทางในการใช้งานวิธีรีแฟคทอริงสำหรับปรับแก้ไขเมท็อด Statement ตามลำดับแบบที่ P2,P3,P1



ภาพที่ 24 แผนภาพต้นไม้แสดงเส้นทางในการใช้งานวิธีรีแฟคทอริงสำหรับปรับแก้ไขเมท็อด Statement ตามลำดับแบบที่ P3,P1,P2



ภาพที่ 25 แผนภาพต้นไม้แสดงเส้นทางในการใช้งานวิธีรีแฟคทอริงสำหรับปรับแก้ไขเมท็อด Statement ตามลำดับแบบที่ P3,P2,P1

3.3 ขั้นตอนการคำนวณหาค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ของซอร์ซโค้ด

เป็นขั้นตอนในการกำหนดค่าทรัพยากรที่ใช้ในการเดินไปแต่ละเส้นทางโดยการใช้ค่าความสามารถในการบำรุงรักษาซอฟต์แวร์เป็นตัวกำหนดค่าทรัพยากร วิทยานิพนธ์นี้ได้ใช้มาตรวัดเชิงวัตถุแบบซีเคเป็นตัววัดในการหาค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ โดยมาตรวัดที่นำมาใช้เป็นเกณฑ์ในการพิจารณาค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ ได้แก่ การเข้าคู่กันระหว่างวัตถุ (Coupling Between Object Classes - CBO) ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส (Lack of Cohesion in Method - LCOM) และมาตรวัดผลรวมค่าความซับซ้อนต่อคลาส (Weighted Methods Per Class - WMC) ซึ่งมาตรวัดทั้ง 3 ตัวมีความสัมพันธ์กับค่าความสามารถในการบำรุงรักษาในลักษณะแปรผกผัน [15] กล่าวคือ ถ้ามาตรวัดทั้ง 3 ตัวมีค่าน้อย ค่าความสามารถในการบำรุงรักษาซอฟต์แวร์จะมีค่ามาก ในทางกลับกันถ้ามาตรวัดทั้ง 3 ตัวมีค่ามาก ค่าความสามารถในการบำรุงรักษาซอฟต์แวร์จะมีค่าน้อย ลักษณะของมาตรวัดแต่ละตัวที่ทำให้ค่าความสามารถมาตรในการบำรุงรักษาซอฟต์แวร์มากนั้นควรมีคุณลักษณะดังนี้คือ

1.การเข้าคู่กันระหว่างวัตถุบ่งบอกถึงการเป็นอิสระระหว่างกันของอ็อบเจ็คของคลาสต่างๆ ซึ่งกันและกัน ยิ่งมีค่าการเข้าคู่กันระหว่างวัตถุน้อยหมายถึงอ็อบเจ็คของคลาสนั้นมีความเป็นอิสระต่อกัน เมื่อมีการเปลี่ยนแปลงคลาสใดคลาสหนึ่งก็จะไม่กระทบกับอีกคลาสหนึ่ง ข้อผิดพลาดที่เกิดขึ้นจากการเปลี่ยนแปลงก็จะน้อย

2.ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสนั้น ยิ่งมีค่าระดับการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสน้อยนั้นหมายถึงเมท็อดต่างๆ ภายในคลาสนั้นมีการเรียกใช้งานตัวแปรอินสแตนซ์ที่มีความคล้ายคลึงกันอยู่ ทำให้ตัวแปรอินแสตนซ์นั้นเป็นตัวแปรของ

คลาสนั้นอย่างแท้จริง ลดจำนวนตัวแปรอินสแตนซ์ที่ไม่มีความเกี่ยวข้องกันออกไป ทำให้การอ่านทำความเข้าใจซอร์ซโค้ดนั้นทำได้ง่ายขึ้น

3.มาตรวัดผลรวมค่าความซับซ้อนต่อคลาสนั้น ยิ่งมีค่ามาตรวัดผลรวมค่าความซับซ้อนต่อคลาสน้อยจะทำให้คลาสนั้นมีจำนวนเมท็อดที่มีความซับซ้อนน้อย การอ่านทำความเข้าใจและแก้ไขโค้ดนั้นก็จะทำได้ง่ายขึ้น

ความสัมพันธ์ระหว่างค่าความสามารถในการบำรุงรักษาซอฟต์แวร์กับมาตรวัดทั้ง 3 ตัวนั้นแสดงได้ดังสมการดังนี้

$$M_t \propto (WMC)$$
$$M_t \propto (LCOM)$$
$$M_t \propto (CBO)$$

ในส่วนของการเข้าคู่กันระหว่างวัตถุนั้นจะแยกพิจารณาออกเป็น 2 มาตรวัด คือ เอฟเฟอร์เรนคัปปลิง (Efferent Coupling) และแอฟเฟอร์เรนคัปปลิง (Afferent Coupling) โดยเอฟเฟอร์เรนคัปปลิงจะเป็นการเข้าคู่กันระหว่างวัตถุในมุมมองของคลาสที่สนใจนั้นมีความสัมพันธ์ที่ไปอ้างอิงกับคลาสอื่นมากน้อยเพียงใด ในส่วนของแอฟเฟอร์เรนคัปปลิงนั้นจะเป็นการเข้าคู่กันระหว่างวัตถุในมุมมองของคลาสที่สนใจนั้นมีคลาสอื่นๆ มาอ้างอิงมากน้อยเพียงใด รายละเอียดของการคำนวณค่ามาตรวัดแต่ละตัวที่ใช้พิจารณาค่าความสามารถในการบำรุงรักษาซอฟต์แวร์นั้นแสดงได้ดังตารางที่ 6

ตารางที่ 6 แสดงรายละเอียดของการคำนวณค่ามาตรวัดที่ใช้พิจารณาค่าความสามารถในการบำรุงรักษาซอฟต์แวร์

มาตรวัด		คำอธิบาย
การเข้าคู่กันระหว่างวัตถุ	เอฟเฟอร์เรนคัปปลิง	จำนวนคลาสอื่นที่คลาสหลักต้องอ้างอิง
	แอฟเฟอร์เรนคัปปลิง	จำนวนคลาสอื่นที่มาอ้างอิงคลาสหลัก
ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส		จำนวนของตัวแปรอินสแตนซ์ที่ไม่ได้ใช้งานร่วมกันที่ถูกเรียกใช้งานภายในเมท็อดของคลาสเดียวกัน
ผลรวมค่าความซับซ้อนต่อคลาส		ผลรวมของค่าความซับซ้อนของแต่ละเมท็อดภายในคลาส ค่าความซับซ้อนภายในเมท็อดนั้นคำนวณได้จากสูตร $M = E - N - 2P$ โดยที่ E คือ จำนวนด้าน (เส้น) ของกราฟ N คือ จำนวนโหนดของกราฟ P คือ จำนวนคอมโพเนนต์ในกราฟ

จากแผนภาพต้นไม้แสดงการใช้งานวิธีรีแฟคทอริงสำหรับปรับแก้ไขเมท็อด Statement ที่ได้จากขั้นตอนการประยุกต์ใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดนั้น ในส่วนของการปรับแก้ไขในลำดับแบบที่ P1,P2,P3 เมื่อนำมาคำนวณหามาตรวัดแต่ละตัวเพื่อระบุเป็นค่าความสามารถในการบำรุงรักษาซอฟต์แวร์สำหรับแต่ละเส้นทางนั้น แต่ละเส้นทางมีค่าของมาตรวัดแสดงดังตารางที่ 7 จากตารางที่ 7 จะเห็นว่าการปรับแก้ไขโค้ดด้วยวิธีรีแฟคทอริงแต่ละเส้นทางได้ค่ามาตรวัดที่แตกต่างกัน ในขั้นตอนถัดไปจะใช้อัลกอริทึมแบบละโมบในการเลือกเส้นทางในการปรับแก้ไขโค้ดด้วยวิธีรีแฟคทอริงเพื่อให้ได้ซอร์ซโค้ดภายหลังจากการปรับแก้ไขมีค่าความสามารถในการบำรุงรักษามากที่สุด

ตารางที่ 7 แสดงรายละเอียดค่าของมาตรวัดสำหรับเส้นทาง
การปรับแก้ไขเมท็อด Statement ในลำดับแบบที่ P1,P2,P3

โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
P1[R1]	13	0.667	0	1
P1[R2]	7	0.667	0	1
P2[R1,R3]	15	0.625	0	1
P2[R1,R4]	13	0.625	0	1
P2[R2,R3]	9	0.625	0	1
P2[R2,R4]	7	0.625	0	1
P3[R1,R3,R5]	22	0.6	0	1
P3[R1,R3,R6]	17	0.6	0	1
P3[R1,R4,R5]	20	0.6	0	1
P3[R1,R4,R6]	15	0.6	0	1
P3[R2,R3,R5]	16	0.6	0	1
P3[R2,R3,R6]	11	0.6	0	1
P3[R2,R4,R5]	14	0.6	0	1
P3[R2,R4,R6]	9	0.6	0	1

3.4 ขั้นตอนการเลือกลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด

เป็นขั้นตอนในการเลือกเส้นทางในการปรับแก้ไขโค้ดด้วยวิธีรีแฟคทอริง โดยการใช้อัลกอริทึมละโมบในการเลือกเส้นทาง เพื่อลดจำนวนเส้นทางที่เป็นไปได้ลง (search space) และใช้เวลาในการ

หาผลลัพธ์น้อย ซึ่งหลักการของอัลกอริทึมละโมบนั้นจะเลือกเส้นทางที่ให้ผลลัพธ์ที่ดีที่สุดในแต่ละรอบของการเดินทาง เลือกเช่นนี้ไปเรื่อยๆ จนกระทั่งสิ้นสุดเส้นทาง จะได้ผลลัพธ์ในการแก้ไขปัญหาที่ดีที่สุดจากการเลือกเส้นทางที่ดีที่สุดในแต่ละรอบ จากขั้นตอนการประยุกต์ใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดจะเห็นว่าการปรับแก้ไขเมท็อด Statement นั้นสามารถเลือกปรับแก้ไขได้ทั้งหมด 48 แบบ ถ้าต้องการหาเส้นทางที่ดีที่สุดหรือปรับแก้ไขโค้ดให้ได้ซอร์ซโค้ดที่มีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ที่มากที่สุดนั้น จะต้องทำการปรับแก้ไขโค้ดให้ครบทั้งหมด 48 แบบที่เป็นไปได้ทั้งหมดในการปรับแก้ไข จากนั้นจึงเลือกซอร์ซโค้ดที่ให้ค่าบำรุงรักษาที่มากที่สุด การทำแบบนี้จะทำให้เสียเวลามากในการแก้ปัญหา วิทยานิพนธ์นี้จึงใช้อัลกอริทึมละโมบในการเลือกเส้นทางในการปรับแก้ไขโค้ดโดยวิธีรีแฟคทอริง ในเกณฑ์ในการเลือกลำดับการใช้งานวิธีรีแฟคทอริงนั้นจะพิจารณาวิธีรีแฟคทอริงที่ภายหลังจากการปรับแก้ไขแล้วนั้นให้ค่าความสามารถในการบำรุงรักษาที่มากที่สุดเป็นหลัก ซึ่งจะไม่พิจารณาลำดับของการปรับแก้ไขของลักษณะของร่องรอยที่ผิดพลาด กล่าวคือ ไม่จำเป็นต้องแก้ไขร่องรอยที่ผิดพลาดชนิดใดชนิดหนึ่งให้เสร็จก่อนถึงจะปรับแก้ไขร่องรอยที่ผิดพลาดชนิดอื่นได้ เพียงแต่สนใจเลือกวิธีรีแฟคทอริงที่ให้ผลลัพธ์ของค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ที่สูงสุดเท่านั้น โดยขั้นตอนในการเลือกเส้นทางในการปรับแก้ไขโค้ดนั้นจะเป็นลักษณะของการวนรอบการทำงาน ซึ่งมีขั้นตอนดังนี้

1. ประยุกต์ใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดในแต่ละตำแหน่ง ในการปรับแก้ไขโค้ดนั้นจะแก้ไขเพียง 1 ตำแหน่งต่อ 1 รอบของการปรับแก้ไข ตัวอย่างเช่นเมท็อด Statement ของคลาส Customer สามารถปรับแก้ไขด้วยวิธีรีแฟคทอริงได้ทั้งหมด 6 แบบ ในรอบแรกของการปรับแก้ไขโค้ดนั้น จะได้ซอร์ซโค้ด 6 แบบที่แตกต่างกันตามวิธีรีแฟคทอริงที่ปรับแก้ไขทั้งหมด 6 แบบ

2. คำนวณหาค่าความสามารถในการบำรุงรักษาซอฟต์แวร์สำหรับซอร์ซโค้ดที่ปรับแก้ไขด้วยวิธีรีแฟคทอริงในแต่ละแบบ เพื่อกำหนดให้เป็นค่าทรัพยากรที่ใช้ในการเดินไปยังแต่ละเส้นทาง

3. เลือกซอร์ซโค้ดหลังภายจากปรับแก้ไขด้วยวิธีรีแฟคทอริงที่มีค่าความสามารถในการบำรุงรักษาที่มากที่สุด (มาตรวัดทั้ง 3 ตัวมีค่าน้อยที่สุด โดยการเปรียบเทียบทีละตัว) เพื่อนำมาเป็นซอร์ซโค้ดที่จะนำมาปรับแก้ไขในตำแหน่งที่เหลือในรอบถัดไป

4. นำซอร์ซโค้ดที่ได้จากข้อ 3 มาประยุกต์ใช้งานวิธีรีแฟคทอริงในตำแหน่งที่เหลือ

5. ทำขั้นตอนที่ 2-4 ซ้ำไปเรื่อยๆ จนกระทั่งแก้ไขโค้ดการทำงานจนครบทุกตำแหน่ง จึงสิ้นสุดการวนรอบการเลือกเส้นทางในการปรับแก้ไขโค้ด

โดยจำนวนโหนดที่ต้องพิจารณาทั้งหมดในการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบหาได้จากสูตรดังนี้

จำนวนโหนดที่ต้องค้นหาทั้งหมด = ผลรวมของจำนวนวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขในแต่ละรอบ ($\sum_{r=1}^{z} \mathrm{Re}_r$)

กำหนดให้ :

- Re_r เป็นผลรวมของจำนวนวิธีรีแฟคทอริงที่ใช้ในการแก้ไขของแต่ละตำแหน่ง ($\sum_{i=1}^{z} Pi$) ของรอบการแก้ไขที่ r

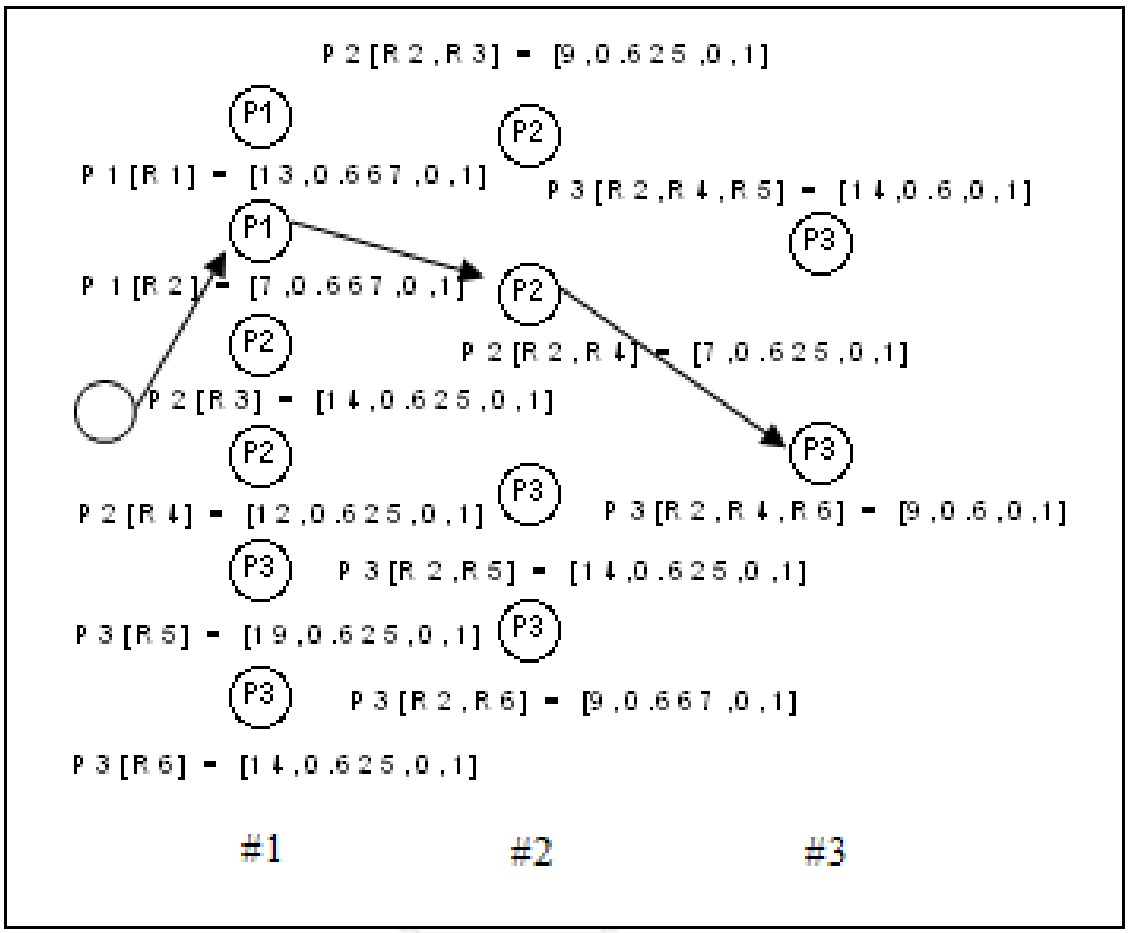
- Pi เป็นจำนวนวิธีรีแฟคทอริงที่ใช้ในการแก้ไขของตำแหน่งที่ i

- z เป็นจำนวนตำแหน่งที่จะปรับแก้ไข

- i เป็นตำแหน่งที่ปรับแก้ไข

- เมื่อสิ้นสุดการแก้ไขในตำแหน่งหนึ่งๆ จำนวนของตำแหน่งที่จะแก้ไขจะลดลง 1 ตำแหน่ง

ภายหลังจากสิ้นสุดการวนรอบการเลือกเส้นทางในการปรับแก้ไขโค้ดแล้ว จะได้ลำดับของการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดที่ภายหลังจากการปรับแก้ไขแล้วจะได้ซอร์ซโค้ดที่มีค่าความสามารถในการบำรุงรักษามากที่สุด แผนภาพต้นไม้แสดงการเลือกเส้นทางในการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบในการปรับแก้ไขโค้ดเมท็อด Statement แสดงได้ดังภาพที่ 26 การเลือกเส้นทางการปรับแก้ไขโค้ดนั้นเริ่มจากนำซอร์ซโค้ดเมท็อด Statement ที่เป็นซอร์ซโค้ดนำเข้าที่จะนำมาปรับแก้ไขด้วยวิธีรีแฟคทอริง โดยจะแทนด้วยสัญลักษณ์โหนดว่างเปล่า (Blank node) มาทำการปรับแก้ไขด้วยวิธีรีแฟคทอริงในแต่ละตำแหน่งและแต่ละวิธีที่เป็นไปได้ โดยเมท็อด Statement ของคลาส Customer นั้นมีส่วนที่ต้องปรับแก้ไข 3 ตำแหน่ง โดยที่แต่ละตำแหน่งนั้นสามารถปรับแก้ไขได้ด้วยวิธีรีแฟคทอริงได้ 2 วิธี คือ ตำแหน่งที่ 1 (P1) สามารถปรับแก้ไขได้ด้วยวิธีรีแฟคทอริงแบบ Extract Method (R1) หรือ Move Method (R2) ตำแหน่งที่ 2 (P2) สามารถปรับแก้ไขได้ด้วยวิธีรีแฟคทอริงแบบ Extract Method (R3) หรือ Move Method (R4) และตำแหน่งที่ 3 (P3) สามารถปรับแก้ไขได้ด้วยวิธีรีแฟคทอริงแบบ Extract Method (R5) หรือ Move Method (R6) ในรอบที่ 1 ของการปรับแก้ไขโค้ดจะได้ซอร์ซโค้ดทั้งหมด 6 แบบ โดยค่ามาตรวัดแต่ละตัวของซอร์ซโค้ดแต่ละแบบนั้นจะแสดงอยู่ภายในเครื่องหมายกล่องข้อความเรียงจากซ้ายไปขวา คือ ผลรวมค่าความซับซ้อนต่อคลาส (WMC) ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส (LCOM) แอฟเฟอร์เรนคัปปลิง (AC) และเอฟเฟอร์เรนคัปปลิง (EC) ตามลำดับ โดยค่ามาตรวัดของเมท็อด Statement ของคลาส Customer ก่อนปรับแก้ไขนั้นมีรายละเอียดดังนี้ ค่าผลรวมความซับซ้อนต่อคลาสมีค่าเท่ากับ 12 หน่วย, ค่าระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสมีค่าเท่ากับ 0.667 หน่วย, ค่าเอฟเฟอร์เรนคัปปลิงมีค่าเท่ากับ 0 หน่วยและค่าเอฟเฟอร์เรนคัปปลิงเท่ากับ 1 แสดงได้ดังตารางที่ 8



ภาพที่ 26 แผนภาพต้นไม้แสดงเส้นทางการเลือกใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด โดยใช้อัลกอริทึมละโมบ

ตารางที่ 8 แสดงค่ามาตรวัดของเมท็อด Statement ของคลาส Customer ก่อนการปรับแก้ไขโค้ดด้วยวิธีรีแฟคทอริง

โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
วางเปล่า (Blank)	12	0.667	0	1

จากตารางที่ 9 รอบที่ 1 ของการปรับแก้ไขโค้ดเมท็อด Statement นั้น การเลือกปรับแก้ไขด้วยวิธีรีแฟคทอริงแบบ Extract Method แยกซอร์ซโค้ดส่วนของการคำนวณค่าเช่าภาพยนตร์ไปเป็นเมท็อดใหม่ภายใต้คลาส Customer (วิธีรีแฟคทอริงแบบ R2) หรือการเลือกปรับแก้ไขด้วยวิธีรีแฟคทอริงแบบ Move Method ย้ายซอร์ซโค้ดส่วนของการคำนวณแต้มความถี่สะสมไปเป็นเมท็อด

ใหม่ภายใต้คลาส Rental นั้น จะได้ซอร์ซโค้ดภายหลังจากการปรับแก้ไขที่มีค่ามาตรวัดทั้ง 4 ตัวที่ใกล้เคียงกัน คือ ค่าแอฟเฟอร์เรนคัปปลิงและเอฟเฟอร์เรนคัปปลิงของซอร์ซโค้ดที่ปรับแก้ไขด้วยวิธีรีแฟคทอริงทั้ง 2 แบบมีค่าเท่ากัน ส่วนมาตรวัดผลรวมค่าความซับซ้อนต่อคลาสนั้นวิธีรีแฟคทอริงแบบ R2 จะมีค่าน้อยกว่าวิธีรีแฟคทอริงแบบ R4 แต่ค่าระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสนั้นวิธีรีแฟคทอริงแบบ R4 จะมีค่าน้อยกว่าวิธีรีแฟคทอริงแบบ R2 ซึ่งตรงส่วนนี้ทางวิทยานิพนธ์ได้นำเอาส่วนต่างของค่ามาตรวัดแต่ละชนิดของทั้ง 2 วิธีมาเปรียบเทียบกัน ซึ่งส่วนต่างของค่าผลรวมค่าความซับซ้อนต่อคลาสนั้นมีค่าต่างกัน 5 หน่วยหรือคิดเป็น 41.67% จาก 12 หน่วย ในส่วนของค่าระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสนั้นมีค่าต่างกัน 0.042 หน่วยหรือคิดเป็น 6.30% จาก 0.667 หน่วย ส่วนต่างของค่าผลรวมค่าความซับซ้อนต่อคลาสมีมากกว่าส่วนต่างของค่าระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส ดังนั้นในรอบแรกของการปรับแก้ไขโค้ดจึงเลือกการปรับแก้ไขด้วยวิธี Extract Method แยกซอร์ซโค้ดส่วนของการคำนวณค่าเช่าภาพยนตร์ไปเป็นเมท็อดใหม่ภายใต้คลาส Customer ซอร์ซโค้ดภายหลังจากการปรับแก้ไขแสดงได้ดังภาพที่ 27

ตารางที่ 9 แสดงค่ามาตรวัดของเมท็อด Statement ของคลาส Customer สำหรับการปรับแก้ไขโค้ดรอบที่ 1

โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
P1[R1]	13	0.667	0	1
P1[R2]	7	0.667	0	1
P2[R3]	14	0.625	0	1
P2[R4]	12	0.625	0	1
P3[R5]	19	0.625	0	1
P3[R6]	14	0.625	0	1

```
01 public class Customer {
02        public String statement(){
03                double totalAmount = 0;
04                int frequentRenterPoints = 0;
05                Enumeration rentals = _rentals.elements();
06                String result = "Rental Record for " + getName() + "\n";
07                while (rentals.hasMoreElements()) {
08                        double thisAmount = 0;
09                        Rental each = (Rental) rentals.nextElement();
10                        thisAmount = each.getCharge();
11
12                        frequentRenterPoints ++;
13
14                        if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) && each.getDaysRented() > 1) frequentRenterPoints ++;
15
16                        result += "\t" + each.getMovie().getTitle()+ "\t" + String.valueOf(thisAmount) + "\n";
17                        totalAmount += thisAmount;
18                }
19                //add footer lines
20                result += "Amount owed is " + String.valueOf(totalAmount) + "\n";
21                result += "You earned " + String.valueOf(frequentRenterPoints) + " frequent renter points";
22                return result;
23
24        }
25}
26
27 public class Rental {
28        public double getCharge() {
29                double result = 0;
30                switch (getMovie().getPriceCode()) {
31                case Movie.REGULAR:
32                        result += 2;
33                        if (getDaysRented() > 2)
34                                result += (getDaysRented() - 2) * 1.5;
35                        break;
36                case Movie.NEW_RELEASE:
37                        result += getDaysRented() * 3;
38                        break;
39                case Movie.CHILDRENS:
40                        result += 1.5;
41                        if (getDaysRented() > 3)
42                                result += (getDaysRented() - 3) * 1.5;
43                        break;
44                }
45                return result;
46        }
47}
```

ภาพที่ 27 ซอร์ซโค้ดเมท็อด Statement ที่ปรับแก้ไขในตำแหน่งที่ 1 ด้วยวิธีรีแฟคทอริงแบบ R2

จากตารางที่ 10 รอบที่ 2 ของการปรับแก้ไขโค้ดเมท็อด Statement นั้นจะนำซอร์ซโค้ดที่ปรับแก้ไขจากรอบที่ 1 มาทำการปรับแก้ไขโค้ดในตำแหน่งที่เหลือต่อคือตำแหน่งที่ 2 หรือส่วนของการคำนวณค่าแต้มความถี่สะสมในการเช่าภาพยนตร์ของลูกค้า และตำแหน่งที่ 3 หรือส่วนของการคำนวณค่าเช่าภาพยนตร์รวมในการเช่าภาพยนตร์ครั้งหนึ่งๆ ของลูกค้า จากตารางที่ 10 การปรับแก้ไขโค้ดด้วยวิธีรีแฟคทอริงแบบ Move Method ย้ายซอร์ซโค้ดส่วนของการคำนวณแต้มความถี่สะสมไปเป็นเมท็อดใหม่ภายใต้คลาส Rental (วิธีรีแฟคทอริงแบบ R4) นั้นจะได้ค่ามาตรวัดทั้ง 3 ตัวน้อยที่สุด คือ ค่ามาตรวัดผลรวมค่าความซับซ้อนของคลาสมีค่าเท่ากับ 7 หน่วย ค่าระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสมีค่าเท่ากับ 0.667 หน่วย ค่าแอฟเฟอร์เรนคัปปลิงเท่ากับ 0 หน่วย และค่าเอฟเฟอร์เรนคัปปลิงเท่ากับ 1 หน่วย ดังนั้นจึงเลือกการปรับแก้ไขโค้ดด้วยวิธีรีแฟคทอริงแบบ Move Method ย้ายซอร์ซโค้ดส่วนของการคำนวณแต้มความถี่สะสมไปเป็นเมท็อดใหม่ภายใต้คลาส Rental ปรับแก้ไขในรอบที่ 2 ซอร์ซโค้ดภายหลังจากปรับแก้ไขแสดงได้ดังภาพที่ 28

ตารางที่ 10 แสดงค่ามาตรวัดของเมท็อด Statement ของคลาส Customer สำหรับการปรับแก้ไขโค้ดรอบที่ 2

โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
P2[R2,R3]	13	0.667	0	1
P2[R2,R4]	7	0.667	0	1
P2[R2,R5]	14	0.625	0	1
P2[R2,R6]	12	0.625	0	1

```
01 public class Customer {
02      public String statement(){
03              double totalAmount = 0;
04              int frequentRenterPoints = 0;
05              Enumeration rentals = _rentals.elements();
06              String result = "Rental Record for " + getName() + "\n";
07              while (rentals.hasMoreElements()) {
08                      double thisAmount = 0;
09                      Rental each = (Rental) rentals.nextElement();
10                      thisAmount = each.getCharge();
11
12                      result += "\t" + each.getMovie().getTitle()+ "\t" + String.valueOf(thisAmount) + "\n";
13                      totalAmount += thisAmount;
14              }
15
16              result += "Amount owed is " + String.valueOf(totalAmount) + "\n";
17              result += "You earned " + String.valueOf(getTotalFrequentRenterPoints()) + " frequent renter points";
18              return result;
19
20      }
21
22      private int getTotalFrequentRenterPoints(){
23              int result = 0;
24              Enumeration rentals = _rentals.elements();
25              while (rentals.hasMoreElements()) {
26                      Rental each = (Rental) rentals.nextElement();
27                      result += each.getFrequentRenterPoints();
28              }
29              return result;
30      }
31}
32 public class Rental {
33      public double getCharge() {
34              double result = 0;
35              switch (getMovie().getPriceCode()) {
36              case Movie.REGULAR:
37                      result += 2;
38                      if (getDaysRented() > 2)
39                              result += (getDaysRented() - 2) * 1.5;
40                      break;
41              case Movie.NEW_RELEASE:
42                      result += getDaysRented() * 3;
43                      break;
44              case Movie.CHILDRENS:
45                      result += 1.5;
46                      if (getDaysRented() > 3)
47                              result += (getDaysRented() - 3) * 1.5;
48                      break;
49              }
50              return result;
51      }
52
53      public int getFrequentRenterPoints() {
54              if ((getMovie().getPriceCode() == Movie.NEW_RELEASE) &&
55                              getDaysRented() > 1)
56                      return 2;
57              else
58                      return 1;
59      }
60
61}
```

ภาพที่ 28 ซอร์ซโค้ดเมท็อด Statement ที่ปรับแก้ไขในตำแหน่งที่ 1 และ 2 ด้วยวิธีรีแฟคทอริงแบบ R2 และ R4 ตามลำดับ

จากตารางที่ 11 รอบที่ 3 ของการปรับแก้ไขโค้ดเมท็อด Statement นั้นนำซอร์ซโค้ดที่ปรับแก้ไขจากรอบที่ 2 มาทำการปรับแก้ไขโค้ดในตำแหน่งสุดท้าย คือ ตำแหน่งที่ 3 ส่วนของการคำนวณค่าเช่าภาพยนตร์รวมในการเช่าภาพยนตร์ครั้งหนึ่งๆ ของลูกค้า ซึ่งในตำแหน่งนี้สามารถปรับแก้ไขด้วยวิธีรีแฟคทอริงได้ 2 วิธี คือ วิธีรีแฟคทอริงแบบ Extract Method แยกซอร์ซโค้ดส่วนนี้ไปเป็นเมท็อดใหม่ภายใต้คลาส Customer (วิธีรีแฟคทอริงแบบ R5) และวิธีรีแฟคทอริงแบบ Move Method ย้ายซอร์ซโค้ดส่วนของการคำนวณแต้มความถี่สะสมไปเป็นเมท็อดใหม่ภายใต้คลาส Rental (วิธีรีแฟคทอริงแบบ 6) การปรับแก้ไขโค้ดด้วยวิธีรีแฟคทอริงแบบ Move Method ย้ายซอร์ซโค้ดส่วนของการคำนวณค่าเช่าภาพยนตร์รวมในการเช่าภาพยนตร์ไปเป็นเมท็อดใหม่ภายใต้คลาส Rental (วิธีรีแฟคทอริงแบบ R6) นั้นจะได้ค่ามาตรวัดทั้ง 3 ตัวน้อยที่สุด คือ ค่ามาตรวัดผลรวมค่าความซับซ้อนของคลาสมีค่าเท่ากับ 9 หน่วย ค่าระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสมีค่าเท่ากับ 0.6 หน่วย ค่าแอฟเฟอร์เรนคัปปลิงเท่ากับ 0 หน่วย และค่าเอฟเฟอร์เรนคัปปลิงเท่ากับ 1 หน่วย ดังนั้นจึงเลือกการปรับแก้ไขโค้ดด้วยวิธีรีแฟคทอริงแบบ Move Method ย้ายซอร์ซโค้ดส่วนของการคำนวณค่าเช่าภาพยนตร์รวมในการเช่าภาพยนตร์เป็นเมท็อดใหม่ภายใต้คลาส Rental ปรับแก้ไขในรอบที่ 3 ซอร์ซโค้ดภายหลังจากปรับแก้ไขแสดงได้ดังภาพที่ 29

ตารางที่ 11 แสดงค่ามาตรวัดของเมท็อด Statement ของคลาส Customer สำหรับการปรับแก้ไขโค้ดรอบที่ 3

โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
P3[R2,R4,R5]	14	0.6	0	1
P3[R2,R4,R6]	9	0.6	0	1

ภายหลังจากปรับแก้ไขตำแหน่งที่ 3 เสร็จสิ้นแล้วจะไม่เหลือตำแหน่งที่ต้องปรับแก้ไขในเมท็อด Statement อีก หรือได้ขจัดร่องรอยที่ผิดพลาดภายในเมท็อดจนครบแล้ว จึงหยุดขั้นตอนการค้นหาลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด เมื่อสิ้นสุดขั้นตอนนี้จะได้ลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดเมท็อด Statement คือ ใช้วิธีรีแฟคทอริงแบบ Move Method ในการย้ายซอร์ซโค้ดส่วนของการคำนวณค่าเช่าภาพยนตร์ไปเป็นเมท็อดใหม่ภายใต้คลาส Rental (วิธีรีแฟคทอริงแบบ R2) จากนั้นวิธีรีแฟคทอริงแบบ Move Method ในการย้ายซอร์ซโค้ดส่วนของการคำนวณแต้มความถี่สะสมไปเป็นเมท็อดใหม่ภายใต้คลาส Rental (วิธีรีแฟคทอริงแบบ R4) และใช้วิธีรีแฟคทอริงแบบ Move Method ในการย้ายซอร์ซโค้ดส่วนของการคำนวณค่าเช่าภาพยนตร์รวมไปเป็นเมท็อดใหม่ภายใต้คลาส Rental (วิธีรีแฟคทอริงแบบ R6) ซึ่งซอร์ซโค้ดที่ได้จากการปรับแก้ไขด้วยวิธีรีแฟคทอริงตามลำดับดังกล่าวจะมีค่ามาตรวัดต่างๆ แสดงดังตารางที่ 12 ซึ่งมีรายละเอียดดังนี้ ค่าผลรวมความซับซ้อนต่อคลาสมีค่าเท่ากับ 9 หน่วย ค่าระดับของการขาดการเกาะกันเป็นก้อนของ

เมท็อดภายในคลาสมีค่าเท่ากับ 0.6 หน่วย ค่าแอฟเฟอร์เรนคัปปลิงมีค่าเท่ากับ 0 หน่วย และค่าเอฟเฟอร์เรนคัปปลิงมีค่าเท่ากับ 1 หน่วย ในส่วนของซอร์ซโค้ดของเมท็อด Statement ภายหลังจากการปรับแก้ไขตามลำดับการใช้งานวิธีรีแฟคทอริงนั้นแสดงได้ดังภาพที่ 29 จะเห็นได้ว่าการใช้งานอัลกอริทึมละโมบมาช่วยจัดลำดับการใช้งานวิธีรีแฟคทอริงนั้นจะทำการสำรวจโหนดหรือซอร์ซโค้ดเพียงแค่ 12 แบบเท่านั้น จึงช่วยลดเวลาในการปรับแก้ไขซอร์ซโค้ด

ตารางที่ 12 แสดงค่ามาตรวัดของเมท็อด Statement ของคลาส Customer ภายหลังการปรับแก้ไขตามลำดับการใช้งานวิธีรีแฟคทอริง

โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
P3[R2,R4,R6]	9	0.6	0	1

3.5 ขั้นตอนการประเมินผลลัพธ์ของลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด

เป็นขั้นตอนการประเมินผลลัพธ์ของลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดด้วยอัลกอริทึมละโมบ โดยการประเมินนั้นจะพิจารณาคุณลักษณะ 3 ด้าน คือ ความสามารถในการบำรุงรักษาซอฟต์แวร์ของซอร์ซโค้ดของที่ถูกปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง เวลาที่ใช้ในการค้นหาลำดับการใช้งานวิธีรีแฟคทอริงของวิธีละโมบ และความถูกต้องของซอร์ซโค้ดภายหลังจากที่ปรับแก้ไขด้วยวิธีรีแฟคทอริง เพื่อเป็นการยืนยันว่าการพิจารณาลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดนั้น จะทำให้ได้ซอร์ซโค้ดที่มีความสามารถในการบำรุงรักษาซอฟต์แวร์มากกว่าซอร์ซโค้ดที่ไม่ได้พิจารณาลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด และการใช้อัลกอริทึมละโมบในการค้นหาลำดับการใช้วิธีรีแฟคทอริงจะใช้เวลาและทรัพยากรน้อยกว่าอัลกอริทึมแบบอื่นๆ ซึ่งขั้นตอนการของประเมินคุณลักษณะดังกล่าวจะอธิบายในบทที่ 4 ต่อไป

```
01 public class Customer {
02        public String statement(){
03                Enumeration rentals = _rentals.elements();
04                String result = "Rental Record for " + getName() + "\n";
05                while (rentals.hasMoreElements()) {
06                        double thisAmount = 0;
07                        Rental each = (Rental) rentals.nextElement();
08                        thisAmount = each.getCharge();
09
10                        result += "\t" + each.getMovie().getTitle()+ "\t" + String.valueOf(thisAmount) + "\n";
11                }
12
13                result += "Amount owed is " + String.valueOf(getTotalCharge()) + "\n";
14                result += "You earned " + String.valueOf(getTotalFrequentRenterPoints()) + " frequent renter points";
15                return result;
16        }
17
18        private double getTotalCharge() {
19                double result = 0;
20                Enumeration rentals = _rentals.elements();
21                while (rentals.hasMoreElements()) {
22                        Rental each = (Rental) rentals.nextElement();
23                        result += each.getCharge();
24                }
25                return result;
26        }
27
28        private int getTotalFrequentRenterPoints(){
29                int result = 0;
30                Enumeration rentals = _rentals.elements();
31                while (rentals.hasMoreElements()) {
32                        Rental each = (Rental) rentals.nextElement();
33                        result += each.getFrequentRenterPoints();
34                }
35                return result;
36        }
37
38 }
39
40 public class Rental {
41        public double getCharge() {
42                double result = 0;
43                switch (getMovie().getPriceCode()) {
44                case Movie.REGULAR:
45                        result += 2;
46                        if (getDaysRented() > 2)
47                                result += (getDaysRented() - 2) * 1.5;
48                        break;
49                case Movie.NEW_RELEASE:
50                        result += getDaysRented() * 3;
51                        break;
52                case Movie.CHILDRENS:
53                        result += 1.5;
54                        if (getDaysRented() > 3)
55                                result += (getDaysRented() - 3) * 1.5;
56                        break;
57                }
58                return result;
59        }
60
61        public int getFrequentRenterPoints() {
62                if ((getMovie().getPriceCode() == Movie.NEW_RELEASE) &&
63                                getDaysRented() > 1)
64                        return 2;
65                else
66                        return 1;
67        }
68 }
```

ภาพที่ 29 ซอร์ซโค้ดเมท็อด Statement ภายหลังการปรับแก้ไขตามลำดับการใช้งาน

วิธีรีแฟคทอริง

บทที่ 4
การประเมินผลลัพธ์ของลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด

เป็นการประเมินผลลัพธ์ที่ได้จากการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด และประเมินอัลกอรึมละโมบที่ใช้ในการค้นหาลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด โดยแยกพิจารณาคุณลักษณะ 3 ด้าน คือ

1. ความสามารถในการบำรุงรักษาซอฟต์แวร์ของซอร์ซโค้ดภายหลังจากการปรับแก้ไข

2. เวลาที่ใช้ในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดด้วยอัลกอริทึมละโมบ

3. ความถูกต้องของซอร์ซโค้ดภายหลังจากการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ

โดยการประเมินผลลัพธ์นั้นจะทำการทดสอบการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดกับซอร์ซโค้ดจากโอเพนซอร์ซ (Open Source) 2 โปรแกรม คือ ระบบเช่าภาพยนตร์ (Movie Rental) ตัวอย่างซอร์ซโค้ดจากบทที่ 3 และระบบคำนวณค่าไฟฟ้า (Electric Payment) [1] โดยรายละเอียดของซอร์ซโค้ดก่อนและหลังการปรับแก้ไขด้วยการจัดลำดับวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบของระบบคำนวณค่าไฟฟ้านั้นจะอธิบายที่ภาคผนวก ก ซึ่งลักษณะความแตกต่างของซอร์ซโค้ดโปรแกรมทั้งสองแสดงได้ดังตารางที่ 13 ซึ่งการประเมินผลในแต่ละด้านนั้นมีรายละเอียดดังต่อไปนี้

ตารางที่ 13 ลักษณะความแตกต่างระหว่างซอร์ซโค้ดนำเข้า 2 โปรแกรม

ระบบเช่าภาพยนตร์	ระบบคำนวณค่าไฟฟ้า
1. มีร่องรอยที่ผิดพลาดทั้ง 3 ชนิดอยู่ภายในคลาสเดียวกัน คือ ร่องรอยที่ผิดพลาดแบบเมท็อดที่มีความยาวมาก ร่องรอยที่ผิดพลาดแบบคลาสที่มีขนาดใหญ่ และร่องรอยที่ผิดพลาดแบบฟีเจอร์เอนวี	1. มีร่องรอยที่ผิดพลาดชนิดเดียว คือ ร่องรอยที่ผิดพลาดแบบเมท็อดที่มีความยาวมาก โดยพบร่องรอยที่ผิดพลาดชนิดนี้มากกว่า 1 คลาส
2. แต่ละตำแหน่งที่ต้องปรับแก้ไขนั้น มีวิธีรีแฟคทอริงที่สามารถใช้ปรับแก้ไขได้มากกว่า 1 วิธีให้เลือกใช้งาน	2. แต่ละตำแหน่งที่ต้องปรับแก้ไข มีวิธีรีแฟคทอริงเพียงวิธีเดียวที่สามารถใช้ปรับแก้ไขได้
3. วิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขนั้น มีวิธีที่มีคุณสมบัติในการย้ายการทำงานของซอร์ซโค้ดไปไว้ที่คลาสอื่น	3. วิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขนั้น มีเพียงแค่ปรับแก้ไขภายในคลาสของตัวเองเท่านั้น

4.1 ความสามารถในการบำรุงรักษาซอฟต์แวร์ของซอร์ซโค้ดภายหลังจากการปรับแก้ไข

ในการประเมินด้านความสามารถในการบำรุงรักษาซอฟต์แวร์ของซอร์ซโค้ดภายหลังจากการปรับแก้ไขนั้นเพื่อพิจารณาว่าการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดด้วยอัลกอริทึมละโมบนั้นจะทำให้ได้ซอร์ซโค้ดที่มีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์มากกว่าซอร์ซโค้ดที่ถูกปรับแก้ไขโดยไม่พิจารณาการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดหรือสุ่มเลือกลำดับในการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไข และพิจารณาว่าลำดับการใช้งานวิธีรีแฟคทอริงที่ได้จากการใช้อัลกอริทึมละโมบนั้นเป็นลำดับที่ทำให้ได้ซอร์ซโค้ดภายหลังจากการแก้ไขนั้นมีค่าความสามารถในการบำรุงรักษามากที่สุดจากลำดับทั้งหมดที่เป็นไปได้ในการแก้ไขโค้ดนั้นๆ หรือไม่

4.1.1 การเปรียบเทียบค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ของซอร์ซโค้ดระหว่างซอร์ซโค้ดที่ถูกปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบกับซอร์ซโค้ดที่ปรับแก้ไขโดยไม่พิจารณาการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด

1. ระบบเช่าภาพยนตร์

ในส่วนของการเปรียบเทียบค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ของซอร์ซโค้ดระหว่างซอร์ซโค้ดที่ถูกปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงโดยอัลกอริทึมละโมบ กับซอร์ซโค้ดที่ปรับแก้ไขโดยไม่พิจารณาการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดของระบบเช่าภาพยนตร์นั้น จะทำการเปรียบเทียบโดยนำเอาผลลัพธ์ของซอร์ซโค้ดเมท็อด Statement ของคลาส Customer ภายหลังจากปรับแก้ไขด้วยลำดับ R2,R4,R6 ที่ได้จากการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโดยอัลกอริทึมละโมบจากบทที่ 3 ซึ่งมีรายละเอียดของแต่ละมาตรวัดดังตารางที่ 14 คือ ผลรวมค่าความซับซ้อนต่อคลาสมีค่าเท่ากับ 9 หน่วย ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสมีค่าเท่ากับ 0.6 หน่วย แอฟเฟอร์เรนคัปปลิงมีค่าเท่ากับ 0 หน่วย และเอฟเฟอร์เรนคัปลิงมีค่าเท่ากับ 1 หน่วย นำมาเปรียบกับผลลัพธ์ที่ได้จากการแก้ปรับแก้ไขเมท็อด Statement ด้วยลำดับ R3,R5,R1 ที่เกิดจากการสุ่มโดยไม่พิจารณาการจัดลำดับการใช้งานวิธีรีแฟคทอริง

จากตารางที่ 15 การแก้ไขเมท็อด Statement ของคลาส Customer ด้วยวิธีรีแฟคทอริงโดยไม่พิจารณาลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมแบบละโมบ จากการสุ่มเลือกปรับแก้ไขตามลำดับ R3, R5 และ R1 ซอร์ซโค้ดภายหลังจากการปรับแก้ไขนั้นจะมีค่ามาตรวัดต่างๆ มีรายละเอียดดังนี้คือ ผลรวมค่าความซับซ้อนต่อคลาสมีค่าเท่ากับ 22 หน่วย ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสมีค่าเท่ากับ 0.6 หน่วย แอฟเฟอร์เรนคัปปลิงมีค่าเท่ากับ 0 หน่วย และเอฟเฟอร์เรนคัปปลิงมีค่าเท่ากับ 1 หน่วย ซึ่งตารางแสดงการเปรียบเทียบค่ามาตรวัดของซอร์ซโค้ดแสดงได้ดังตารางที่ 16 เครื่องหมายลบ (-) หมายถึง ค่ามาตรวัดของซอร์ซโค้ดภายหลังจากการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบมีค่าที่ดีกว่าของซอร์ซโค้ดภายหลังจากการปรับแก้ไขด้วยวิธีรีแฟคทอริงโดยไม่พิจารณาลำดับการใช้งานวิธีรีแฟคทอริง

เครื่องหมายบวก (+) หมายถึง ค่ามาตรวัดของซอร์ซโค้ดภายหลังจากการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบมีค่าที่แย่กว่าของซอร์ซโค้ดภายหลังจากการปรับแก้ไขด้วยวิธีรีแฟคทอริงโดยไม่พิจารณาลำดับการใช้งานวิธีรีแฟคทอริง และศูนย์ (0) หมายถึง ค่ามาตรวัดของซอร์ซโค้ดภายหลังจากการปรับแก้ไขด้วยการเรียงลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบได้ค่าไม่แตกต่างกับของซอร์ซโค้ดภายหลังจากการปรับแก้ไขด้วยวิธีรีแฟคทอริงโดยไม่พิจารณาลำดับการใช้งานวิธีรีแฟคทอริง

ตารางที่ 14 แสดงค่ามาตรวัดของเมท็อด Statement ของคลาส Customer ภายหลังการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ

โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
P1[R2]	7	0.667	0	1
P2[R2,R4]	7	0.625	0	1
P3[R2,R4,R6]	9	0.6	0	1

ตารางที่ 15 แสดงค่ามาตรวัดของเมท็อด Statement ของคลาส Customer ภายหลังการปรับแก้ไขด้วยวิธีรีแฟคทอริงโดยไม่พิจารณาลำดับการใช้งานวิธีรีแฟคทอริง

โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
P2[R3]	14	0.625	0	1
P3[R3,R5]	21	0.6	0	1
P1[R3,R5,R1]	22	0.6	0	1

ตารางที่ 16 แสดงการเปรียบเทียบค่ามาตรวัดของเมท็อด Statement ของคลาส Customer ภายหลังจากการปรับแก้ได้วยวิธีรีแฟคทอริง ระหว่างการพิจารณาและไม่พิจารณาการจัดลำดับวิธีรีแฟคทอริง

รอบ	ค่าความแตกต่างของมาตรวัด (พิจารณาลำดับการใช้งานวิธีรีแฟคทอริง – ไม่พิจารณาลำดับการใช้งานวิธีรีแฟคทอริง)			
	ผลรวมค่าความซับซ้อนต่อคลาส	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
1	-7	+0.042	0	0
2	-14	0	0	0
3	-13	0	0	0

จากตารางที่ 16 รอบที่ 1 ของการแก้ไขเมท็อด Statement ด้วยการเลือกปรับแก้ไขโค้ดในส่วนของการคำนวณค่าเช่าภาพยนตร์ด้วยการใช้วิธีรีแฟคทอริงแบบ Move Method ในการย้ายซอร์ซโค้ดส่วนของการคำนวณค่าเช่าภาพยนตร์ไปเป็นเมท็อดใหม่ภายใต้คลาส Rental (R2) จะทำให้ได้ค่ามาตรวัดผลรวมค่าความซับซ้อนต่อคลาสน้อยกว่าการเลือกปรับแก้ไขโค้ดในส่วนของการคำนวณค่าแต้มความถี่สะสมในการเช่าภาพยนตร์ด้วยการใช้วิธีรีแฟคทอริงแบบ Extract Method แยกส่วนซอร์ซโค้ดนี้ไปเป็นเมท็อดใหม่ภายใต้คลาส Customer (R3) เท่ากับ 7 หน่วย แต่การเลือกปรับแก้ไขด้วยวิธีรีแฟคทอริงแบบ R2 นั้นจะทำให้ได้ค่ามาตรวัดของระดับการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสสูงกว่าการปรับแก้ไขด้วยวิธีรีแฟคทอริงแบบ R3 เท่ากับ 0.042 หน่วย โดยที่ค่าของแอฟเฟอร์เรนคัปปลิงและเอฟเฟอร์เรนคัปปลิงของทั้ง 2 วิธีนั้นไม่แตกต่างกัน

รอบที่ 2 ของการแก้ไขนั้นการเลือกปรับแก้ไขโค้ดในส่วนของการคำนวณแต้มความถี่สะสมในการเช่าภาพยนตร์ด้วยการใช้วิธีรีแฟคทอริงแบบ Move Method ในการย้ายซอร์ซโค้ดส่วนของการคำนวณแต้มความถี่สะสมในการเช่าภาพยนตร์ไปเป็นเมท็อดใหม่ภายใต้คลาส Rental (R4) จะได้ค่ามาตรวัดผลรวมค่าความซับซ้อนต่อคลาสน้อยกว่าการปรับแก้ไขโค้ดในส่วนของการคำนวณค่าเช่าภาพยนตร์รวมด้วยการใช้วิธีรีแฟคทอริงแบบ Extract Method แยกส่วนซอร์ซโค้ดนี้ไปเป็นเมท็อดใหม่ภายใต้คลาส Customer เท่ากับ 14 หน่วย โดยที่มาตรวัดตัวอื่นๆ นั้นได้ค่าไม่แตกต่างกัน

ส่วนของรอบสุดท้ายของการแก้ไขนั้นการเลือกปรับแก้ไขโค้ดส่วนของการคำนวณค่าเช่าภาพยนตร์รวมด้วยการใช้วิธีรีแฟคทอริงแบบ Move Method ในการย้ายซอร์ซโค้ดส่วนของการคำนวณค่าเช่าภาพยนตร์รวมไปเป็นเมท็อดใหม่ภายใต้คลาส Rental (R6) จะได้ค่าผลรวมค่าความซับซ้อนต่อคลาสน้อยกว่าการปรับแก้ไขโค้ดในส่วนของการคำนวณค่าเช่าภาพยนตร์ด้วยการใช้วิธีรีแฟคทอริงแบบ Extract Method ในการย้ายซอร์ซโค้ดส่วนนี้ไปเป็นเมท็อดใหม่ภายใต้คลาส Customer เท่ากับ 13 หน่วย

เมื่อสิ้นสุดการปรับแก้ไขจะได้ค่าผลรวมค่าความซับซ้อนต่อคลาสของซอร์ซโค้ดที่ปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบนั้นจะมีค่าน้อยกว่าของซอร์ซโค้ดที่ปรับแก้ไขด้วยวิธีรีแฟคทอริงโดยไม่พิจารณาลำดับการใช้งานวิธีรีแฟคทอริง โดยที่ค่ามาตรวัดทั้ง 3 ตัวที่เหลือมีค่าเท่ากัน ทำให้ค่าความสามารถในการบำรุงรักษาซอฟตแวร์ของซอร์ซโค้ดที่ปรับแก้ไขด้วยวิธีรีแฟคทอริงด้วยการเรียงลำดับการใช้งานวิธีรีแฟคทอริงมีค่ามากกว่าซอร์ซโค้ดที่ปรับแก้ไขด้วยวิธีรีแฟคทอริงโดยไม่พิจารณาการเรียงลำดับการใช้งานวิธีรีแฟคทอริง สรุปได้ว่าการปรับแก้ไขโค้ดด้วยวิธีรีแฟคทอริงด้วยการเรียงลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบมีส่วนช่วยทำให้ได้ซอร์ซโค้ดภายหลังจากการปรับแก้ไขมีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์มากขึ้นกว่าเดิม

2. ระบบคำนวณค่าไฟฟ้า

ในส่วนของการเปรียบเทียบค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ของซอร์ซโค้ดระหว่างซอร์ซโค้ดที่ถูกปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงโดยอัลกอริทึมละโมบ กับซอร์ซโค้ดที่ปรับแก้ไขโดยไม่พิจารณาการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดของระบบคำนวณค่าไฟฟ้านั้น จะทำการเปรียบเทียบโดยนำเอาผลลัพธ์ของซอร์ซโค้ดเมท็อด Charge ของทั้งคลาส DisabilitySite และคลาส ResidentialSite ภายหลังจากปรับแก้ไขด้วยลำดับ R2,R1,R3 ของคลาส DisabilitySite (รายละเอียดของแต่ละการปรับแก้ไขแสดงได้ดังตารางที่ 30) และลำดับ R5,R4,R6 ของคลาส ResidentialSite (รายละเอียดของแต่ละการปรับแก้ไขแสดงได้ดังตารางที่ 32) ที่ได้จากการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโดยอัลกอริทึมละโมบจากภาคผนวก ก การปรับแก้ไขซอร์ซโค้ดด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงนั้น จะนำเอามาตรวัดค่าความซับซ้อนของเมคเคป (V(G)) และจำนวนบรรทัดของเมท็อด (Method Lines of Code) มาใช้เป็นเกณฑ์พิจารณาเพิ่มเติม เนื่องจากวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขนั้นมีเพียงแค่การปรับแก้ไขภายในคลาสของตัวเอง ได้แก่ วิธีรีแฟคทอริงแบบ Replace Temp with Query วิธี Extract Method และวิธี Replace Parameters with Method ทำให้ค่ามาตรวัดของระดับการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส แอฟเฟอร์เรนคัปปลิง และเอฟเฟอร์เรนคัปปลิง ไม่มีการเปลี่ยนแปลง จึงทำให้ต้องนำเอามาตรวัดของค่าความซับซ้อนของเมคเคปและจำนวนบรรทัดของเมท็อดมาพิจารณาเพิ่มเติมร่วมกับผลรวมค่าความซับซ้อนต่อคลาส

รายละเอียดของแต่ละมาตรวัดของคลาส DisabilitySite ภายหลังการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ แสดงดังตารางที่ 17 คือ ผลรวมค่าความซับซ้อนต่อคลาสมีค่าเท่ากับ 33 หน่วย ค่าความซับซ้อนของเมคเคปของเมท็อด Charge เท่ากับ 2 หน่วย จำนวนบรรทัดของเมท็อด Charge เท่ากับ 10 หน่วย ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสมีค่าเท่ากับ 0.5 หน่วย แอฟเฟอร์เรนคัปปลิงมีค่าเท่ากับ 0 หน่วย และเอฟเฟอร์เรนคัปลิงมีค่าเท่ากับ 1 หน่วย และรายละเอียดของแต่ละมาตรวัดของคลาส ResidentialSite ภายหลังจากการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ แสดงดังตารางที่ 18 คือ ผลรวมค่าความซับซ้อนต่อคลาสมีค่าเท่ากับ 35 หน่วย ค่าความซับซ้อนของเมคเคปของเมท็อด Charge เท่ากับ 2 หน่วย จำนวนบรรทัดของเมท็อด Charge

เท่ากับ 7 หน่วย ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสมีค่าเท่ากับ 0.5 หน่วย แอฟเฟอร์เรนคัปปลิงมีค่าเท่ากับ 0 หน่วย และเอฟเฟอร์เรนคัปลิงมีค่าเท่ากับ 1 หน่วย โดยจะนำคลาสทั้ง 2 มาเปรียบกับผลลัพธ์ที่ได้จากการแก้ปรับแก้ไขเมท็อด Charge ของคลาส DisabilitySite ด้วยลำดับ R3,R2,R1 และเมท็อด Charge ของคลาส ResidentialSite ด้วยลำดับ R6,R5,R4 ที่เกิดจากการสุ่มโดยไม่พิจารณาการจัดลำดับการใช้งานวิธีรีแฟคทอริง

ตารางที่ 17 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส DisabilitySite ภายหลังการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ

โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ค่าความซับซ้อนของเมคเคปของเมท็อด Charge	จำนวนบรรทัดของเมท็อด Charge	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
P2[R2]	31	2	10	0.5	0	1
P1[R2,R1]	32	2	10	0.5	0	1
P3[R2,R1,R3]	33	2	10	0.5	0	1

ตารางที่ 18 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส ResidentialSite ภายหลังการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ

โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ค่าความซับซ้อนของเมคเคปของเมท็อด Charge	จำนวนบรรทัดของเมท็อด Charge	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
P5[R5]	32	2	8	0.5	0	1
P4[R5,R4]	33	2	7	0.5	0	1
P6[R5,R4,R6]	35	2	7	0.5	0	1

ตารางที่ 19 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส DisabilitySite ภายหลังการปรับแก้ไขโดยไม่พิจารณาการจัดลำดับการใช้งานวิธีรีแฟคทอริง

โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ค่าความซับซ้อนของเมคเคปของเมท็อด Charge	จำนวนบรรทัดของเมท็อด Charge	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
P3[R3]	32	9	25	0.5	0	1
P2[R3,R2]	33	2	10	0.5	0	1
P1[R3,R2,R1]	33	2	10	0.5	0	1

ตารางที่ 20 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส ResidentialSite ภายหลังการปรับแก้ไขโดยไม่พิจารณาการจัดลำดับการใช้งานวิธีรีแฟคทอริง

โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ค่าความซับซ้อนของเมคเคปของเมท็อด Charge	จำนวนบรรทัดของเมท็อด Charge	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
P6[R6]	31	9	25	0.5	0	1
P5[R6,R5]	34	2	8	0.5	0	1
P4[R6,R5,R4]	35	2	7	0.5	0	1

จากตารางที่ 19 - ตารางที่ 20 จะเห็นได้ว่าซอร์ซโค้ดที่ได้จากการปรับแก้ไขโดยไม่พิจารณาการจัดลำดับการใช้งานวิธีรีแฟคทอริงของคลาส DisabilitySite และคลาส ResidentialSite นั้นมีค่าเท่ากับซอร์ซโค้ดที่ได้ภายหลังจากการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมแบบละโมบ เนื่องจากลักษณะของตำแหน่งที่ต้องปรับแก้ไขนั้นในแต่ละตำแหน่งมีเพียงวิธีเดียวในการปรับแก้ไขเท่านั้น ทำให้ผลลัพธ์ของซอร์ซโค้ดที่ได้นั้นลำดับการใช้งานวิธีรีแฟคทอริงไม่มีผลต่อการปรับแก้ไข

ตารางที่ 21 แสดงการเปรียบเทียบค่ามาตรวัดของเมท็อด Charge ของคลาส DisabilitySite ภายหลังจากการปรับแก้ไขด้วยวิธีรีแฟคทอริง ระหว่างการพิจารณาและไม่พิจารณาการจัดลำดับวิธีรีแฟคทอริง

รอบ	ค่าความแตกต่างของมาตรวัด (พิจารณาลำดับการใช้งานวิธีรีแฟคทอริง – ไม่พิจารณาลำดับการใช้งานวิธีรีแฟคทอริง)					
	ผลรวมค่าความซับซ้อนต่อคลาส	ค่าความซับซ้อนของเมคเคปของเมท็อด Charge	จำนวนบรรทัดของเมท็อด Charge	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
1	-1	-7	-15	0	0	0
2	-1	0	0	0	0	0
3	0	0	0	0	0	0

จากตารางที่ 21 รอบที่ 1 ของการแก้ไขเมท็อด Charge ของคลาส DisabilitySite ด้วยการใช้วิธีแฟคทอริงแบบ Extract Method ในการย้ายโค้ดส่วนของการคำนวณค่า summerFlagtion มาสร้างเป็นเมท็อดใหม่ คือ เมท็อด summerFlagtion และ fuelCharge-Taxes ทำให้ค่าผลรวมค่าความซับซ้อนต่อคลาสนั้นน้อยกว่าการปรับแก้ไขด้วยวิธีรีแฟคทอริงแบบ Replace Parameters with Method โดยการแทนที่การใช้พารามิเตอร์ fullUsage ด้วยการสร้าง

เมท็อด lastUsage ขึ้นมาใหม่อยู่ 1 หน่วย ได้ค่าซับซ้อนของเมคเคปน้อยกว่าอยู่ 7 หน่วย และค่าจำนวนบรรทัดของเมท็อดน้อยกว่าอยู่ 15 หน่วย

รอบที่ 2 ของการแก้ไขการใช้วิธีแฟคทอริงแบบ Replace Temp with Query เพื่อลดการใช้งานตัวแปร fuel ด้วยการสร้างเมท็อด fuelCharge ขึ้นเพื่อให้เรียกใช้งานแทน ทำให้ค่าผลรวมค่าความซับซ้อนต่อคลาสนั้นน้อยกว่าการปรับแก้ไขด้วยการใช้วิธีแฟคทอริงแบบ Extract Method ในการย้ายโค้ดส่วนของการคำนวณค่า summerFlagtion มาสร้างเป็นเมท็อดใหม่ คือ เมท็อด summerFlagtion และ fuelChargeTaxes อยู่ 1 หน่วย โดยได้ค่าซับซ้อนของเมคเคปและค่าจำนวนบรรทัดของเมท็อดนั้นเท่ากัน

รอบสุดท้ายของการแก้ไขการปรับแก้ไขด้วยวิธีรีแฟคทอริงแบบ Replace Parameters with Method โดยการแทนที่การใช้พารามิเตอร์ fullUsage ด้วยการสร้างเมท็อด lastUsage ขึ้นมาใหม่ กับการปรับแก้ไขการใช้วิธีแฟคทอริงแบบ Replace Temp with Query เพื่อลดการใช้งานตัวแปร fuel ด้วยการสร้างเมท็อด fuelCharge ขึ้นเพื่อให้เรียกใช้งานแทนสุดท้ายจะได้ค่ามาตรวัดที่เท่ากันทุกตัว

ตารางที่ 22 แสดงการเปรียบเทียบค่ามาตรวัดของเมท็อด Charge ของคลาส ResidentialSite ภายหลังจากการปรับแก้ไขด้วยวิธีรีแฟคทอริง ระหว่างการพิจารณาและไม่พิจารณาการจัดลำดับวิธีรีแฟคทอริง

รอบ	ค่าความแตกต่างของมาตรวัด (พิจารณาลำดับการใช้งานวิธีรีแฟคทอริง – ไม่พิจารณาลำดับการใช้งานวิธีรีแฟคทอริง)					
	ผลรวมค่าความซับซ้อนต่อคลาส	ค่าความซับซ้อนของเมคเคปของเมท็อด Charge	จำนวนบรรทัดของเมท็อด Charge	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
1	+1	-7	-17	0	0	0
2	-1	0	-1	0	0	0
3	0	0	0	0	0	0

จากตารางที่ 22 รอบที่ 1 ของการแก้ไขเมท็อด Charge ของคลาส ResidentialSite ด้วยการใช้วิธีแฟคทอริงแบบ Extract Method ในการย้ายโค้ดส่วนของการคำนวณค่า summerFlagtion มาสร้างเป็นเมท็อดใหม่ คือ เมท็อด summerFlagtion และ fuelChargeTaxes ทำให้ค่าผลรวมค่าความซับซ้อนต่อคลาสนั้นมากกว่าการปรับแก้ไขด้วยวิธีรีแฟคทอริงแบบ Replace Parameters with Method โดยการแทนที่การใช้พารามิเตอร์ fullUsage ด้วยการสร้างเมท็อด lastUsage ขึ้นมาใหม่อยู่ 1 หน่วย แต่ได้ค่าซับซ้อนของเมคเคปน้อยกว่าอยู่ 7 หน่วย และค่าจำนวนบรรทัดของเมท็อดน้อยกว่าอยู่ 17 หน่วย

รอบที่ 2 ของการแก้ไขการใช้วิธีแฟคทอริงแบบ Replace Temp with Query เพื่อลดการใช้งานตัวแปร fuel ด้วยการสร้างเมท็อด fuelCharge ขึ้นเพื่อให้เรียกใช้งานแทน ทำให้ค่าผลรวมค่าความซับซ้อนต่อคลาสนั้นน้อยกว่าการปรับแก้ไขด้วยการใช้วิธีแฟคทอริงแบบ Extract Method ในการย้ายโค้ดส่วนของการคำนวณค่า summerFlagtion มาสร้างเป็นเมท็อดใหม่ คือ เมท็อด summerFlagtion และ fuelChargeTaxes อยู่ 1 หน่วย และได้ค่าจำนวนบรรทัดของเมท็อดน้อยกว่าอยู่ 1 หน่วย โดยที่ค่าความซับซ้อนของเมคเคปนั้นเท่ากัน

รอบสุดท้ายของการแก้ไขการปรับแก้ไขด้วยวิธีรีแฟคทอริงแบบ Replace Parameters with Method โดยการแทนที่การใช้พารามิเตอร์ fullUsage ด้วยการสร้างเมท็อด lastUsage ขึ้นมาใหม่ กับการปรับแก้ไขการใช้วิธีแฟคทอริงแบบ Replace Temp with Query เพื่อลดการใช้งานตัวแปร fuel ด้วยการสร้างเมท็อด fuelCharge ขึ้นเพื่อให้เรียกใช้งานแทนสุดท้ายจะได้ค่ามาตรวัดที่เท่ากันทุกตัว

สรุปการเปรียบเทียบค่าความสามารถในการบำรุงรักษาซอฟตแวร์ของซอร์ซโค้ดระหว่างซอร์ซโค้ดที่ถูกปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบกับซอร์ซโค้ดที่ปรับแก้ไขโดยไม่พิจารณาการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดนั้น ลักษณะของซอร์ซโค้ดที่แต่ละตำแหน่งมีวิธีรีแฟคทอริงที่สามารถเลือกปรับแก้ไขได้มากกว่า 1 วิธี จะได้ซอร์ซโค้ดภายหลังการปรับแก้ไขโดยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบที่มีค่าความสามารถในการบำรุงรักษาที่ดีกว่าซอร์ซโค้ดที่ไม่พิจารณาการจัดลำดับการใช้งานวิธีรีแฟคทอริง

4.1.2 ตรวจสอบซอร์ซโค้ดภายหลังจากการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดนั้นมีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์สูงสุดหรือไม่

1.ระบบเช่าภาพยนตร์

ในส่วนของการตรวจสอบซอร์ซโค้ดภายหลังจากการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดจะมีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์สูงสุดหรือไม่นั้น จะทำการเปรียบเทียบซอร์ซโค้ดภายหลังจากการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบกับซอร์ซโค้ดทุกแบบที่เป็นไปได้ที่ถูกปรับแก้ไขด้วยวิธีรีแฟคทอริง เพื่อตรวจสอบว่าลำดับการใช้งานวิธีรีแฟคทอริงที่ได้จากการใช้อัลกอริทึมละโมบนั้น จะทำให้ได้ซอร์ซโค้ดภายหลังจากการปรับแก้ไขมีค่าความสามารถในการบำรุงรักษาสูงที่สุดในบรรดาซอร์ซโค้ดที่ถูกปรับแก้ไขด้วยวิธีรีแฟคทอริงในแบบต่างๆ จากตัวอย่างตัวระบบเช่าภาพยนตร์ การแก้ไขเมท็อด Statement ของคลาส Customer จากบทที่ 3 เพื่อขจัดร่องรอยที่ผิดพลาดในเมท็อดนั้นมีตำแหน่งที่ต้องปรับแก้ไขและวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขแสดงดังตารางที่ 23

จากตารางที่ 23 มีตำแหน่งทั้งหมด 3 ตำแหน่ง (P1,P2,P3) ที่ต้องทำการปรับแก้ไข และในแต่ละตำแหน่งนั้นมีวิธีรีแฟคทอริงให้เลือกปรับแก้ไข 2 วิธี (วิธีรีแฟคทอริง R1-R2 สำหรับตำแหน่ง P1 วิธีรีแฟคทอริง R3-R4 สำหรับตำแหน่ง P2 และวิธีรีแฟคทอริง R5-R6 สำหรับตำแหน่ง P3) ในการหาลำดับทั้งหมดที่เป็นไปได้ในการปรับแก้ไขเมท็อด Statement นั้นมาจากการ

สลับลำดับของตำแหน่งที่จะทำการปรับแก้ไข ซึ่งจะได้รูปแบบตำแหน่งที่แตกต่างกันทั้งหมด 6 แบบ (3 แฟคทอเรียล) ได้แก่ [P1,P2,P3] [P1,P3,P2] [P2,P1,P3] [P2,P3,P1] [P3,P1,P2] และ [P3,P2,P1] แต่ละแบบจะมีลำดับวิธีรีแฟคทอริงที่เป็นไปได้ในการปรับแก้ไขทั้งหมด 8 แบบ ดังนั้น ลำดับทั้งหมดที่เป็นไปได้ในการปรับแก้ไขเมท็อด Statement มีค่าเท่ากับ 48 แบบ โดยรายละเอียดของมาตรวัดของซอร์ซโค้ดที่ถูกปรับแก้ไขด้วยวิธีรีแฟคทอริงในแต่ละแบบแสดงได้ดังตารางที่ 24 - ตารางที่ 29

ตารางที่ 23 แสดงรายละเอียดของตำแหน่งที่ต้องปรับแก้ไขและ
วิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขเมท็อด Statement ของคลาส Customer

ตำแหน่งที่ต้องปรับแก้ไข	วิธีรีแฟคทอริงที่ใช้สำหรับปรับแก้ไข
P1 (บรรทัดที่ 8-23) : ส่วนของการคำนวณค่าเช่าภาพยนตร์ของลูกค้า	R1 : วิธี Extract Method แยกซอร์ซโค้ดส่วนของการคำนวณค่าเช่าภาพยนตร์ไปเป็นเมท็อดใหม่ภายใต้คลาส Customer
	R2 : วิธี Move Method ย้ายซอร์ซโค้ดส่วนของการคำนวณค่าเช่าภาพยนตร์ไปเป็นเมท็อดใหม่ภายใต้คลาส Rental
P2 (บรรทัดที่ 25-26) : ส่วนของการคำนวณค่าแต้มความถี่สะสมในการเช่าภาพยนตร์ของลูกค้า	R3 : วิธี Extract Method แยกซอร์ซโค้ดส่วนของการคำนวณค่าแต้มความถี่สะสมในการเช่าภาพยนตร์ไปเป็นเมท็อดใหม่ภายใต้คลาส Customer
	R4 : วิธี Move Method ย้ายซอร์ซโค้ดส่วนของการคำนวณแต้มความถี่สะสมในการเช่าภาพยนตร์ไปเป็นเมท็อดใหม่ภายใต้คลาส Rental
P3 (บรรทัดที่ 28) : ส่วนของการคำนวณค่าเช่าภาพยนตร์รวมในการเช่าภาพยตร์ครั้งหนึ่งๆ ของลูกค้า	R5 : วิธี Extract Method แยกซอร์ซโค้ดส่วนของการคำนวณค่าเช่าภาพยนตร์รวมไปเป็นเมท็อดใหม่ภายใต้คลาส Customer
	R6 : วิธี Move Method ย้ายซอร์ซโค้ดส่วนของการคำนวณค่าเช่าภาพยนตร์รวมไปเป็นเมท็อดใหม่ภายใต้คลาส Rental

ตารางที่ 24 แสดงมาตรวัดของแต่ละลำดับการใช้งานวิธีรีแฟคทอริงในการแก้ไขเมท็อด Statement โดยเลือกแก้ไขในตำแหน่งที่ P1,P2,P3 ตามลำดับ

ลำดับ	ผลรวมค่าความซับซ้อนต่อคลาส	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
R1,R3,R5	22	0.6	0	1
R1,R3,R6	17	0.6	0	1
R1,R4,R5	20	0.6	0	1
R1,R4,R6	15	0.6	0	1
R2,R3,R5	16	0.6	0	1
R2,R3,R6	11	0.6	0	1
R2,R4,R5	14	0.6	0	1
R2,R4,R6	9	0.6	0	1

ตารางที่ 25 แสดงมาตรวัดของแต่ละลำดับการใช้งานวิธีรีแฟคทอริงในการแก้ไขเมท็อด Statement โดยเลือกแก้ไขในตำแหน่งที่ P1,P3,P2 ตามลำดับ

ลำดับ	ผลรวมค่าความซับซ้อนต่อคลาส	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
R1,R5,R3	22	0.6	0	1
R1,R5,R4	20	0.6	0	1
R1,R6,R3	17	0.6	0	1
R1,R6,R4	15	0.6	0	1
R2,R5,R3	16	0.6	0	1
R2,R5,R4	14	0.6	0	1
R2,R6,R3	11	0.6	0	1
R2,R6,R4	9	0.6	0	1

ตารางที่ 26 แสดงมาตรวัดของแต่ละลำดับการใช้งานวิธีรีแฟคทอริงในการแก้ไขเมท็อด Statement โดยเลือกแก้ไขในตำแหน่งที่ P2,P1,P3 ตามลำดับ

ลำดับ	ผลรวมค่าความซับซ้อนต่อคลาส	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
R3,R1,R5	22	0.6	0	1
R3,R1,R6	17	0.6	0	1
R3,R2,R5	16	0.6	0	1
R3,R2,R6	11	0.6	0	1
R4,R1,R5	20	0.6	0	1
R4,R1,R6	15	0.6	0	1
R4,R2,R5	14	0.6	0	1
R4,R2,R6	9	0.6	0	1

ตารางที่ 27 แสดงมาตรวัดของแต่ละลำดับการใช้งานวิธีรีแฟคทอริงในการแก้ไขเมท็อด Statement โดยเลือกแก้ไขในตำแหน่งที่ P2,P3,P1 ตามลำดับ

ลำดับ	ผลรวมค่าความซับซ้อนต่อคลาส	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
R3,R5,R1	22	0.6	0	1
R3,R5,R2	16	0.6	0	1
R3,R6,R1	17	0.6	0	1
R3,R6,R2	11	0.6	0	1
R4,R5,R1	20	0.6	0	1
R4,R5,R2	14	0.6	0	1
R4,R6,R1	15	0.6	0	1
R4,R6,R2	9	0.6	0	1

ตารางที่ 28 แสดงมาตรวัดของแต่ละลำดับการใช้งานวิธีรีแฟคทอริงในการแก้ไขเมท็อด Statement โดยเลือกแก้ไขในตำแหน่งที่ P3,P1,P2 ตามลำดับ

ลำดับ	ผลรวมค่าความซับซ้อนต่อคลาส	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
R5,R1,R3	22	0.6	0	1
R5,R1,R4	20	0.6	0	1
R5,R2,R3	16	0.6	0	1
R5,R2,R4	14	0.6	0	1
R6,R1,R3	17	0.6	0	1
R6,R1,R4	15	0.6	0	1
R6,R2,R3	11	0.6	0	1
R6,R2,R4	9	0.6	0	1

ตารางที่ 29 แสดงมาตรวัดของแต่ละลำดับการใช้งานวิธีรีแฟคทอริงในการแก้ไขเมท็อด Statement โดยเลือกแก้ไขในตำแหน่งที่ P3,P2,P1 ตามลำดับ

ลำดับ	ผลรวมค่าความซับซ้อนต่อคลาส	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
R5,R3,R1	22	0.6	0	1
R5,R3,R2	16	0.6	0	1
R5,R4,R1	20	0.6	0	1
R5,R4,R2	14	0.6	0	1
R6,R3,R1	17	0.6	0	1
R6,R3,R2	11	0.6	0	1
R6,R4,R1	15	0.6	0	1
R6,R4,R2	9	0.6	0	1

จากลำดับการใช้งานวิธีรีแฟคทอริงในการแก้ไขเมท็อด Statement ทั้งหมดนั้น จากตารางที่ 24 - ตารางที่ 29 จะเห็นได้ว่าลำดับการใช้งานวิธีรีแฟคทอริงที่ทำให้ได้ซอร์ซโค้ดภายหลังจากการปรับแก้ไขมีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์สูงที่สุด ได้แก่ [R2,R4,R6] [R2,R6,R4] [R4,R2,R6] [R4,R6,R2] [R6,R2,R4] และ [R6,R4,R2] มีค่ามาตรวัดผลรวมค่าความซับซ้อนต่อคลาสเท่ากับ 9 หน่วย ค่าระดับการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสเท่ากับ 0.6 หน่วย ค่าแอฟเฟอร์เรนคัปปลิงเท่ากับ 0 หน่วย และค่าเอฟเฟอร์เรนคัปปลิงเท่ากับ 1 หน่วย ซึ่งตรงกับลำดับ [R2,R4,R6] ที่ได้จากการจัดลำดับการใช้งานวิธีรีแฟคทอริงโดยอัลกอริทึมละโมบ จึงสรุปได้ว่าลำดับที่ได้จากการจัดลำดับการใช้งานวิธีรีแฟคทอริงโดยอัลกอริทึมละโมบนั้นจะได้ซอร์ซโค้ดภายหลังจากการปรับแก้ไขมีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์สูงที่สุดในบรรดาลำดับที่เป็นไปได้ในการปรับแก้ไขด้วยวิธีรีแฟคทอริง

ในส่วนของการสลับการใช้งานตำแหน่งการใช้งานวิธีรีแฟคทอริงระหว่างวิธี R2 R4 และ R6 ในการปรับแก้ไขเมท็อด Statement นั้น ซอร์ซโค้ดภายหลังจากการปรับแก้ไขจะไม่แตกต่างกัน เพราะการแก้ไขแต่ละตำแหน่งนั้นไม่เกิดความขัดแย้งกันระหว่างซอร์ซโค้ด (conflict) ระหว่างตำแหน่ง P1 P2 และ P3 จะเลือกลำดับใดแก้ไขก่อนจึงไม่มีผลต่อการเปลี่ยนแปลงซอร์ซโค้ดภายหลังจากที่ปรับแก้ไขจนครบทุกตำแหน่ง

2. ระบบคำนวณค่าไฟฟ้า

ในส่วนของระบบคำนวณค่าไฟฟ้านั้นจะแยกพิจารณา 2 คลาสที่ต้องทำการปรับแก้ไข คือ คลาส DisabilitySite และคลาส ResideltialSite ในการแก้ไขเมท็อด Charge ของทั้ง 2 คลาส เพื่อขจัดร่องรอยที่ผิดพลาดที่มีอยู่ในซอร์ซโค้ด โดยเมท็อด Charge ของคลาส Disability-Site นั้นมีตำแหน่งที่ต้องปรับแก้ไขและวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขแสดงดังตารางที่ 30

จากตารางที่ 30 มีตำแหน่งทั้งหมด 3 ตำแหน่ง (P1,P2,P3) ที่ต้องทำการปรับแก้ไข และในแต่ละตำแหน่งนั้นมีวิธีรีแฟคทอริงให้เลือกปรับแก้ไขเพียงแค่หนึ่งวิธี คือ วิธีรีแฟคทอริง R1 สำหรับตำแหน่ง P1 วิธีรีแฟคทอริง R2 สำหรับตำแหน่ง P2 และวิธีรีแฟคทอริง R3 สำหรับตำแหน่ง P3 ในการหาลำดับทั้งหมดที่เป็นไปได้ในการปรับแก้ไขเมท็อด Charge ของคลาส DisabilitySite นั้นมาจากการสลับลำดับของตำแหน่งที่จะทำการปรับแก้ไข ซึ่งจะได้รูปแบบตำแหน่งที่แตกต่างกันทั้งหมด 6 แบบ (3 แฟคทอเรียล) ได้แก่ [P1,P2,P3] [P1,P3,P2] [P2,P1,P3] [P2,P3,P1] [P3,P1,P2] [P3,P2,P1] และแต่ละแบบจะมีลำดับวิธีรีแฟคทอริงที่เป็นไปได้ในการปรับแก้ไขทั้งหมด 3 แบบ ดังนั้นลำดับทั้งหมดที่เป็นไปได้ในการปรับแก้ไขเมท็อด Charge ของคลาส DisabilitySite มีค่าเท่ากับ 6 แบบ โดยรายละเอียดของมาตรวัดของซอร์ซโค้ดที่ถูกปรับแก้ไขด้วยวิธีรีแฟคทอริงในแต่ละแบบแสดงได้ดังตารางที่ 31

ตารางที่ 30 แสดงรายละเอียดของตำแหน่งที่ต้องปรับแก้ไขและ
วิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขเมท็อด Charge ของคลาส DisabilitySite

ตำแหน่งที่ต้องปรับแก้ไข	วิธีรีแฟคทอริงที่ใช้สำหรับปรับแก้ไข
P1 (บรรทัดที่ 27-29) : ส่วนของการคำนวณค่า fuel	R1 : วิธี Replace Temp with Query ลดการใช้งานตัวแปร fuel ด้วยการสร้างเมท็อด fuelCharge เพื่อให้เรียกใช้งานแทน
P2 (บรรทัดที่ 7-22) : ส่วนของการคำนวณค่า summerFlagtion	R2 : วิธี Extract Method ย้ายโค้ดส่วนของการคำนวณค่า summerFlagtion มาสร้างเป็นเมท็อดใหม่ คือ เมท็อด summerFlagtion และเมท็อด fuelChargeTaxes แทน
P3 (บรรทัดที่ 6, 25-27) : ส่วนของการคำนวณค่า fullUsage	R3 : วิธี Replace Parameters with Method แทนที่การใช้พารามิเตอร์ fullUsage ด้วยการสร้างเมท็อด lastUsage

ตารางที่ 31 แสดงมาตรวัดของแต่ละลำดับในการปรับแก้ไขเมท็อด Charge
ของคลาส DisabilitySite

โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ค่าความซับซ้อนของเมคเคปของเมท็อด Charge	จำนวนบรรทัดของเมท็อด Charge	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
R1,R2,R3	33	2	10	0.5	0	1
R1,R3,R2	33	2	10	0.5	0	1
R2,R1,R3	33	2	10	0.5	0	1
R2,R3,R1	33	2	10	0.5	0	1
R3,R1,R2	33	2	10	0.5	0	1
R3,R2,R1	33	2	10	0.5	0	1

จากลำดับการใช้งานวิธีรีแฟคทอริงในการแก้ไขเมท็อด Charge ของคลาส DisabilitySite ทั้งหมดนั้นจากตารางที่ 31 จะเห็นได้ว่าทุกลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดนั้นจะได้ซอร์ซโค้ดภายหลังจากการปรับแก้ไขที่มีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์เป็นค่าเดียวกันหมด เนื่องจากลักษณะของตำแหน่งที่ต้องปรับแก้ไขนั้นในแต่ละตำแหน่งมีเพียงวิธีเดียวในการปรับแก้ไขเท่านั้น ทำให้การสลับตำแหน่งการปรับแก้ไขโค้ดจึงไม่มีผลต่อซอร์ซโค้ดภายหลังจากปรับแก้ไขในทุกตำแหน่ง

ส่วนของเมท็อด Charge ของคลาส ResidentialSite นั้นมีตำแหน่งที่ต้องปรับแก้ไขและวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขแสดงดังตารางที่ 32

ตารางที่ 32 แสดงรายละเอียดของตำแหน่งที่ต้องปรับแก้ไขและวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขเมท็อด Charge ของคลาส ResidentialSite

ตำแหน่งที่ต้องปรับแก้ไข	วิธีรีแฟคทอริงที่ใช้สำหรับปรับแก้ไข
P4 (บรรทัดที่ 29-31) : ส่วนของการคำนวณค่า fuel	R4 : วิธี Replace Temp with Query ลดการใช้งานตัวแปร fuel ด้วยการสร้างเมท็อด fuelCharge เพื่อให้เรียกใช้งานแทน
P5 (บรรทัดที่ 7-27) : ส่วนของการคำนวณค่า summerFlagtion	R5 : วิธี Extract Method ย้ายโค้ดส่วนของการคำนวณค่า summerFlagtion มาสร้างเป็นเมท็อดใหม่ คือ เมท็อด summerFlagtion และเมท็อด fuelChargeTaxes แทน
P6 (บรรทัดที่ 3, 27-29) : ส่วนของการคำนวณค่า fullUsage	R6 : วิธี Replace Parameters with Method แทนที่การใช้พารามิเตอร์ fullUsage ด้วยการสร้างเมท็อด lastUsage

ตารางที่ 33 แสดงมาตรวัดของแต่ละลำดับในการปรับแก้ไขเมท็อด Charge ของคลาส ResidentialSite

โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ค่าความซับซ้อนของเมคเคปของเมท็อด Charge	จำนวนบรรทัดของเมท็อด Charge	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรน คัปปลิง	เอฟเฟอร์เรน คัปปลิง
R4,R5,R6	35	2	7	0.5	0	1
R4,R6,R5	35	2	7	0.5	0	1
R5,R4,R6	35	2	7	0.5	0	1
R5,R6,R4	35	2	7	0.5	0	1
R6,R4,R5	35	2	7	0.5	0	1
R6,R5,R4	35	2	7	0.5	0	1

จากลำดับการใช้งานวิธีรีแฟคทอริงในการแก้ไขเมท็อด Charge ของคลาส ResidentialSite ทั้งหมดนั้นตามตารางที่ 33 จะเห็นได้ว่าทุกลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดนั้นจะได้ซอร์ซโค้ดภายหลังจากการปรับแก้ไขที่มีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์เป็นค่าเดียวกันหมดซึ่งจะเหมือนกันกับการปรับแก้ไขเมท็อด Charge ของคลาส Disabili-

tySite เนื่องจากลักษณะของตำแหน่งที่ต้องปรับแก้ไขนั้นในแต่ละตำแหน่งมีเพียงวิธีเดียวในการปรับแก้ไขเท่านั้น ทำให้การสลับตำแหน่งการปรับแก้ไขโค้ดจึงไม่มีผลต่อซอร์ซโค้ดภายหลังจากปรับแก้ไขในทุกตำแหน่ง

สรุปการตรวจสอบซอร์ซโค้ดภายหลังจากการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดนั้น ลักษณะซอร์ซโค้ดที่มีตำแหน่งหนึ่งสามารถปรับแก้ไขด้วยวิธีรีแฟคทอริงมากกว่า 1 วิธีนั้น ซอร์ซโค้ดที่ได้จากการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงมีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ที่สูงที่สุด แต่การจัดลำดับการใช้งานวิธีรีแฟคทอริงจะไม่มีผลกับซอร์ซโค้ดที่มีลักษณะที่แต่ละตำแหน่งที่ต้องปรับแก้ไขสามารถใช้งานวิธีรีแฟคทอริงเพียงวิธีเดียวที่สามารถใช้ปรับแก้ไข แต่ละลำดับที่ได้ภายหลังจากปรับแก้ไขครบทุกตำแหน่งนั้นจะได้ค่าความสามารถในการบำรุงรักษาซอฟต์แวร์เท่ากันทุกลำดับ

4.2 เวลาที่ใช้ในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดด้วยอัลกอริทึมละโมบ

ในการประเมินด้านเวลาที่ใช้ในการค้นหาลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขด้วยอัลกอริทึมละโมบนั้นจะพิจารณาจากจำนวนโหนดทั้งหมดที่จะต้องสืบค้นเพื่อจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด โดยจะทำการเปรียบเทียบกับการใช้งานอัลกอริทึมแบบอื่นๆ ในการจัดลำดับการใช้งานวิธีรีแฟคทอริง เพื่อเป็นการยืนยันว่าการใช้อัลกอริทึมละโมบนั้นใช้เวลาในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดน้อยที่สุดในบรรดาอัลกอริทึมแบบอื่นๆ อัลกอรึทึมที่นำมาเปรียบเทียบกับอัลกอริทึมละโมบ ได้แก่ อัลกอริทึมค้นหาแนวกว้าง (Breadth First Search) อัลกอรึทึมปีนเขา (Hill Climbing) และอัลกอริทึมเอสตาร์ (A* - A Star) รายละเอียดการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดของแต่ละอัลกอริทึมมีนั้นอยู่ที่ภาคผนวก ก โดยตารางแสดงการเปรียบเทียบรายละเอียดต่างๆ ระหว่างอัลกอริทึมทั้ง 4 แบบ แสดงได้ดังตารางที่ 34

ตารางที่ 34 แสดงการเปรียบเทียบรายละเอียดต่างๆ ระหว่างอัลกอริทึมทั้ง 4 แบบ ในการจัดลำดับการใช้งานวิธีรีแฟคทอริง

คุณลักษณะ	อัลกอริทึม			
	ละโมบ	ค้นหาแนวกว้าง	ปีนเขา	เอสตาร์
1. ลักษณะการค้นหา	เป็นเส้นตรงไม่ย้อนกลับเส้นทางเดิม	ย้อนกลับเส้นทางเดิมที่ดีกว่า	เป็นเส้นตรงไม่ย้อนกลับเส้นทางเดิม	เป็นเส้นตรงพิจารณาค่าฟังก์ชันการเดินทางไปยังจุดหมาย
2. ลักษณะการปรับแก้ไข	ครบทุกตำแหน่ง	ครบทุกตำแหน่ง	ไม่ครบทุกตำแหน่ง	ครบทุกตำแหน่ง

จากตารางที่ 34 ลักษณะการค้นหาของอัลกอริทึมละโมบ อัลกอริทึมปีนเขาและอัลกอริทึมเอสตาร์เป็นเส้นตรงไม่ย้อนกลับทางเดิม แต่จะแตกต่างกันตรงที่อัลกอริทึมปีนเขานั้นจะหยุดก็ต่อเมื่อไม่พบโหนดหรือเส้นทางที่ดีกว่าเดิมจึงทำให้มีโอกาสที่จะหยุดทำการปรับแก้ไขโค้ดก่อนที่จะปรับแก้ไขครบทุกตำแหน่ง ซึ่งจะแตกต่างจากอัลกอริทึมละโมบและเอสตาร์ที่จะแก้ไขจนครบทุกตำแหน่งในส่วนของอัลกอริทึมค้นหาแนวกว้างจะพิจารณาโหนดก่อนหน้านี้ เพื่อเลือกเส้นทางเดินที่ดีที่สุด จึงมีโอกาสที่จะเดินย้อนกลับเส้นทางเดิมที่ดีกว่าได้และจะแก้ไขจนครบทุกตำแหน่งก่อนที่จะหยุดการปรับแก้ไข

รายละเอียดของการค้นหาของแต่ละอัลกอริทึมมีรายละเอียดดังนี้

1. ระบบเช่าภาพยนตร์

ตารางที่ 35 แสดงค่ามาตรวัดของคลาส Customer ภายหลังจากปรับแก้ไขด้วยการจัดลำดับวิธีรีแฟคทอริงด้วยอัลกอริทึมแบบต่างๆ

อัลกอริทึม	มาตรวัด				ลำดับการใช้งานวิธีรีแฟคทอริง
	ผลรวมค่าความซับซ้อนต่อคลาส	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง	
ละโมบ	9	0.6	0	1	R2,R4,R6
ค้นหาแนวกว้าง	9	0.6	0	1	R2,R4,R6
ปีนเขา	7	0.625	0	1	R2,R4
เอสตาร์	9	0.6	0	1	R2,R4,R6

จากตารางที่ 35 อัลกอริทึมละโมบ อัลกอริทึมค้นหาแนวกว้าง และอัลกอริทึมเอสตาร์ สามารถปรับแก้ไขได้ครบทุกตำแหน่งภายในเมท็อด Statement ของคลาส Customer โดยได้ลำดับการใช้งานวิธีรีแฟคทอริง คือ S[R2,R4,R6] ซึ่งมีรายละเอียดของมาตรวัด ดังนี้ ค่าผลรวมค่าความซับซ้อนต่อคลาสเท่ากับ 9 หน่วย ค่าระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสเท่ากับ 0.6 หน่วย ค่าแอฟเฟอร์เรนคัปปลิงเท่ากับ 0 หน่วย และค่าเอฟเฟอร์เรนคัปปลิงเท่ากับ 1 หน่วย ในส่วนอัลกอริทึมปีนเขานั้นจะสามารถปรับแก้ไขได้เพียง 2 ตำแหน่งเท่านั้น คือ S[R2,R4] เนื่องจากการปรับแก้ไขในส่วนของการคำนวณค่าเช่าภาพยนตร์รวมในการเช่าภาพยตร์ครั้งหนึ่งๆ ของลูกค้าหรือตำแหน่ง P3 ได้ค่ามาตรวัดหลังจากการปรับแก้ไขด้วยวิธีรีแฟคทอริงแล้วมีค่าที่แย่กว่าก่อนการปรับแก้ไขที่ตำแหน่งดังกล่าว คือ ค่าผลรวมค่าความซับซ้อนต่อคลาสเท่ากับ 7 หน่วย ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสเท่ากับ 0.625 หน่วย จึงหยุดการจัดลำดับการใช้งานวิธีรีแฟคทอริงที่การแก้ไขเพียง 2 ตำแหน่ง คือ ส่วนของการคำนวณค่าแต้มความถี่สะสมในการ

เช่าภาพยนตร์ของลูกค้าหรือตำแหน่ง P1 และส่วนของการคำนวณค่าเช่าภาพยนตร์รวมในการเช่าภาพยตร์ครั้งหนึ่งๆ ของลูกค้าหรือตำแหน่ง P2 เท่านั้น

สรุปการใช้อัลกอริทึมละโมบ อัลกอริทึมค้นหาแนวกว้าง และอัลกอริทึมเอสตาร์ในการจัดลำดับการใช้งานวิธีรีแฟคทอริงแก้ไขเมท็อด Statement ของคลาส Customer ทำให้ได้ซอร์ซโค้ดที่ขจัดร่องรอยที่ผิดพลาดได้ครบทุกตำแหน่ง คุณภาพของซอร์ซโค้ดที่ได้จึงดีกว่าการใช้อัลกอริทึมปีนเขาที่ขจัดร่องรอยที่ผิดพลาดได้ไม่ครบทุกตำแหน่ง

ในส่วนของผลลัพธ์การพิจารณาเวลาที่ใช้ในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขเมท็อด Statement ของคลาส Customer ของแต่ละอัลกอริทึมจากตารางที่ 36 จะเห็นว่าอัลกอริทึมปีนเขาใช้เวลาในการจัดลำดับการใช้งานวิธีรีแฟคทอริงน้อยที่สุด คือ ใช้เพียง 10 โหนดในการปรับแก้ไข แต่ยังคงเหลือโค้ดในส่วนของการคำนวณค่าเช่าภาพยนตร์รวมในการเช่าภาพยตร์ครั้งหนึ่งๆ ของลูกค้าหรือตำแหน่ง P3 ที่ไม่ได้แก้ไข เนื่องจากเงื่อนไขของอัลกอริทึมปีนเขาที่จะหยุดการค้นหาเมื่อไม่พบโหนดหรือเส้นทางถัดไปที่ให้ผลลัพธ์ที่ดีกว่าเส้นทางปัจจุบัน จึงทำให้เวลาในจัดการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบใช้เวลาน้อยสุดในการปรับแก้ไขเมท็อด Statement ทุกตำแหน่ง คือ พิจารณาเพียง 12 โหนด ซึ่งดีกว่าการใช้อัลกอริทึมค้นหาแนวกว้างและอัลกอริทึมเอสตาร์ที่พิจารณาโหนดทั้งหมด 20 โหนดและ 56 โหนดตามลำดับ ในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขเมท็อด Statement สรุปในการประเมินด้านเวลาในการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบใช้เวลาน้อยกว่าอัลกอริทึมทั้งสามแบบ

ตารางที่ 36 แสดงจำนวนโหนดทั้งหมดที่ใช้ในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขเมท็อด Statement ของคลาส Customer โดยการใช้อัลกอริทึมแบบต่างๆ

รอบที่	อัลกอริทึม (โหนด)			
	ละโมบ	ค้นหาแนวกว้าง	ปีนเขา	เอสตาร์
1	6	6	6	48
2	4	9	4	16
3	2	5	-	2

2. ระบบคำนวณค่าไฟฟ้า

การพิจารณารายละเอียดการค้นหาของแต่ละอัลกอริทึมของระบบคำนวณค่าไฟฟ้านั้นจะแยกพิจารณาออกเป็น 2 คลาส คือ คลาส DisabilitySite และคลาส ResidentialSite จากตารางที่ 37 อัลกอริทึมละโมบ อัลกอริทึมค้นหาแนวกว้าง และอัลกอริทึมเอสตาร์ สามารถปรับแก้ไขได้ครบทุกตำแหน่งภายในเมท็อด Charge ของคลาส DisabilitySite โดยได้ลำดับการใช้งานวิธีรีแฟคทอริง คือ D[R2,R1,R3] ซึ่งมีรายละเอียดของมาตรวัด ดังนี้ ค่าผลรวมค่าความซับซ้อนต่อคลาสเท่ากับ 33 หน่วย ค่าความซับซ้อนของเมคเคปของเมท็อด Charge เท่ากับ 2 หน่วย ค่าจำนวน

บรรทัดของเมท็อดเท่ากับ 10 หน่วย ค่าระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสเท่ากับ 0.6 หน่วย ค่าแอฟเฟอร์เรนคัปปลิงเท่ากับ 0 หน่วย และค่าเอฟเฟอร์เรนคัปปลิงเท่ากับ 1 หน่วย

ในส่วนอัลกอริทึมปีนเขานั้นจะสามารถปรับแก้ไขได้เพียง 1 ตำแหน่งเท่านั้น คือ D[R2] เนื่องจากการปรับแก้ไขในส่วนของการคำนวณค่า fuel หรือตำแหน่ง P1 และส่วนของการคำนวณค่า fullUsage หรือตำแหน่ง P3 ต่อจากการปรับแก้ไขในส่วนของการคำนวณค่า summerFlagtion จะได้ค่ามาตรวัดหลังจากการปรับแก้ไขด้วยวิธีรีแฟคทอริงแล้วมีค่าที่แย่กว่าการปรับแก้ไขที่ตำแหน่ง P2 เพียงตำแหน่งเดียว คือ D[R2,R1] มีค่าผลรวมค่าความซับซ้อนต่อคลาสเท่ากับ 32 หน่วย ค่าความซับซ้อนของเมคเคปของเมท็อด Charge เท่ากับ 2 หน่วย ค่าจำนวนบรรทัดของเมท็อดเท่ากับ 10 หน่วย ค่าระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสเท่ากับ 0.6 หน่วย ค่าแอฟเฟอร์เรนคัปปลิงเท่ากับ 0 หน่วย และค่าเอฟเฟอร์เรนคัปปลิงเท่ากับ 1 หน่วย ในส่วนของ D[R2,R3] มีค่าผลรวมค่าความซับซ้อนต่อคลาสเท่ากับ 33 หน่วย ค่าความซับซ้อนของเมคเคปของเมท็อด Charge เท่ากับ 2 หน่วย ค่าจำนวนบรรทัดของเมท็อดเท่ากับ 10 หน่วย ค่าระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสเท่ากับ 0.6 หน่วย ค่าแอฟเฟอร์เรนคัปปลิงเท่ากับ 0 หน่วย และค่าเอฟเฟอร์เรนคัปปลิงเท่ากับ 1 หน่วย จึงหยุดการจัดลำดับการใช้งานวิธีรีแฟคทอริงที่การแก้ไขเพียงตำแหน่งเดียว คือส่วนของการคำนวณค่า summerFlagtion ตำแหน่ง P2 เท่านั้น

ตารางที่ 37 แสดงค่ามาตรวัดของคลาส DisabilitySite ภายหลังจากปรับแก้ไขเมท็อด Charge ด้วยการจัดลำดับวิธีรีแฟคทอริงด้วยอัลกอริทึมแบบต่างๆ

อัลกอริทึม	มาตรวัด						ลำดับการใช้งานวิธีรีแฟคทอริง
	ผลรวมค่าความซับซ้อนต่อคลาส	ค่าความซับซ้อนของเมคเคปของเมท็อด Charge	จำนวนบรรทัดของเมท็อด Charge	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง	
ละโมบ	33	2	10	0.6	0	1	R2,R1,R3
ค้นหาแนวกว้าง	33	2	10	0.6	0	1	R2,R1,R3
ปีนเขา	31	2	10	0.6	0	1	R2
เอสตาร์	33	2	10	0.6	0	1	R2,R1,R3

ตารางที่ 38 แสดงจำนวนโหนดทั้งหมดที่ใช้ในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขเมท็อด Charge ของคลาส DisabilitySite โดยการใช้อัลกอริทึมแบบต่างๆ

รอบที่	อัลกอริทึม (โหนด)			
	ละโมบ	ค้นหาแนวกว้าง	ปีนเขา	เอสตาร์
1	3	3	3	6
2	2	4	2	2
3	1	1	-	1

จากตารางที่ 38 จะเห็นว่าการอัลกอริทึมปีนเขาใช้เวลาในการจัดลำดับการใช้งานวิธีรีแฟคทอริงน้อยที่สุด คือใช้เพียง 5 โหนดในการปรับแก้ไข แต่หยุดการแก้ไขหลังจากที่ปรับแก้ไขไปเพียงแค่ตำแหน่ง P2 เท่านั้น ทำให้ยังคงเหลือตำแหน่งที่ต้องปรับแก้ไขอยู่ 2 ตำแหน่ง จึงทำให้เวลาในการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบใช้เวลาน้อยสุดในการปรับแก้ไขเมท็อด Charge ของคลาส DisabilitySite ทุกตำแหน่ง คือ พิจารณา 6 โหนด ซึ่งดีกว่าการใช้อัลกอริทึมค้นหาแนวกว้างและอัลกอริทึมเอสตาร์ที่พิจารณาโหนดทั้งหมด 8 โหนดและ 9 โหนดตามลำดับ

ในส่วนของการปรับแก้ไขเมท็อด Charge ของคลาส ResidentialSite นั้น จากตารางที่ 39 อัลกอริทึมละโมบ อัลกอริทึมค้นหาแนวกว้าง และอัลกอริทึมเอสตาร์ สามารถปรับแก้ไขได้ครบทุกตำแหน่งภายในเมท็อด Charge ของคลาส ResidentialSite โดยได้ลำดับการใช้งานวิธีรีแฟคทอริง คือ R[R5,R4,R6] ซึ่งมีรายละเอียดของมาตรวัด ดังนี้ ค่าผลรวมค่าความซับซ้อนต่อคลาสเท่ากับ 33 หน่วย ค่าความซับซ้อนของเมคเคปของเมท็อด Charge เท่ากับ 2 หน่วย ค่าจำนวนบรรทัดของเมท็อดเท่ากับ 10 หน่วย ค่าระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาสเท่ากับ 0.6 หน่วย ค่าแอฟเฟอร์เรนคัปปลิงเท่ากับ 0 หน่วย และค่าเอฟเฟอร์เรนคัปปลิงเท่ากับ 1 หน่วย

ตารางที่ 39 แสดงค่ามาตรวัดของคลาส ResidentialSite ภายหลังจากปรับแก้ไขเมท็อด Charge ด้วยการจัดลำดับวิธีรีแฟคทอริงด้วยอัลกอริทึมแบบต่างๆ

อัลกอริทึม	มาตรวัด						ลำดับการใช้งานวิธีรีแฟคทอริง
	ผลรวมค่าความซับซ้อนต่อคลาส	ค่าความซับซ้อนของเมคเคปของเมท็อด Charge	จำนวนบรรทัดของเมท็อด Charge	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง	
ละโมบ	35	2	7	0.5	0	1	R5,R4,R6
ค้นหาแนวกว้าง	35	2	7	0.5	0	1	R5,R4,R6

ปีนเขา	32	2	8	0.5	0	1	R4
เอสตาร์	35	2	7	0.5	0	1	R5,R4,R6

ตารางที่ 40 แสดงจำนวนโหนดทั้งหมดที่ใช้ในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขเมท็อด Charge ของคลาส ResidentialSite โดยการใช้อัลกอริทึมแบบต่างๆ

รอบที่	อัลกอริทึม (โหนด)			
	ละโมบ	ค้นหาแนวกว้าง	ปีนเขา	เอสตาร์
1	3	3	3	6
2	2	4	2	2
3	1	1	-	1

จากตารางที่ 40 ผลลัพธ์ของจำนวนโหนดที่ใช้ในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขเมท็อด Charge ของคลาส ResidentialSite จะคล้ายกับการปรับแก้ไขเมท็อด Charge ของคลาส DisabilitySite คือ อัลกอริทึมปีนเขาใช้เวลาในการจัดลำดับการใช้งานวิธีรีแฟคทอริงน้อยที่สุด คือใช้เพียง 5 โหนดในการปรับแก้ไข แต่หยุดการแก้ไขหลังจากที่ปรับแก้ไขไปเพียงแค่ตำแหน่ง P5 เท่านั้น ทำให้ยังคงเหลือตำแหน่งที่ต้องปรับแก้ไขอยู่ 2 ตำแหน่ง จึงทำให้เวลาในการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบใช้เวลาน้อยสุดในการปรับแก้ไขเมท็อด Charge ของคลาส ResidentialSite ทุกตำแหน่ง คือ พิจารณา 6 โหนด ซึ่งดีกว่าการใช้อัลกอริทึมค้นหาแนวกว้างและอัลกอริทึมเอสตาร์ที่พิจารณาโหนดทั้งหมด 8 โหนดและ 9 โหนด ตามลำดับ

สรุปการใช้อัลกอริทึมละโมบ อัลกอริทึมค้นหาแนวกว้าง และอัลกอริทึมเอสตาร์ในการจัดลำดับการใช้งานวิธีรีแฟคทอริงแก้ไขเมท็อด Charge ของทั้งคลาส DisabilitySite และ ResidentialSite ทำให้ได้ซอร์ซโค้ดที่ขจัดร่องรอยที่ผิดพลาดได้ครบทุกตำแหน่ง คุณภาพของซอร์ซโค้ดที่ได้จึงดีกว่าการใช้อัลกอริทึมปีนเขาที่ขจัดร่องรอยที่ผิดพลาดได้ไม่ครบทุกตำแหน่ง ในส่วนของผลลัพธ์การพิจารณาเวลาที่ใช้ในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขนั้นการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบจะใช้เวลาในการปรับแก้ไขในทุกตำแหน่งโดยใช้เวลาน้อยที่สุด

สรุปเวลาที่ใช้ในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดด้วยอัลกอริทึมละโมบนั้น ทั้งลักษณะของซอร์ซโค้ดที่มีตำแหน่งหนึ่งสามารถปรับแก้ไขด้วยวิธีรีแฟคทอริงมากกว่า 1 วิธีและซอร์ซโค้ดที่มีลักษณะที่แต่ละตำแหน่งที่ต้องปรับแก้ไขสามารถใช้งานวิธีรีแฟคทอริงเพียงวิธีเดียวที่สามารถใช้ปรับแก้ไขนั้นจะใช้เวลาในการปรับแก้ไขครบทุกตำแหน่งนั้นน้อยที่สุดในบรรดาอัลกอริทึมแบบต่างๆ

4.3 ความถูกต้องของซอร์ซโค้ดภายหลังจากการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ

ในส่วนของการตรวจสอบความถูกต้องของซอร์ซโค้ดภายหลังจากการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบนั้น เพื่อพิจารณาว่าภายหลังจากการปรับแก้ไขแล้วการทำงานของซอร์ซโค้ดจะยังคงทำงานได้อย่างถูกต้องและครบถ้วนเหมือนก่อนการปรับแก้ไขและเป็นการยืนยันว่าค่าความสามารถของการบำรุงรักษาซอฟต์แวร์ที่เพิ่มขึ้นภายหลังจากการปรับแก้ไขจะไม่กระทบกับฟังก์ชันการทำงานของโค้ดส่วนที่แก้ไขไป

1. ระบบเช่าภาพยนตร์

การปรับแก้ไขเมท็อด Statement ของคลาส Customer ของระบบเช่าภาพยนตร์จากบทที่ 3 นั้น จะทำการตรวจสอบทุกฟังก์ชันการทำงานของทั้งระบบเช่าภาพยนตร์ที่ประกอบด้วยคลาส Customer คลาส Rental และคลาส Movie ว่าภายหลังการปรับแก้ไขแล้ว ทุกฟังก์ชันการทำงานยังอยู่ครบถ้วนและสามารถทำงานได้อย่างถูกต้อง รายละเอียดของฟังก์ชันการทำงานของระบบเช่าภาพยนตร์ก่อนการปรับแก้ไขแสดงได้ดังตารางที่ 41 - ตารางที่ 43 ดังนี้

จากตารางที่ 41 จะเห็นได้ว่าการคำนวณค่าต่างๆ ที่เกี่ยวข้องกับการเช่าภาพยนตร์ของลูกค้านั้นไม่ว่าจะเป็นค่าเช่าภาพยนตร์แต่ละเรื่อง ค่าความถี่สะสมในการเช่าภาพยนตร์ ค่าเช่าภาพยนตร์รวมในการเช่าภาพยนตร์ในแต่ละครั้ง อยู่ภายในเมท็อด Statement ของคลาส Customer ทั้งสิ้น และภายหลังจากการคำนวณค่าต่างๆ ต้องแสดงผลลัพธ์รายละเอียดการเช่าภาพยนตร์ของลูกค้าผ่านทางหน้าจอ จึงทำให้เมท็อด Statement มีการทำงานที่มากเกินกว่าเมท็อดหนึ่งๆ ควรรับผิดชอบจึงเป็นสาเหตุทำให้เกิดร่องรอยที่ผิดพลาดขึ้นในเมท็อด ในส่วนของคลาส Rental และ คลาส Movie มีเพียงแค่เมท็อดที่ทำหน้าที่เก็บและส่งคืนค่ารายละเอียดต่างๆ คือ รายละเอียดการเช่าภาพยนตร์และรายละเอียดของภาพยนตร์ตามลำดับ ดังนั้นจึงต้องทำการปรับแก้ไขเมท็อด Statement ใหม่ด้วยวิธีรีแฟคทอริงเพื่อกำจัดร่องรอยที่ผิดพลาดและแบ่งเบาภาระการทำงานให้กับเมท็อด

ตารางที่ 41 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส Customer ก่อนการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง

เมท็อด	ข้อมูลขาเข้า	ข้อมูลขาออก	รายละเอียด
Statement	-	String	- คำนวนค่าเช่าภาพยนตร์แต่ละเรื่อง - คำนวณค่าความถี่สะสมในการเช่าภาพยนตร์ - คำนวนค่าเช่าภาพยนตร์รวมในการเช่าภาพยนตร์แต่ละครั้ง - แสดงผลรายละเอียดการเช่าของ

			ลูกค้าออกทางหน้าจอ
addRental	Rental	-	เพิ่มจำนวนภาพยนตร์ที่ลูกค้ายืม
getName	-	String	ส่งคืนค่าชื่อลูกค้า

ตารางที่ 42 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส Rental ก่อนการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง

เมท็อด	ข้อมูลขาเข้า	ข้อมูลขาออก	รายละเอียด
getDayRented	-	int	ส่งคืนค่าจำนวนวันที่เช่า
getMovie	-	Movie	ส่งคืนอ็อบเจ็คภาพยนตร์ที่ยืม

ตารางที่ 43 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส Movie ก่อนการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง

เมท็อด	ข้อมูลขาเข้า	ข้อมูลขาออก	รายละเอียด
getPriceCode	-	int	ส่งคืนราคาค่าเช่าของภาพยนตร์
getTitle	-	String	ส่งคืนค่าชื่อของภาพยนตร์

โดยรายละเอียดของฟังก์ชันการทำงานของระบบเช่าภาพยนตร์ภายหลังการปรับแก้ไขแสดงได้ดังตารางที่ 44 - ตารางที่ 46 ดังนี้

ตารางที่ 44 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส Customer หลังการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง

เมท็อด	ข้อมูลขาเข้า	ข้อมูลขาออก	รายละเอียด
Statement	-	String	แสดงผลรายละเอียดการเช่าของลูกค้าออกทางหน้าจอ
addRental	Rental	-	เพิ่มจำนวนภาพยนตร์ที่ลูกค้ายืม
getName	-	String	ส่งคืนค่าชื่อลูกค้า
getTotalCharge	-	double	รวมผลค่าเช่าภาพยนตร์ของลูกค้า
getTotalFrequent RenterPoints	-	int	รวมผลค่าความถี่สะสมในการเช่าภาพยนตร์ของลูกค้า

ตารางที่ 45 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส Rental หลังการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง

เมท็อด	ข้อมูลขาเข้า	ข้อมูลขาออก	รายละเอียด
getDayRented	-	int	ส่งคืนค่าจำนวนวันที่เช่า
getMovie	-	Movie	ส่งคืนอ็อบเจ็คภาพยนตร์ที่ยืม
getCharge	-	double	คำนวนค่าเช่าภาพยนตร์แต่ละเรื่อง
getFrequentRenter Pointers	-	int	คำนวณค่าความถี่สะสมในการเช่าภาพยนตร์

ตารางที่ 46 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส Movie หลังการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง

เมท็อด	ข้อมูลขาเข้า	ข้อมูลขาออก	รายละเอียด
getPriceCode	-	int	ส่งคืนราคาค่าเช่าของภาพยนตร์
getTitle	-	String	ส่งคืนค่าชื่อของภาพยนตร์

จากตารางที่ 44 จะเห็นได้ว่าภายหลังจากการปรับแก้ไขเมท็อด Statement ด้วยวิธีรีแฟคทอริงแล้วนั้นการทำงานของเมท็อดลดลงเหลือแค่การทำงานในส่วนของแสดงผลรายละเอียดการเช่าของลูกค้าออกทางหน้าจอเท่านั้น ในส่วนของการคำนวณค่าเช่าภาพยนตร์แต่ละเรื่องค่าความถี่สะสมในการเช่าภาพยนตร์นั้นจะย้ายการทำงานไปอยู่ภายใต้คลาส Rental คือ เมท็อด getCharge และเมท็อด getFrequentRenterPointers ตามลำดับแทน เพื่อให้คลาส Customer เรียกใช้งานในการแสดงรายละเอียดการเช่าภาพยนตร์ของลูกค้าผ่านทางเมท็อด Statement เมท็อด getTotalCharge และเมท็อด getTotalFrequentRenterPointers การแยกการทำงานออกเป็นเมท็อดหลายๆ เมท็อดแบ่งตามหน้าที่เป็นสำคัญจะทำให้การแก้ไขโค้ดนั้นทำได้ง่าย การนำโค้ดกลับไปใช้งานทำได้ง่ายขึ้น

กรณีทดสอบ (Test Case) ที่ใช้ในการตรวจสอบความถูกต้องของซอร์ซโค้ดภายหลังจากการปรับแก้ไขด้วยวิธีรีแฟคทอริงโดยใช้อัลกอริทึมละโมบนั้นจะทดสอบแยกออกตามฟังก์ชันหลักได้แก่

1. แสดงรายละเอียดของภาพยนตร์ที่ลูกค้าเช่า เช่น ชื่อภาพยนตร์ ราคาค่าเช่า
2. คำนวณค่าเช่าภาพยนตร์แต่ละเรื่อง
3. คำนวณค่าความถี่สะสมในการเช่าภาพยนตร์แต่ละเรื่อง
4. คำนวณผลรวมค่าเช่าภาพยนตร์ของลูกค้าในการเช่าครั้งหนึ่งๆ
5. คำนวณผลรวมค่าความถี่สะสมในการเช่าภาพยนตร์ของลูกค้าในการเช่าครั้งหนึ่งๆ

โดยรายละเอียดของกรณีทดสอบและผลลัพธ์ของการทดสอบแสดงแยกตามฟังก์ชันงานหลัก ได้ดังตารางที่ 47 - ตารางที่ 51

ตารางที่ 47 แสดงรายละเอียดกรณีทดสอบส่วนของการแสดงรายละเอียดของภาพยนตร์ที่ลูกค้าเช่า

รหัสกรณีทดสอบ	ข้อมูลนำเข้า (เรื่อง,ประเภทหนัง,จำนวนวันที่เช่า)	ผลลัพธ์ก่อนปรับแก้ไข	ผลลัพธ์ภายหลังปรับแก้ไข
1_1	- “Iron Man 3”, “0”, “3” - “Thor 2”, “0”, “4” - “Captain America”, “1”, “1”	“Customer : Ratapong W. No. Name Type DayRented 1 Iron Man 3 0 3 2 Thor 2 0 4 3 Captain America 1 2”	“Customer : Ratapong W. No. Name Type DayRented 1 Iron Man 3 0 3 2 Thor 2 0 4 3 Captain America 1 2”

ตารางที่ 48 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณค่าเช่าภาพยนตร์แต่ละเรื่อง

รหัสกรณีทดสอบ	ข้อมูลนำเข้า (เรื่อง,ประเภทหนัง,จำนวนวันที่เช่า)	ผลลัพธ์ก่อนปรับแก้ไข	ผลลัพธ์ภายหลังปรับแก้ไข
2_1	- “Iron Man 3”, “0”, “3” - “Thor 2”, “0”, “4” - “Captain America”, “1”, “2”	Iron Man 3 : 9.0 Thor 2 : 12.0 Captain America : 2.0	Iron Man 3 : 9.0 Thor 2 : 12.0 Captain America : 2.0

ตารางที่ 49 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณค่าความถี่สะสมในการเช่าภาพยนตร์แต่ละเรื่อง

รหัสกรณีทดสอบ	ข้อมูลนำเข้า (เรื่อง,ประเภทหนัง,จำนวนวันที่เช่า)	ผลลัพธ์ก่อนปรับแก้ไข	ผลลัพธ์ภายหลังปรับแก้ไข
3_1	- “Iron Man 3”, “0”, “3” - “Thor 2”, “0”, “4” - “Captain America”, “1”, “2”	- Iron Man 3 : 2 - Thor 2 : 2 - Captain America : 1	- Iron Man 3 : 2 - Thor 2 : 2 - Captain America : 1

ตารางที่ 50 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณผลรวมค่าเช่าภาพยนตร์ของลูกค้าในการเช่าครั้งหนึ่งๆ

รหัสกรณีทดสอบ	ข้อมูลนำเข้า (เรื่อง,ประเภทหนัง,จำนวนวันที่เช่า)	ผลลัพธ์ก่อนปรับแก้ไข	ผลลัพธ์ภายหลังปรับแก้ไข
4_1	- “Iron Man 3”, “0”, “3” - “Thor 2”, “0”, “4” - “Captain America”, “1”, “2”	Amount owed : 23	Amount owed : 23

ตารางที่ 51 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณผลรวมค่าความถี่สะสมในการเช่าภาพยนตร์ของลูกค้าในการเช่าครั้งหนึ่งๆ

รหัสกรณีทดสอบ	ข้อมูลนำเข้า (เรื่อง,ประเภทหนัง,จำนวนวันที่เช่า)	ผลลัพธ์ก่อนปรับแก้ไข	ผลลัพธ์ภายหลังปรับแก้ไข
5_1	- “Iron Man 3”, “0”, “3” - “Thor 2”, “0”, “4” - “Captain America”, “1”, “2”	Frequent renter points : 5	Frequent renter points : 5

จากตารางที่ 47 - ตารางที่ 51 กรณีทดสอบทั้งหมด 5 ฟังก์ชันการทำงาน เมื่อทดสอบกับข้อมูลค่าเข้าเป็นภาพยนตร์จำนวนทั้งหมด 3 เรื่องที่มีรายละเอียดดังนี้

1. เรื่อง Iron Man ประเภทหนัง 0 (เข้าใหม่) จำนวนวันที่เช่า 3 วัน
2. เรื่อง Thor 2 ประเภทหนัง 0 (เข้าใหม่) จำนวนวันที่เช่า 4 วัน
3. เรื่อง Captain America ประเภทหนัง 1 (หนังทั่วไป) จำนวนวันที่เช่า 2 วัน

กับซอร์ซโค้ดก่อนและหลังการแก้ไขด้วยวิธีรีแฟคทอริง ทุกกรณีทดสอบจะได้ผลลัพธ์ที่เท่ากัน จึงสรุปได้ว่าการแก้ไขซอร์ซโค้ดเพื่อกำจัดร่องรอยที่ผิดพลาดในเมท็อด Statement ของคลาส Customer ด้วยวิธีรีแฟคทอริง ทำให้โครงสร้างของเมท็อดนั้นเปลี่ยนแปลงไป แต่ยังคงทำงานได้เหมือนกันก่อนการแก้ไข

2. ระบบคำนวณค่าไฟฟ้า

การปรับแก้ไขเมท็อด Charge ของทั้งคลาส DisabilitySite และ Residential-Site นั้น จะทำการตรวจสอบแค่เมท็อด Charge เท่านั้น เนื่องวิธีรีแฟคทอริงที่ใช้แก้ไขนั้นมีการเปลี่ยนแปลงแค่ภายในคลาส Charge ไม่กระทบกับการทำงานของคลาสอื่นๆ ว่าภายหลังการปรับแก้ไขแล้วยังทำงานได้อย่างถูกต้องเหมือนก่อนการปรับแก้ไข ซึ่งรายละเอียดของการทำงานของคลาส DisabilitySite และ ResidentialSite ก่อนการปรับแก้ไขแสดงได้ดังตารางที่ 52 - ตารางที่ 55 ดังนี้

ตารางที่ 52 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส DisabilitySite ก่อนการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง

เมท็อด	ข้อมูลขาเข้า	ข้อมูลขาออก	รายละเอียด
charge	fullUsage start end	Dollars	- ตรวจสอบรายการว่าเป็นช่วงหน้าร้อนหรือไม่ - คำนวณค่าไฟตามอัตรา - คำนวนค่าไฟฟ้ารวมของรายการ
addReading	Reading	-	เพิ่มข้อมูลมาตรวัดการใช้ไฟฟ้า
dayOfYear	Date	int	ส่งคืนค่าวันในรอบปี

ตารางที่ 53 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส DisabilitySite หลังการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง

เมท็อด	ข้อมูลขาเข้า	ข้อมูลขาออก	รายละเอียด
charge	fullUsage start end	Dollars	คำนวนค่าไฟฟ้าของรายการ
addReading	Reading	-	เพิ่มข้อมูลมาตรวัดการใช้ไฟฟ้า
dayOfYear	Date	int	ส่งคืนค่าวันในรอบปี
lastUsage	-	int	คำนวณจำนวนค่ามาตรวัดไฟฟ้าของการใช้งานรายการล่าสุด
summerFraction	start	end	คำนวณค่าอัตราช่วงหน้าร้อน
fuelCharge	fullUsage	Dollars	คำนวณค่าไฟตามอัตรา

จากตารางที่ 53 และ ตารางที่ 55 จะเห็นได้ว่าภายหลังจากการปรับแก้ไขเมท็อด Charge ของทั้งคลาส DisabilitySite และคลาส ResidentialSite ด้วยวิธีรีแฟคทอริงดังตารางที่ 53 และตารางที่ 55นั้น เป็นการลดจำนวนบรรทัดของเมท็อด Charge ลงโดยการย้ายซอร์ซโค้ดการทำงานภายในแล้วสร้างเป็นเมท็อดใหม่เพื่อเรียกใช้งานแทน เช่น เมท็อด lastUsage เมท็อด summerFraction และ fuelCharge ทำให้จำนวนบรรทัดของเมท็อด Charge นั้นสั้นลง การบำรุงซอฟต์แวร์จึงทำได้ง่ายขึ้น

ตารางที่ 54 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส ResdentialSite ก่อนการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง

เมท็อด	ข้อมูลขาเข้า	ข้อมูลขาออก	รายละเอียด
charge	fullUsage start end	Dollars	- ตรวจสอบรายการว่าเป็นช่วงหน้าร้อนหรือไม่ - คำนวณค่าไฟตามอัตรา - คำนวนค่าไฟฟ้ารวมของรายการ
addReading	Reading	-	เพิ่มข้อมูลมาตรวัดการใช้ไฟฟ้า
dayOfYear	Date	int	ส่งคืนค่าวันในรอบปี

ตารางที่ 55 แสดงรายละเอียดฟังก์ชันการทำงานของคลาส ResidentialSite หลังการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง

เมท็อด	ข้อมูลขาเข้า	ข้อมูลขาออก	รายละเอียด
charge	fullUsage start end	Dollars	คำนวนค่าไฟฟ้าของรายการ
addReading	Reading	-	เพิ่มข้อมูลมาตรวัดการใช้ไฟฟ้า
dayOfYear	Date	int	ส่งคืนค่าวันในรอบปี
lastUsage	-	int	คำนวณจำนวนค่ามาตรวัดไฟฟ้าของการใช้งานรายการล่าสุด
summerFraction	start	end	คำนวณค่าอัตราช่วงหน้าร้อน
fuelCharge	fullUsage	Dollars	คำนวณค่าไฟตามอัตรา

สำหรับกรณีทดสอบที่ใช้ในการตรวจสอบความถูกต้องของซอร์ซโค้ดภายหลังจากการปรับแก้ไขด้วยวิธีรีแฟคทอริงโดยใช้อัลกอริทึมละโมบของคลาส DisabilitySite และ คลาส ResidentialSite นั้นจะทดสอบแยกออกตามฟังก์ชันหลักได้แก่

1. คำนวณค่าไฟฟ้าของรายการ
2. คำนวณจำนวนค่ามาตรวัดไฟฟ้าของการใช้งานรายการล่าสุด
3. คำนวณค่าอัตราช่วงหน้าร้อน

โดยรายละเอียดของกรณีทดสอบและผลลัพธ์ของการทดสอบแสดงแยกตามฟังก์ชันงานหลักได้ดังตารางที่ 56 - ตารางที่ 61

ตารางที่ 56 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณค่าไฟฟ้าของรายการของคลาส DisabilitySite

รหัสกรณีทดสอบ	ข้อมูลนำเข้า (จำนวนหน่วยที่ใช้, วันที่บันทึกมาตรวัด)	ผลลัพธ์ก่อนปรับแก้ไข	ผลลัพธ์ภายหลังปรับแก้ไข
1_1	-“50, 15/03/2014” - “120, 15/04/2014”	“Total Charge : 370.01”	“Total Charge : 370.01”

ตารางที่ 57 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณค่ามาตรวัดไฟฟ้าของการใช้งานรายการล่าสุดของคลาส DisabilitySite

รหัสกรณีทดสอบ	ข้อมูลนำเข้า (จำนวนหน่วยที่ใช้, วันที่บันทึกมาตรวัด)	ผลลัพธ์ก่อนปรับแก้ไข	ผลลัพธ์ภายหลังปรับแก้ไข
2_1	-“50, 15/03/2014” - “120, 15/04/2014”	“Last Usage : 70”	“Last Usage : 70”

ตารางที่ 58 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณค่าอัตราช่วงหน้าร้อนของการใช้งานรายการล่าสุดของคลาส DisabilitySite

รหัสกรณีทดสอบ	ข้อมูลนำเข้า (จำนวนหน่วยที่ใช้, วันที่บันทึกมาตรวัด)	ผลลัพธ์ก่อนปรับแก้ไข	ผลลัพธ์ภายหลังปรับแก้ไข
3_1	-“50, 15/03/2014” - “120, 15/04/2014”	“Summer Fraction : 1”	“Summer Fraction : 1”

ตารางที่ 59 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณค่าไฟฟ้าของรายการของคลาส ResidentialSite

รหัสกรณีทดสอบ	ข้อมูลนำเข้า (จำนวนหน่วยที่ใช้, วันที่บันทึกมาตรวัด)	ผลลัพธ์ก่อนปรับแก้ไข	ผลลัพธ์ภายหลังปรับแก้ไข
4_1	-“50, 15/03/2014” - “120, 15/04/2014”	“Total Charge : 517.01”	“Total Charge : 517.01”

ตารางที่ 60 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณค่ามาตรวัดไฟฟ้า ของการใช้งานรายการล่าสุดของคลาส ResidentialSite

รหัสกรณีทดสอบ	ข้อมูลนำเข้า (จำนวนหน่วยที่ใช้, วันที่บันทึกมาตรวัด)	ผลลัพธ์ก่อนปรับแก้ไข	ผลลัพธ์ภายหลังปรับแก้ไข
5_1	-“50, 15/03/2014” - “120, 15/04/2014”	“Last Usage : 70”	“Last Usage : 70”

ตารางที่ 61 แสดงรายละเอียดกรณีทดสอบส่วนของการคำนวณค่าอัตราช่วงหน้าร้อน ของการใช้งานรายการล่าสุดของคลาส ResidentialSite

รหัสกรณีทดสอบ	ข้อมูลนำเข้า (จำนวนหน่วยที่ใช้, วันที่บันทึกมาตรวัด)	ผลลัพธ์ก่อนปรับแก้ไข	ผลลัพธ์ภายหลังปรับแก้ไข
6_1	-“50, 15/03/2014” - “120, 15/04/2014”	“Summer Fraction : 1”	“Summer Fraction : 1”

จากตารางที่ 56 - ตารางที่ 61 กรณีทดสอบทั้งหมด 6 ฟังก์ชันการทำงาน แยกเป็นคลาส DisabilitySite และ ResidentialSite อย่างละ 3 ฟังก์ชัน เมื่อทดสอบกับข้อมูลค่าเข้าเป็นรายการบันทึกการใช้ไฟฟ้าที่มีรายละเอียดดังนี้

1. วันที่บันทึก 15/03/2014 จำนวนหน่วยที่วัดได้ 50 หน่วย
2. วันที่บันทึก 15/04/2014 จำนวนหน่วยที่วัดได้ 120 หน่วย

กับซอร์ซโค้ดก่อนและหลังการแก้ไขด้วยวิธีรีแฟคทอริง ทุกกรณีทดสอบจะได้ผลลัพธ์ที่เท่ากันจึงสรุปได้ว่าการแก้ไขซอร์ซโค้ดเพื่อกำจัดร่องรอยที่ผิดพลาดในเมท็อด Charge ของคลาส DisabilitySite และคลาส ResidentialSite ด้วยวิธีรีแฟคทอริง ทำให้โครงสร้างของเมท็อดนั้นเปลี่ยนแปลงไป แต่ยังคงทำงานได้เหมือนกันก่อนการแก้ไข

สรุปความถูกต้องของซอร์ซโค้ดภายหลังจากการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบนั้น ภายหลังจากการปรับแก้ไขการทำงานของระบบก็ยังคงทำงานเหมือนก่อนการปรับแก้ไข

บทที่ 5
การออกแบบและพัฒนาเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริงโดยใช้มาตรวัดเชิงวัตถุและอัลกอริทึมละโมบ

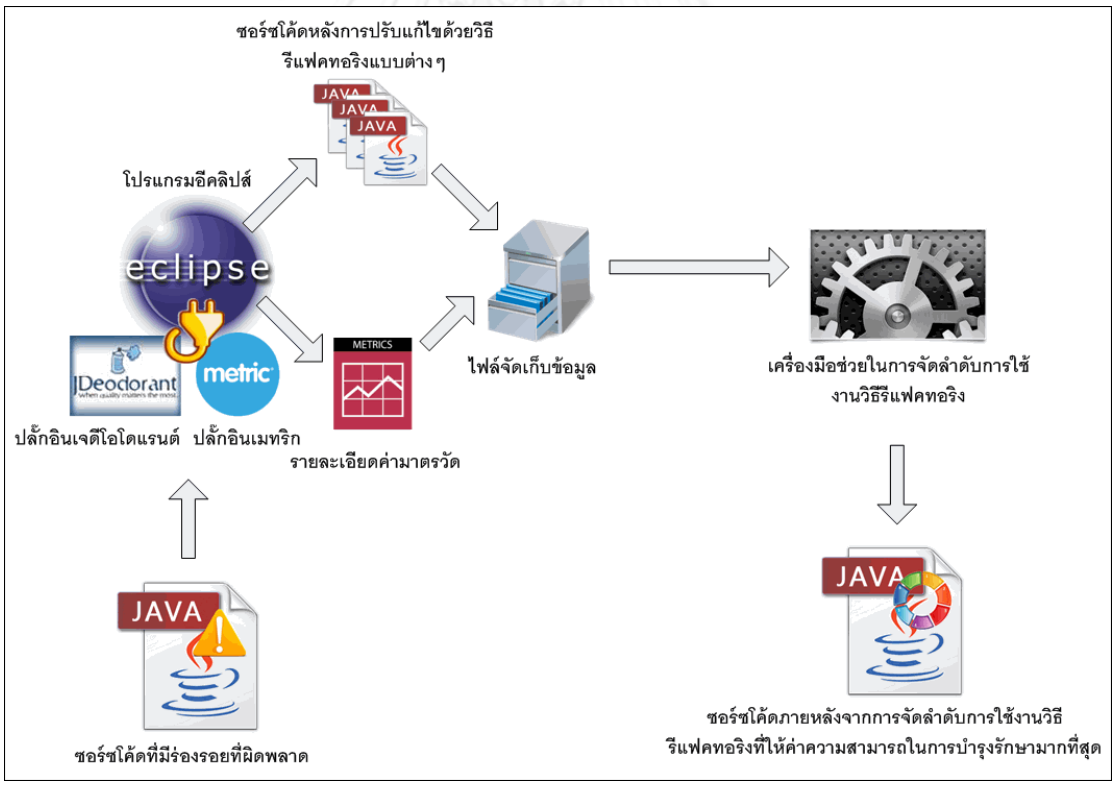
ในบทนี้จะอธิบายรายละเอียดเกี่ยวกับขั้นตอนการออกแบบและพัฒนาเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริงโดยใช้มาตรวัดเชิงวัตถุและอัลกอริทึมละโมบ โดยแบ่งการอธิบายออกเป็น 2 หัวข้อหลัก คือ

1. ส่วนของการออกแบบเครื่องมือ
2. ส่วนของรายละเอียดของเครื่องมือ

รายละเอียดของขั้นตอนการออกแบบและพัฒนาเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริงโดยใช้มาตรวัดเชิงวัตถุและอัลกอริทึมละโมบนั้นมีรายละเอียดดังนี้

5.1 ส่วนของการออกแบบเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริง

5.1.1 สถาปัตยกรรมของเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริง



ภาพที่ 30 สถาปัตยกรรมเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริง

จากภาพที่ 30 สถาปัตยกรรมของเครื่องมือประกอบด้วยส่วนงานหลักทั้งหมด 3 ส่วนด้วยกัน คือ ส่วนการทำงานของโปรแกรมอีคลิปส์ ส่วนของการจัดเก็บข้อมูล และส่วนของการจัดลำดับการใช้งานวิธีรีแฟคทอริง โดยมีข้อมูลนำเข้าสำหรับการใช้งานเป็นซอร์ซโค้ดที่มีร่องรอยที่

ผิดพลาดให้ปรับแก้ไขหลายตำแหน่ง เมื่อสิ้นสุดกระบวนการปรับแก้ไขซอร์ซโค้ดโดยการจัดลำดับการใช้งานวิธีรีแฟคทอริงแล้วจะได้ซอร์ซโค้ดที่มีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ที่ดีที่สุด รายละเอียดของส่วนงานหลักแต่ละส่วนมีรายละเอียด ดังนี้

1. ส่วนของการทำงานของโปรแกรมอีคลิปส์

เป็นโปรแกรมอีคลิปส์ที่เสริมการทำงานเพิ่มเติมด้วยการนำเอาปลั๊กอินเจดีโอโดแรนต์เพื่อช่วยในการค้นหาร่องรอยที่ผิดพลาดที่อยู่ซอร์ซโค้ดและวิธีรีแฟคทอริงในการขจัดร่องรอยที่ผิดพลาดนั้นๆ และปลั๊กอินเมทริกสำหรับคำนวณหาค่ามาตรวัดเชิงวัตถุต่างๆ ของซอร์ซโค้ด รายละเอียดของโปรแกรมอีคลิปส์และปลั๊กอินทั้ง 2 มีรายละเอียดดังต่อไปนี้

1.1 โปรแกรมอีคลิปส์

- ชื่อเต็ม : Eclipse Java EE IDE for Web Developers 3.7
- เวอร์ชัน : Indigo Service Release 2
- ระบบปฏิบัติการ : Window 64bit

1.2 ปลั๊กอินเจดีโอโดแรนต์

- ชื่อเต็ม : JDeodorant
- เวอร์ชัน : 6.1
- รองรับการทำงานร่วมกับ : Eclipse 3.7.x, 3.8.x, 4.2.x

1.3 ปลั๊กอินเมทริก

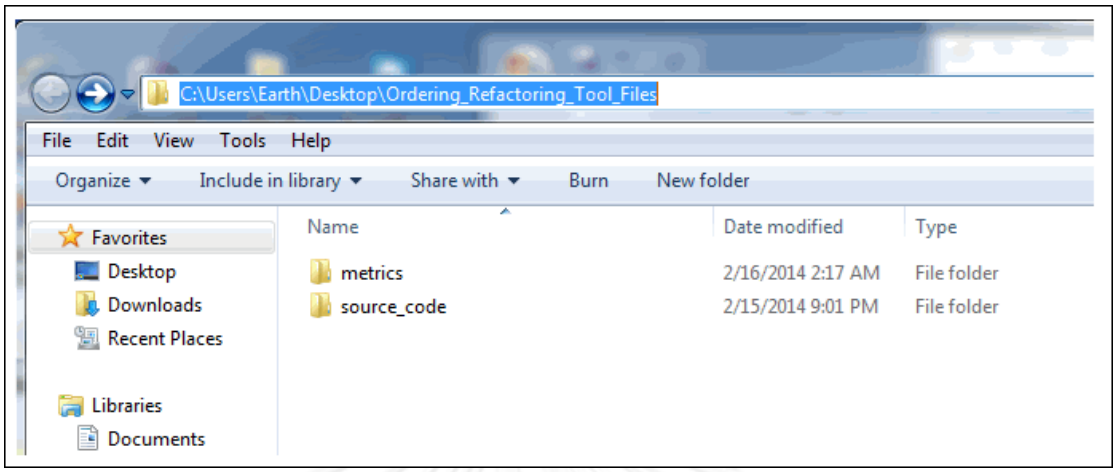
- ชื่อเต็ม : Metrics
- เวอร์ชัน : 1.3.6
- รองรับการทำงานร่วมกับ : Eclipse 3.1 เป็นต้นไป

การทำงานนั้นจะเริ่มจากนำซอร์ซโค้ดที่มีที่ร่องรอยที่ผิดพลาดมาทำการค้นหาตำแหน่งของที่ต้องปรับแก้ไขเพื่อกำจัดร่องรอยที่ผิดพลาดนั้นๆ โดยการใช้การทำงานของปลั๊กอินเจดีโอโดแรนต์ในการค้นหา เมื่อได้ตำแหน่งของร่องรอยที่ผิดพลาดและวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขแล้ว นำมาปรับแก้ไขโค้ดให้เป็นไปตามแนวทางของการปรับแก้ไขด้วยวิธีรีแฟคทอริงที่เป็นไปได้ทั้งหมด จะได้ซอร์ซโค้ดของรูปแบบทั้งหมดในการปรับแก้ไข จากนั้นจึงใช้การงานของปลั๊กอินเมทริกในการคำนวณหาค่ามาตรวัดต่างๆ ของซอร์ซโค้ดแต่ละแบบ จากนั้นจึงนำซอร์ซโค้ดทุกแบบพร้อมกับค่ามาตรวัดส่งไปยังส่วนของการจัดเก็บข้อมูล เพื่อจัดเก็บข้อมูลต่อไป

2. ส่วนของการจัดเก็บข้อมูล

เป็นส่วนที่จัดเก็บข้อมูลซอร์ซโค้ดภายหลังจากการปรับแก้ไขด้วยวิธีรีแฟคทอริงและข้อมูลรายละเอียดของค่ามาตรวัดต่างๆ ของซอร์ซโค้ด เพื่อนำข้อมูลดังกล่าวนั้นมาใช้ในส่วนการจัดลำดับการใช้งานวิธีรีแฟคทอริง โดยจัดเก็บข้อมูลในรูปแบบของไฟล์ (File Processing) แยกเป็น 2

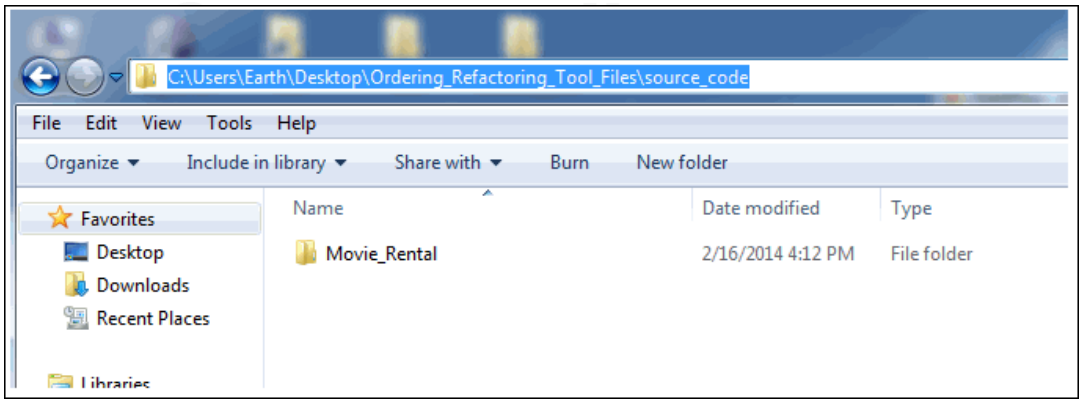
โฟลเดอร์หลัก คือ โฟลเดอร์ source_code และ โฟลเดอร์ metrics แสดงดังภาพที่ 31 รายละเอียดของแต่ละโฟลเดอร์และรูปแบบไฟล์ที่จัดเก็บข้อมูลมีดังนี้



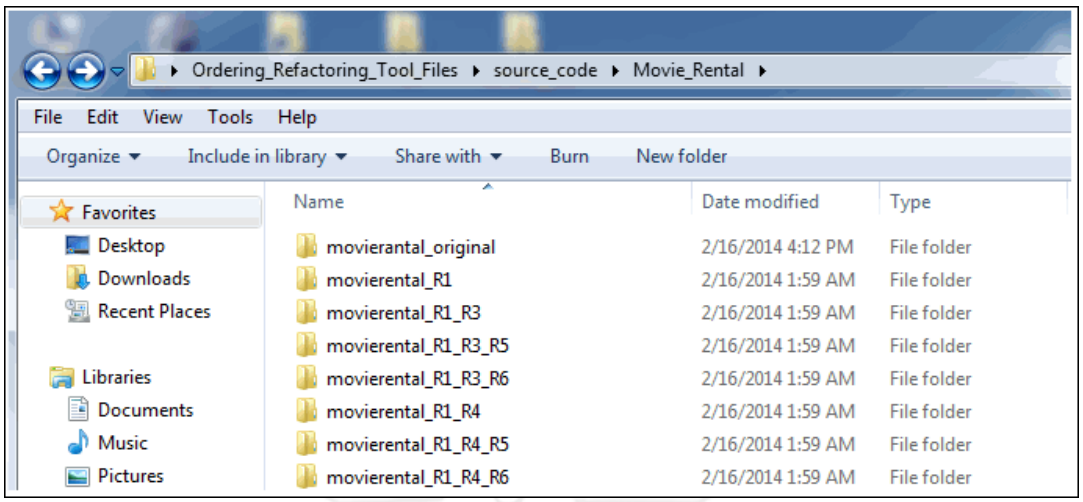
ภาพที่ 31 โฟลเดอร์หลักในการจัดเก็บข้อมูลของเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริง

2.1 โฟลเดอร์ source_code

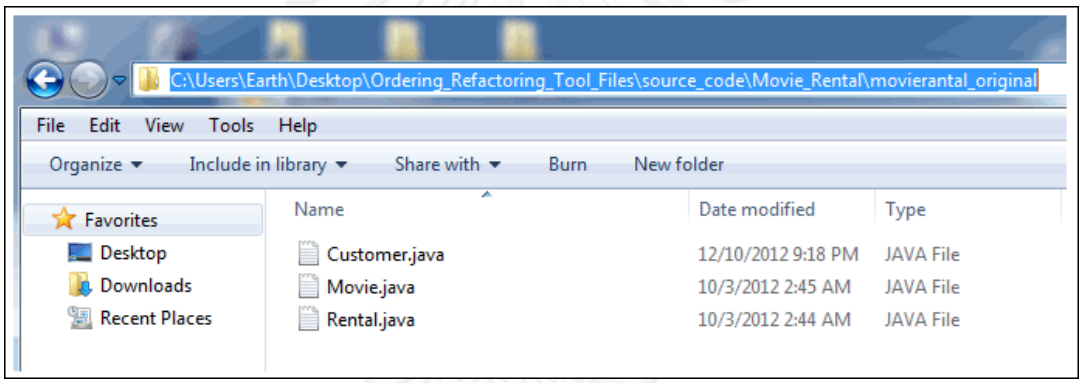
เป็นโฟลเดอร์สำหรับจัดเก็บซอร์ซโค้ดต้นแบบก่อนการปรับแก้ไขและซอร์ซโค้ดภายหลังจากการปรับแก้ไขด้วยวิธีรีแฟคทอริงที่ได้จากส่วนของการทำงานของโปรแกรมอีคลิปส์ ภายในโฟลเดอร์จะจัดเก็บซอร์ซโค้ดแยกตามชื่อระบบ โดยวิทยานิพนธ์นี้จะได้ระบบเช่าภาพยนตร์เป็นข้อมูลนำเข้าในการทดลอง และตั้งชื่อโฟลเดอร์ของระบบเป็น Movie_Rental แสดงได้ดังภาพที่ 32 ซึ่งภายในโฟลเดอร์ของระบบจะจัดเก็บโฟลเดอร์แยกตามการปรับแก้ไขด้วยวิธีรีแฟคทอริง <ชื่อระบบ>_original คือโฟลเดอร์ที่จัดเก็บซอร์ซโค้ดของระบบเช่าภาพยนตร์ก่อนการปรับแก้ไขในส่วนของซอร์ซโค้ดที่ถูกปรับแก้ไขด้วยวิธีรีแฟคทอริงจะแยกเก็บในรูปแบบของ “<ชื่อระบบ>_<วิธีรีแฟคทอริง>_<วิธีรีแฟคทอริง>” โดยที่ภายในจัดเก็บไฟล์ซอร์ซโค้ดของภาษาจาวาไว้โดยแยกตามการปรับแก้ไขด้วยวิธีรีแฟคทอริงแต่ละวิธีแสดงได้ดังภาพที่ 33



ภาพที่ 32 โฟลเดอร์ระบบเช่าภาพยนตร์ที่จัดเก็บซอร์ซโค้ดของระบบ



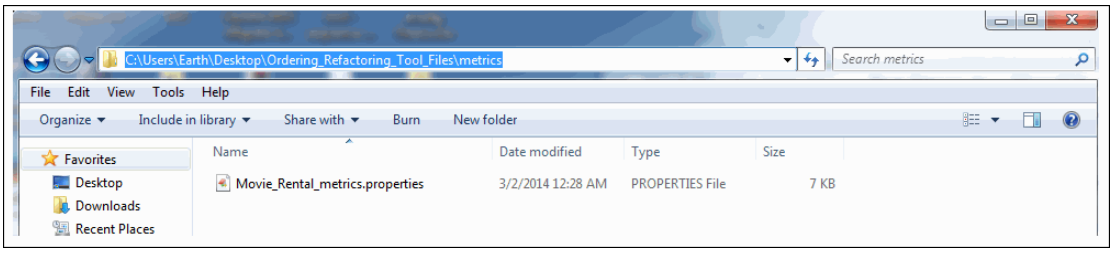
ภาพที่ 33 โฟลเดอร์ระบบเช่าภาพยนตร์แยกตามการปรับแก้ไขด้วยวิธีรีแฟคทอริงแต่ละวิธี



ภาพที่ 34 ไฟล์ซอร์ซโค้ดของระบบเช่าภาพยนตร์ที่จัดเก็บภายในโฟลเดอร์ movierental_original

2.2 โฟลเดอร์ metrics

เป็นโฟลเดอร์ที่จัดเก็บไฟล์รายละเอียดค่ามาตรวัดของซอร์ซโค้ดภายหลังจากการปรับแก้ไขด้วยวิธีรีแฟคทอริง ซึ่งไฟล์รายละเอียดนั้นจะแยกตามชื่อระบบตามรูปแบบ “<ชื่อระบบ>_metrics” เป็นสกุล properties ซึ่งตามภาพที่ 35 นั้นเป็นไฟล์รายละเอียดค่ามาตรวัดของซอร์ซโค้ดระบบเช่าภาพยนตร์ ภายในไฟล์จะแสดงรายละเอียดของมาตรวัดของซอร์ซโค้ดภายหลังจากการปรับแก้ไข โดยจัดเก็บในรูปแบบของ “<ชื่อคลาส>_<วิธีรีแฟคทอริง>=<ค่าผลรวมค่าความซับซ้อนต่อคลาส>|<ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อด>|<แอฟเฟอร์เรน คลับพลิง>|<เอฟเฟอร์เรน คลับพลิง>” แสดงได้ดังภาพที่ 36



ภาพที่ 35 ไฟล์รายละเอียดค่ามาตรวัดของซอร์ซโค้ดระบบเช่าภาพยนตร์ภายหลังจากการปรับแก้ไขด้วยวิธีรีแฟคทอริง

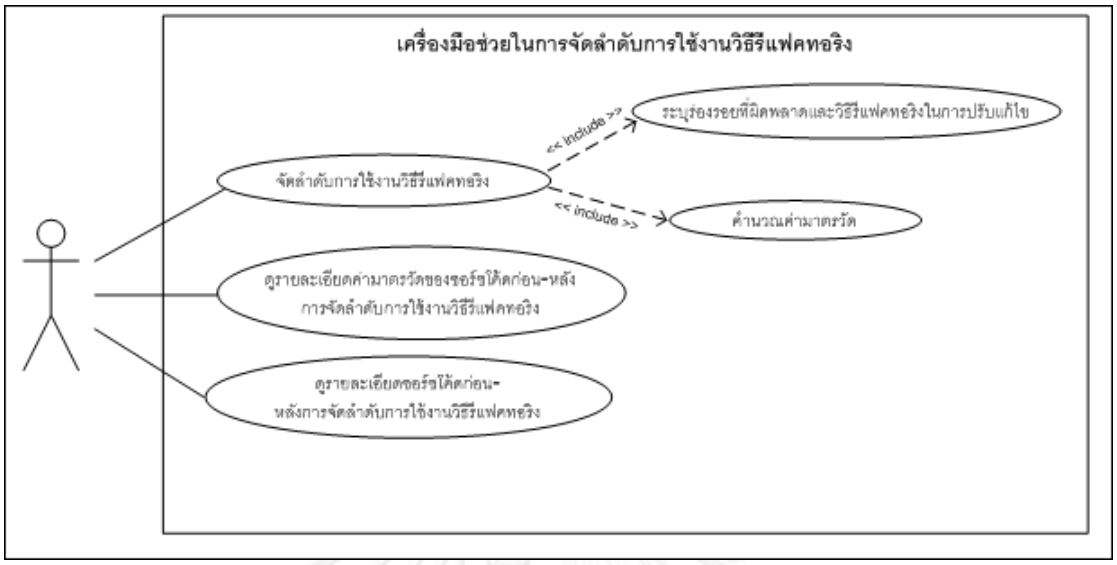
```
Customer_R1=13|0.667|0|1
Movie_R1=4|0.5|2|0
Rental_R1=3|0.5|1|1
Average_R1=6.67|0.556|1|0.67

Customer_R1_R3=15|0.625|0|1
Movie_R1_R3=4|0.5|2|0
Rental_R1_R3=3|0.5|1|1
Average_R1_R3=7.33|0.54|1|0.67
```

ภาพที่ 36 ตัวอย่างรายละเอียดการจัดเก็บค่ามาตรวัดของซอร์ซโค้ดระบบเช่าภาพยนตร์

3. ส่วนของการจัดลำดับการใช้งานวิธีรีแฟคทอริง

เป็นส่วนที่ใช้จัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด โดยการนำเอาข้อมูลซอร์ซโค้ดภายหลังจากการปรับแก้ไขและค่ามาตรวัดของซอร์ซโค้ดแบบต่างๆ มาใช้เป็นข้อมูลในการค้นลำดับการใช้งานวิธีรีแฟคทอริง โดยการใช้อัลกอริทึมละโมบในการค้นหาลำดับการใช้งานวิธีรีแฟคทอริง ซึ่งข้อมูลตั้งต้นสำหรับใช้ในการจัดลำดับการใช้งานวิธีรีแฟคทอริง ได้แก่ รายชื่อโปรเจคข้อมูลนำเข้า ที่อยู่ที่จัดเก็บรายละเอียดของซอร์ซโค้ด ที่อยู่ที่จัดเก็บไฟล์รายละเอียดค่ามาตรวัดของซอร์ซโค้ด รายละเอียดความสัมพันธ์ของตำแหน่งที่ต้องปรับแก้ไขและวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขนั้นจะถูกจัดเก็บที่ไฟล์ที่ชื่อว่า orderingRefactoringTool เป็นไฟล์สกุล properties ซึ่งไฟล์ดังกล่าวจะถูกนำมาประมวลผลเพื่อใช้ในการจัดลำดับการใช้งานวิธีรีแฟคทอริง ในส่วนการทำงานนี้จะพัฒนาด้วยภาษาจาวา ซึ่งรายละเอียดของโครงสร้างการทำงานของส่วนของการจัดลำดับการใช้งานวิธีรีแฟคทอริงนั้นจะอธิบายในหัวข้อโครงสร้างคลาสการทำงานหลักของเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในส่วนของการจัดลำดับการใช้งานวิธีรีแฟคทอริงต่อไป

5.1.2 ฟังก์ชันการทำงานหลักของเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริง



ภาพที่ 37 ยูสเคสไดอะแกรมของเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริง

จากภาพที่ 37 เครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริงมีฟังก์ชันการทำงานและรายละเอียดดังนี้

1. จัดลำดับการใช้งานวิธีรีแฟคทอริง

เป็นฟังก์ชันใช้สำหรับปรับแก้ไขซอร์ซโค้ดที่มีร่องรอยที่ผิดพลาดหลายตำแหน่งด้วยวิธีการจัดลำดับการใช้งานวิธีรีแฟคทอริง ภายหลังจากเสร็จสิ้นการปรับแก้ไขจะได้ซอร์ซโค้ดที่ขจัดร่องรอยที่ผิดพลาดทั้งหมดและมีค่าความสามารถในการบำรุงรักษาที่มากที่สุด ในส่วนของฟังก์ชันการทำงานนี้จะทำงานร่วมกับฟังก์ชันระบุร่องรอยที่ผิดพลาดและวิธีรีแฟคทอริงในการปรับแก้ไข และฟังก์ชันคำนวณค่ามาตรวัดเพื่อใช้ในการปรับแก้ไขซอร์ซโค้ดด้วยวิธีการเรียงลำดับการใช้งานวิธีรีแฟคทอริง

2. ระบุร่องรอยที่ผิดพลาดและวิธีรีแฟคทอริงในการปรับแก้ไข

เป็นฟังก์ชันใช้สำหรับค้นหาร่องรอยที่ผิดพลาดภายในซอร์ซโค้ด พร้อมทั้งระบุวิธีรีแฟคทอริงที่ใช้ในการกำจัดร่องรอยที่ผิดพลาดนั้นๆ โดยชนิดของร่องรอยที่ผิดพลาดที่สามารถค้นหาได้ คือ ร่องรอยที่ผิดพลาดแบบเมท็อดที่มีความยาวมาก ร่องรอยที่ผิดพลาดแบบคลาสที่มีขนาดใหญ่และร่องรอยที่ผิดพลาดแบบฟีเจอร์เอ็นวี โดยฟังก์ชันการทำงานนี้จะทำผ่านโปรแกรมอีคลิปส์

3. คำนวณค่ามาตรวัด

เป็นฟังก์ชันใช้สำหรับคำนวณค่ามาตรวัดต่างๆ ของซอร์ซโค้ด โดยมาตรวัดที่ใช้สำหรับการจัดลำดับการใช้งานวิธีรีแฟคทอริงนั้น ได้แก่ ผลรวมค่าความซับซ้อนต่อคลาส ระดับของการขาดเกาะกันเป็นก้อนของเมท็อดภายในคลาส แอฟเฟอร์เรนคัปปลิง และเอฟเฟอร์เรนคัปปลิง

เพื่อนำค่ามาตรวัดมาเป็นเกณฑ์ในการตัดสินใจเลือกวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด โดยฟังก์ชันนี้จะทำผ่านโปรแกรมอีคลิป

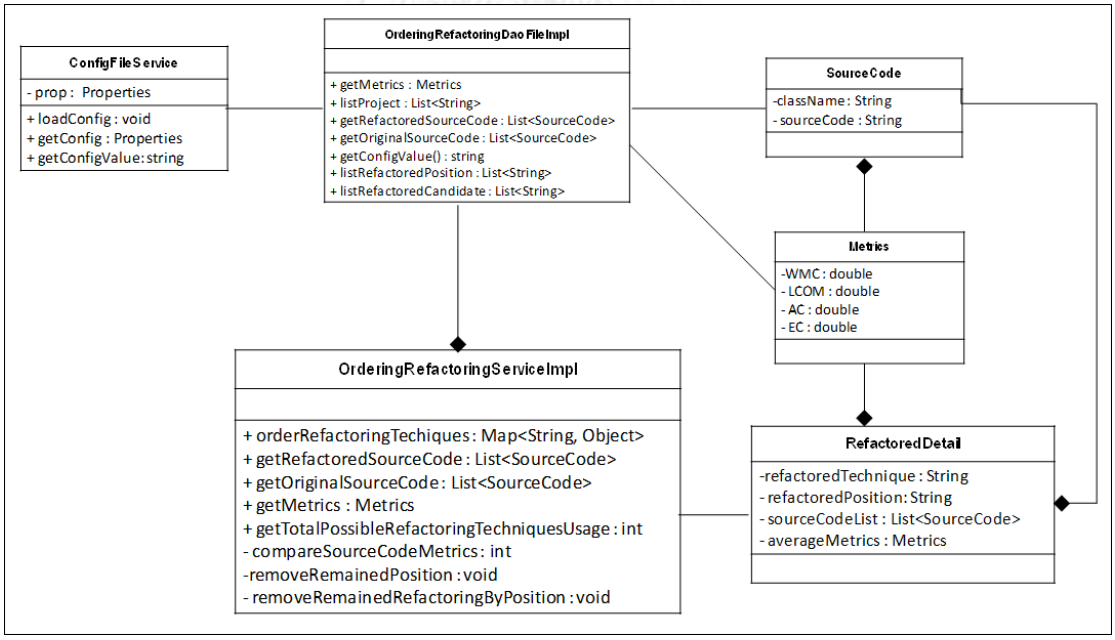
4. ดูรายละเอียดค่ามาตรวัดของซอร์ซโค้ดก่อน-หลังการจัดลำดับการใช้งานวิธีรีแฟคทอริง

เป็นฟังก์ชันใช้สำหรับตรวจสอบค่ามาตรวัดของซอร์ซโค้ดโปรเจคที่เป็นข้อมูลนำเข้าก่อนการจัดลำดับการใช้งานวิธีรีแฟคทอริง และซอร์ซโค้ดภายหลังจากการจัดลำดับการใช้งานวิธีรีแฟคทอริง ค่ามาตรวัดที่จะแสดงผ่านทางหน้าจอนั้น ได้แก่ ผลรวมค่าความซับซ้อนต่อคลาส ระดับของการขาดเกาะกันเป็นก้อนของเมท็อดภายในคลาส แอฟเฟอร์เรนคัปปลิง และเอฟเฟอร์-เรนคัปปลิง

5. ดูรายละเอียดซอร์ซโค้ดก่อน-หลังการจัดลำดับการใช้งานวิธีรีแฟคทอริง

เป็นฟังก์ชันใช้สำหรับดูซอร์ซโค้ดโปรเจคที่เป็นข้อมูลนำเข้าก่อนการจัดลำดับการใช้งานวิธีรีแฟคทอริง และซอร์ซโค้ดภายหลังจากการจัดลำดับวิธีรีแฟคทอริง

5.1.3 โครงสร้างคลาสการทำงานหลักของเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในส่วนของการจัดลำดับการใช้งานวิธีรีแฟคทอริง



ภาพที่ 38 แผนภาพคลาสส่วนของการจัดลำดับการใช้งานวิธีรีแฟคทอริง

ส่วนของการจัดลำดับการใช้งานวิธีรีแฟคทอริงนั้นประกอบด้วยคลาสการทำงานหลักทั้งหมด 6 คลาส ได้แก่

1. คลาส ConfigFileService

เป็นคลาสที่ดึงข้อมูลของไฟล์ orderingRefactoringTool.properties ที่มีข้อมูลตั้งต้นสำหรับส่วนของการจัดลำดับการใช้งานวิธีรีแฟคทอริง โดยจะอ่านไฟล์ดังกล่าวมาจัดเก็บในตัวแปรของคลาส Properties ของภาษาจาวา ซึ่งตัวแปรของคลาสนี้จะจัดเก็บข้อมูลในรูปแบบของคีย์แวร์ลู (key-value) เพื่อนำตัวแปรดังกล่าวไปใช้ในส่วนของคลาส OrderingRefactoringDaoImpl ในการดึงข้อมูลที่ใช้ในการค้นหาลำดับการใช้งานวิธีรีแฟคทอริงต่อไป โดยรายละเอียดของไฟล์ orderingRefactoringTool.properties แสดงได้ดังภาพที่ 39

```
#Input Project
project.list=Movie_Rental

#Source code folder
path.sourcecode.original=c:/OrderingRefactoringTool/source_code/{0}/{1}_original
path.sourcecode.refactored=c:/OrderingRefactoringTool/source_code/{0}/{1}_{2}

#Metrics
path.metrics=c:/OrderingRefactoringTool/metrics/{0}_metrics.properties

#Refactoring Setting
movie.rental.total.refactored.position.list=P1|P2|P3
movie.rental.total.refactored.candidate.list=R1|R2|R3|R4|R5|R6

#Refactoring Techniques For Position
movie.rental.position.P1=R1|R2
movie.rental.position.P2=R3|R4
movie.rental.position.P3=R5|R6
```

ภาพที่ 39 รายละเอียดของไฟล์ orderingRefactoringTool.properties

จากรายละเอียดของไฟล์ orderingRefactoringTool.properties จัดเก็บข้อมูลตั้งต้นดังนี้

- project.list ใช้สำหรับเก็บโปรเจคที่เป็นข้อมูลนำเข้า
- path.sourcecode.original ใช้สำหรับเก็บที่อยู่ของซอร์ซโค้ดก่อนการปรับแก้ไขด้วยวิธีรีแฟคทอริง
- path.sourcecode.refactored ใช้สำหรับเก็บที่อยู่ของซอร์ซโค้ดภายหลังจากการปรับแก้ไขด้วยวิธีรีแฟคทอริง
- path.metrics ใช้สำหรับเก็บที่อยู่ของไฟล์ที่เก็บรายละเอียดค่ามาตรวัดของซอร์ซโค้ด
- movie.rental.total.refactored.position.list ใช้สำหรับเก็บตำแหน่งที่ต้องปรับแก้ไข

- movie.rental.total.refactored.candidate.list ใช้สำหรับเก็บวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไข

- movie.rental.position.<ตำแหน่ง> ใช้สำหรับระบุวิธีรีแฟคทอริงที่ใช้สำหรับแก้ไขตำแหน่งหนึ่งๆ

เมท็อดของคลาส ConfigFileService มีรายละเอียด ดังนี้

- เมท็อด loadConfig ใช้สำหรับโหลดข้อมูลจากไฟล์ orderingRefactoring.properties มาไว้ที่ตัวแปรของคลาส Properties ในภาษาจาวา

- เมท็อด getConfig ใช้สำหรับคืนค่าตัวแปรของคลาส Properties เพื่อให้คลาสอื่นนำไปใช้งาน

- เมท็อด getConfigValue ใช้สำหรับคืนค่าของข้อมูลที่สนใจที่จัดเก็บในตัวแปรของคลาส Properties

2. คลาส OrderingRefactoringDaoFileImpl

เป็นคลาสที่ทำหน้าที่ดึงข้อมูลรายละเอียดต่างๆ ที่ใช้สำหรับส่วนของการจัดลำดับวิธีการใช้งานวิธีรีแฟคทอริง โดยจะเรียกการใช้งานของคลาส ConfigFileService ในการดึงข้อมูลจากไฟล์ต่างๆ อีกที เมื่อดึงข้อมูลเสร็จก็จะนำมาจัดเก็บลงคลาส SourceCode และคลาส Metrics เพื่อให้ง่ายต่อการเรียกใช้งาน ซึ่งรายละเอียดของเมท็อดของคลาสOrderingRefactoring-DaoFileImpl มีดังนี้

- เมท็อด getMetrics ใช้สำหรับคืนค่ามาตรวัดของซอร์ซโค้ด

- เมท็อด listProject ใช้สำหรับคืนค่ารายการโปรเจคที่เป็นข้อมูลนำเข้า

- เมท็อด getRefactoredSourceCode ใช้สำหรับคืนค่าซอร์ซโค้ดภายหลังจากการปรับแก้ไขด้วยวิธีรีแฟคทอริง

- เมท็อด getOriginalSourceCode ใช้สำหรับคืนค่าซอร์ซโค้ดก่อนการปรับแก้ไขด้วยวิธีรีแฟคทอริง

- เมท็อด getConfigValue ใช้สำหรับคืนค่าของข้อมูลที่สนใจที่จัดเก็บในตัวแปรของคลาส Properties

- เมท็อด listRefactoredPosition ใช้สำหรับคืนค่ารายการของตำแหน่งที่ต้องปรับแก้ไข

- เมท็อด listRefactoredCandidate ใช้สำหรับคืนค่ารายการของวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไข

3. คลาส SourceCode

เป็นคลาสที่ทำหน้าที่เก็บรายละเอียดของซอร์ซโค้ด ทั้งก่อนและหลังปรับแก้ไข โดยรายละเอียดตัวแปรของคลาสมีดังนี้

- ตัวแปร className ใช้เก็บชื่อของคลาส

- ตัวแปร sourceCode ใช้เก็บซอร์ซโค้ด

4. คลาส Metrics

เป็นคลาสที่ทำหน้าที่เก็บรายละเอียดมาตรวัดของซอร์ซโค้ด โดยรายละเอียดของตัวแปรคลาสมีดังนี้

- ตัวแปร WMC ใช้เก็บค่ามาตรวัดผลรวมค่าความซับซ้อนต่อคลาส

- ตัวแปร LCOM ใช้เก็บค่ามาตรวัดระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส

- ตัวแปร AC ใช้เก็บค่ามาตรวัดแอฟเฟอร์เรนคัปปลิง

- ตัวแปร EC ใช้เก็บค่ามาตรวัดเอฟเฟอร์เรนคัปปลิง

5. คลาส RefactoredDetail

เป็นคลาสที่ทำหน้าที่เก็บรายละเอียดของการปรับแก้ไขโค้ดด้วยวิธีรีแฟคทอริงในแต่ละตำแหน่งที่ทำการปรับแก้ไข เพื่อใช้ในการเลือกวิธีรีแฟคทอริงที่ให้ค่ามาตรวัดทั้ง 4 ตัวนั้นมีค่าน้อยที่สุดเป็นวิธีที่จะเลือกใช้งานในการปรับแก้ไขในตำแหน่งนั้นๆ โดยรายละเอียดของตัวแปรคลาสมีดังนี้

- ตัวแปร refactoredTechnique ใช้เก็บวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไข

- ตัวแปร refactoredPosition ใช้เก็บตำแหน่งที่ปรับแก้ไข

- ตัวแปร sourceCodeList ใช้เก็บซอร์ซโค้ดในตำแหน่งตามตัวแปร refactoredPosition ด้วยวิธีตามตัวแปร refactoedTechnique

- ตัวแปร averageMetrics ใช้เก็บค่ามาตรวัดเฉลี่ยของซอร์ซโค้ด

6. คลาส RefactoredDetail

เป็นคลาสที่ทำหน้าที่จัดลำดับการใช้งานวิธีรีแฟคทอริง โดยการเรียกใช้งานคลาส OrderingRefactoringServiceImpl ในการดึงข้อมูลเพื่อใช้จัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดของโปรเจคที่เป็นข้อมูลนำเข้า โดยรายละเอียดของตัวแปรคลาสมีดังนี้

- orderRefactoringTechniques ใช้สำหรับจัดเรียงลำดับการใช้งานวิธีรีแฟคทอริง ได้แก่ ลำดับการใช้งานวิธีรีแฟคทอริง ลำดับตำแหน่งที่ปรับแก้ไขในซอร์ซโค้ด จำนวนวิธีรีแฟคทอริงและตำแหน่งที่ต้องพิจารณาในการค้นหาลำดับการใช้งานวิธีรีแฟคทอริงที่เป็นคำตอบ ซึ่งรายละเอียดดังกล่าวจะจัดเก็บในรูปแบบของตัวแปรคลาส Map

- getRefactoredSourceCode ใช้สำหรับคืนค่าซอร์ซโค้ดภายหลังจากปรับแก้ไขด้วยวิธีรีแฟคทอริง

- getOriginalSourceCode ใช้สำหรับคืนค่าซอร์ซโค้ดก่อนการปรับแก้ไขด้วยวิธีรีแฟคทอริง

- getMetrics ใช้สำหรับคืนค่ารายละเอียดของมาตรวัดของซอร์ซโค้ด

- getTotalPossibleRefactoringTechniquesUsage ใช้สำหรับคืนค่าจำนวนรูปแบบที่เป็นไปได้ทั้งหมดในการปรับแก้ไขซอร์ซโค้ดของโปรเจคที่เป็นข้อมูลนำเข้า

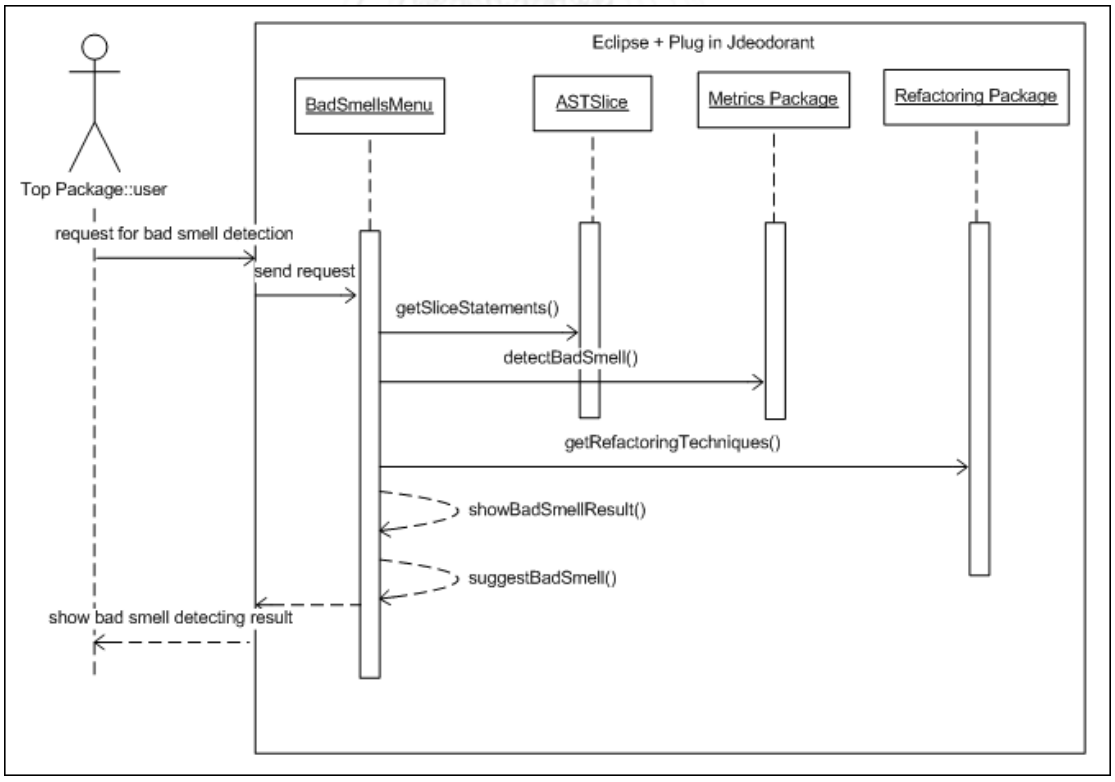
- compareSourceCodeMetrics ใช้สำหรับเปรียบเทียบค่ามาตรวัดระหว่างซอร์ซโค้ด 2 แบบที่ถูกปรับแก้ไข

- removeRemainedPosition ใช้สำหรับลบตำแหน่งที่ถูกปรับแก้ไขแล้วในรายการตำแหน่งที่เหลือที่ต้องปรับแก้ไข

- removeRemainedRefactoringByPosition ใช้สำหรับลบวิธีรีแฟคทอริงที่ใช้งานแล้วออกจากรายการรีแฟคทอริงที่ใช้งานในการปรับแก้ไข

5.1.4 แผนภาพซีเควนของเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริง

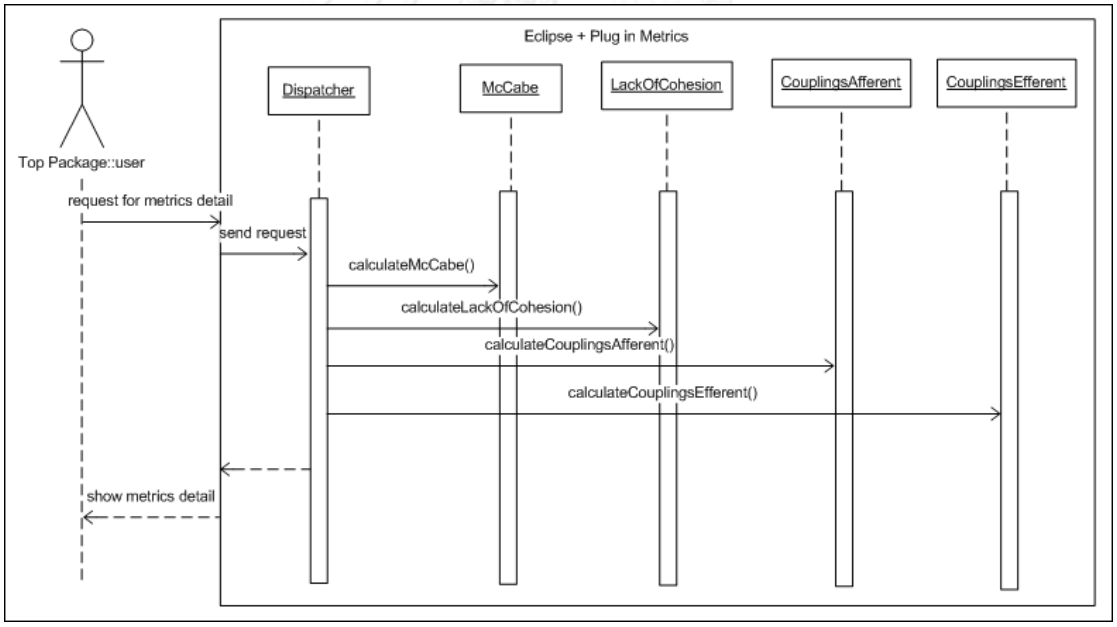
1. แผนภาพซีเควนของฟังก์ชันระบุร่องรอยที่ผิดพลาดและวิธีรีแฟคทอริงในการปรับแก้ไข



ภาพที่ 40 แผนภาพซีเควนของฟังก์ชันระบุร่องรอยที่ผิดพลาดและวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด

จากภาพที่ 40 การค้นหาร่องรอยที่ผิดพลาดและวิธีรีแฟคทอริงนั้นเริ่มจากผู้ใช้งานเรียกใช้งานฟังก์ชันค้นหาร่องรอยที่ผิดพลาดของปลั๊กอินเจดีโอโดแรนต์ในโปรแกรมอีคลิปส์ จากนั้นคลาส BedSmellsMenu จะรับคำร้องขอการค้นหาร่องรอยที่ผิดพลาดจากผู้ใช้งานเพื่อทำการเรียกใช้งานฟังกชันที่เกี่ยวข้องกับการค้นหาร่องรอยที่ผิดพลาด โดยเริ่มจากการเรียกใช้งานคลาส ASTSlice เพื่อทำการตัดซอร์ซโค้ดออกเป็นหน่วยย่อยๆ เพื่อนำไปใช้ในการตรวจสอบกับค่ามาตรวัดที่ใช้บ่งบอกว่าซอร์ซโค้ดดังกล่าวมีลักษณะของร่องรอยที่ผิดพลาดหรือไม่ ซึ่งในการเปรียบเทียบนั้น คลาส ASTSlice จะเรียกใช้งานคลาสคำนวณค่ามาตรวัดภายในแพ็คเกต Metrics เช่น คลาส ConnectivityMetric คลาส MMImportCoupling คลาส LCOM เป็นต้น หลังจากเรียกใช้งานคลาสดังกล่าว ก็จะได้ตำแหน่งของร่องรอยที่ผิดพลาดที่มีอยู่ในซอร์ซโค้ด จากนั้นคลาส BadSmellMenu จะเรียกใช้งานคลาสรีแฟคทอริงแบบต่างๆ ภายในแพ็คเกต Refactoring เช่น คลาสExtractClassRe-factoring คลาส ExtractMethodRefactoring คลาส MoveMethodRefacroring เป็นต้น เพื่อตรวจสอบว่าจะสามารถใช้วิธีรีแฟคทอริงใดบ้างในการปรับแก้ไขโค้ดเพื่อขจัดร่องรอยที่ผิดพลาดที่เกิดขึ้น จากนั้นคลาส BadSmellsMenu จะคืนผลลัพธ์ของการค้นหาร่องรอยที่ผิดพลาดและวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขผ่านทางหน้าจอโปรแกรมอีคลิปส์

2. แผนภาพซีเควนของฟังก์ชันคำนวณค่ามาตรวัด

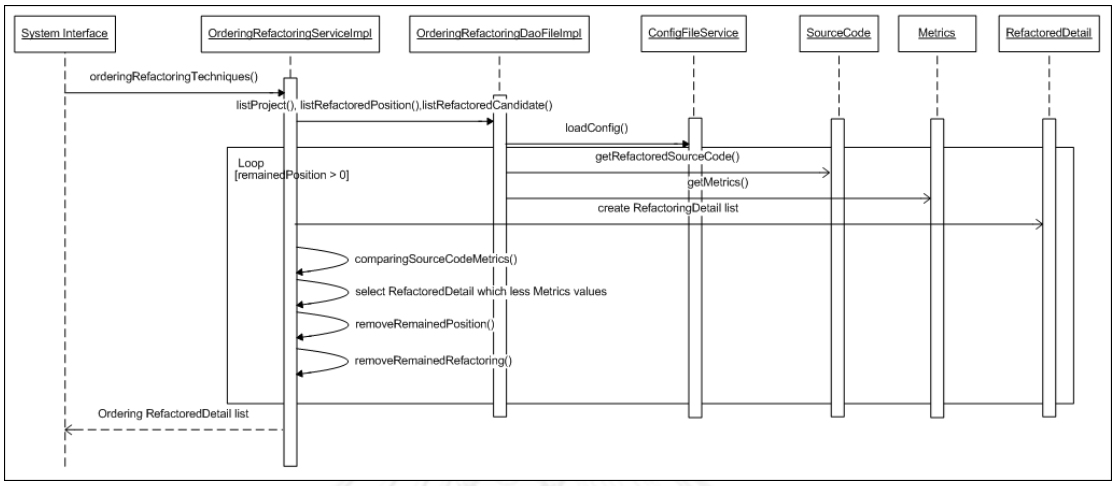


ภาพที่ 41 แผนภาพซีเควนของฟังก์ชันคำนวณค่ามาตรวัด

จากภาพที่ 41 การคำนวณค่ามาตรวัดของซอร์ซโค้ดนั้นเริ่มจากผู้ใช้งานเรียกใช้งานฟังก์ชันคำนวณค่ามาตรวัดของซอร์ซโค้ดของปลั๊กอินเมทริกในโปรแกรมอีคลิปส์ จากนั้นคลาส Dispatch จะเรียกใช้งานเมท็อดในการคำนวณค่ามาตรวัดขของคลาสมาตรวัดต่างๆ เช่น คลาส McCabe คำนวณค่ามาตรวัดผลรวมค่าความซับซ้อนต่อคลาส คลาส LackOfCohesion คำนวณค่าระดับการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส คลาส CouplingAfferent คำนวณค่าแอฟเฟอร์เรนคัปปลิง และคลาส CouplingEfferent คำนวณค่าเอฟเฟอร์เรนคัปปลิง เป็นต้น

ภายหลังจากที่ได้ค่ามาตรวัดต่างๆ แล้ว คลาส Dispatch จะคืนค่ามาตรวัดผ่านทางหน้าจอของโปรแกรมอีคลิปส์

3. แผนภาพซีเควนของฟังก์ชันจัดลำดับการใช้งานวิธีรีแฟคทอริง



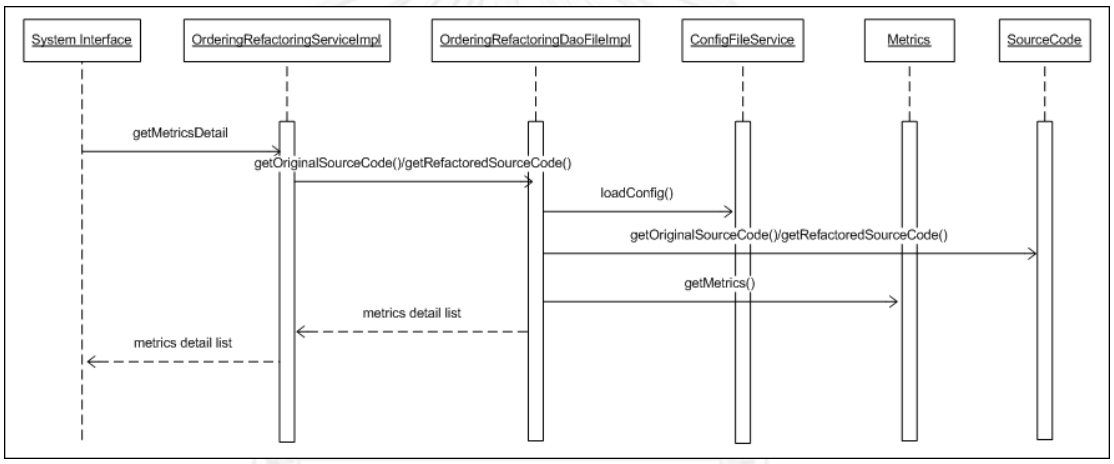
ภาพที่ 42 แผนภาพซีเควนของฟังก์ชันจัดลำดับการใช้งานวิธีรีแฟคทอริง

จากภาพที่ 42 การจัดลำดับการใช้งานวิธีรีแฟคทอริงจะเริ่มจากการคลาสที่ทำหน้าที่เป็นส่วนแสดงผลทางหน้าจอ เช่น หน้าเว็บบราวเซอร์ (web browser), หน้าคอนโซล (console) ต้องการแก้ไขโค้ดโปรแกรมข้อมูลนำเข้า (ในตัวอย่างนี้จะเป็นระบบเช่าภาพยนตร์จากบทที่ 3) ด้วยวิธีรีแฟคทอริง โดยการเรียกใช้งานเมท็อด orderingRefactoringTechniques ของคลาส OrderingRefactoringServiceImpl ภายในเมท็อดนี้จะทำงานเป็นลักษณะของการวนลูปการทำงาน (loop) เพื่อในการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด ก่อนที่จะเริ่มการวนรอบการเรียงลำดับการใช้งานวิธีรีแฟคทอริงนั้น จะทำการดึงข้อมูลเบื้องต้นสำหรับการเรียงลำดับ ได้แก่ ข้อมูลชื่อโปรเจค รายการตำแหน่งที่ต้องปรับแก้ไข และวิธีรีแฟคทอริงที่ใช้ปรับแก้ไข โดยเรียกใช้งานเมท็อด listProject เมท็อด listRefactoredPosition เมท็อด listRefactoredCandidate ของคลาส OrderingRefactoringDaoFileImpl ตามลำดับ ซึ่งคลาส OrderingRefactoringDaoFileImpl จะส่งข้อความให้คลาส ConfigFileService โหลดข้อมูลเบื้องต้นผ่านทางเมท็อด loadCondig แล้วจัดเก็บข้อมูลลงตัวแปรของคลาส Properties เพื่อส่งข้อมูลคืนให้กับคลาส OrderingRefactoring-DaoFileImpl จากนั้นจึงเริ่มวนรอบการทำงาน โดยการวนรอบการทำงานนั้นจะวนรอบตามจำนวนตำแหน่งที่ต้องปรับแก้ไข จำนวนตำแหน่งจะลดลงเรื่อยๆ ทีละ 1 ตำแหน่งเมื่อสิ้นสุดการวนรอบในแต่ละครั้ง และจะหยุดการวนรอบก็ต่อเมื่อไม่มีตำแหน่งเหลือให้ปรับแก้ไข

รายละเอียดของแต่ละรอบการทำงานนั้นจะเริ่มจากดึงค่ารายละเอียดของซอร์ซโค้ดและค่ามาตรวัดของทีละตำแหน่งที่เป็นไปได้ในการปรับแก้ไขในการเรียกใช้เมท็อด getRefactroedSourceCode และ getMetrics ตามลำดับ จากนั้นจึงจัดเก็บรายละเอียดไว้ในตัวแปร RefactoringDetail ซึ่งจะจัดเก็บรายละเอียดของการใช้งานวิธีรีแฟคทอริงในแต่ละตำแหน่งเพื่อนำมาเปรียบเทียบระหว่างการแก้ไขโค้ดด้วยวิธีรีแฟคทอริงแต่ละวิธี ซึ่งจะเรียกใช้เมท็อด

comparingSourceCodeMetrics ในการเปรียบเทียบ โดยจะนับจากจำนวนมาตรวัดที่มีค่าน้อยกว่า วิธีรีแฟคทอริงใดที่มีจำนวนมาตรวัดที่มีค่าน้อยกว่าเป็นจำนวนมากกว่าก็จะถูกเลือกมาให้เป็นลำดับในการใช้งานวิธีรีแฟคทอริงของการวนรอบของรอบนั้นๆ เพื่อนำไปใช้ในการเลือกแก้ไขในตำแหน่งที่เหลือต่อไป ภายหลังจากได้วิธีรีแฟคทอริงและตำแหน่งที่จะปรับแก้ไขในแต่ละรอบแล้วนั้น ก่อนสิ้นสุดการวนรอบจะทำการลบตำแหน่งและวิธีรีแฟคทอริงที่ได้เลือกปรับแก้ไขแล้วนั้นออกจากรายการคงเหลือของตำแหน่งและวิธีรีแฟคทอริงด้วยการเรียกใช้เมท็อด removeRemainedPosition และ removeRemainedRefactoring ตามลำดับ เมื่อสิ้นสุดการวนรอบการทำงานก็จะได้ลำดับรายละเอียดการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดที่ทำให้ได้ซอร์ซโค้ดภายหลังจากการปรับแก้ไขที่มีค่ามาตรวัดทั้ง 4 ชนิดน้อยที่สุดหรือค่าบำรุงรักษาซอฟต์แวร์มากที่สุด

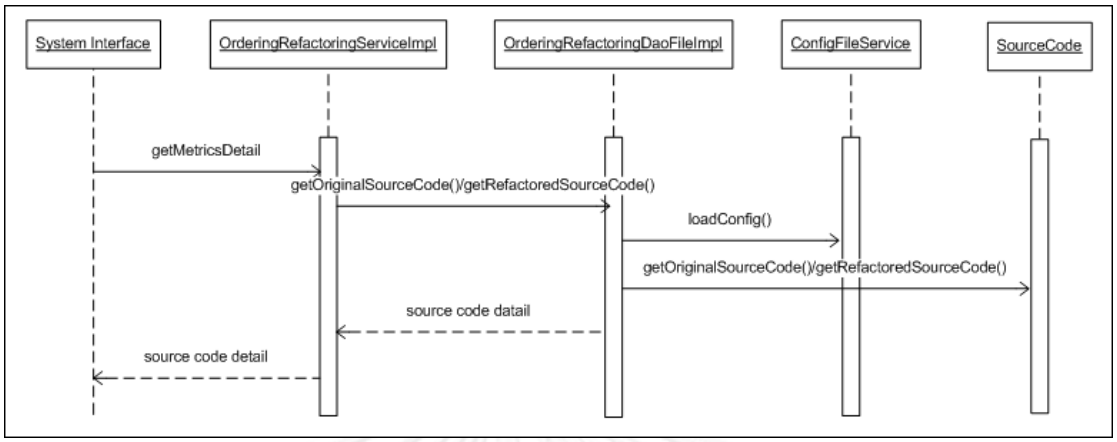
4. **แผนภาพซีเควนของฟังก์ชันดูรายละเอียดค่ามาตรวัดของซอร์ซโค้ดก่อน-หลังการจัดลำดับการใช้งานวิธีรีแฟคทอริง**



ภาพที่ 43 แผนภาพซีเควนของฟังก์ชันดูรายละเอียดค่ามาตรวัดของซอร์ซโค้ดก่อน-หลังการจัดลำดับการใช้งานวิธีรีแฟคทอริง

จากภาพที่ 43 การดูรายละเอียดค่ามาตรวัดของซอร์ซโค้ดก่อน-หลังการจัดลำดับการใช้งานวิธีรีแฟคทอริงเริ่มจากผู้ใช้งานเลือกคลาสที่ต้องดูรายละเอียดของมาตรวัด จากนั้นคลาส OrderingRefactoringSerivceIml จะเรียกใช้งานเมท็อด getOriginalSourceCode กรณีที่คลาสที่เลือกเป็นคลาสก่อนการปรับแก้ไขซอร์ซโค้ด และเรียกใช้งานเมท็อด getRefactoredSourceCode กรณีที่คลาสที่เลือกเป็นคลาสหลังการปรับแก้ไขซอร์ซโค้ด ของคลาส OrderingRefactoringDao-FileImpl โดยคลาส OrderingRefactoringDaoFileImpl จะดึงข้อมูลตั้งต้นของการจัดลำดับการใช้งานวิธีรีแฟคทอริงจากคลาส ConfigFileService จากนั้นจึงดึงข้อมูลรายละเอียดซอร์ซโค้ดและรายละเอียดค่ามาตรวัดผ่านทางเมท็อด getOriginalSourceCode หรือ getRefactoringSource-Code และเมท็อด getMetrics ตามลำดับ หลังจากได้ข้อมูลค่ามาตรวัดคลาส OrderingRefactor-ingFileImpl จะส่งข้อมูลค่ามาตรวัดของซอร์ซโค้ดกลับไปยังคลาส OrderingRefactoringService-Impl เพื่อให้คลาส OrderingRefactoringSeriveImpl ส่งข้อมูลกลับไปยังหน้าจอส่วนแสดงผลต่อไป

5. แผนภาพซีเควนของฟังก์ชันดูรายละเอียดของซอร์ซโค้ดก่อน-หลังการใช้งานวิธีรีแฟคทอริง



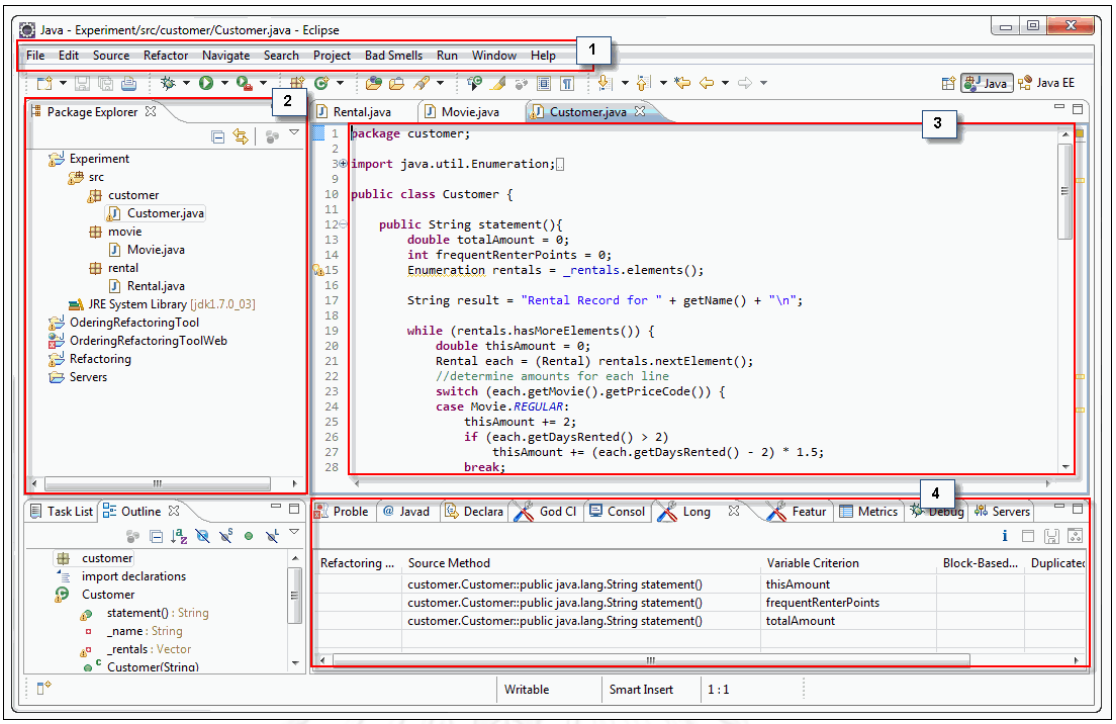
ภาพที่ 44 แผนภาพซีเควนของฟังก์ชันดูรายละเอียดของซอร์ซโค้ดก่อน-หลังการใช้งานวิธีรีแฟคทอริง

จากภาพที่ 44 การดูรายละเอียดของซอร์ซโค้ดก่อน-หลังการจัดลำดับการใช้งานวิธีรีแฟคทอริงเริ่มจากผู้ใช้งานเลือกคลาสที่ต้องดูรายละเอียดของมาตรวัด จากนั้นคลาส OrderingRefactoringSerivceIml จะเรียกใช้งานเมท็อด getOriginalSourceCode กรณีที่คลาสที่เลือกเป็นคลาสก่อนการปรับแก้ไขซอร์ซโค้ด และเรียกใช้งานเมท็อด getRefactoredSourceCode กรณีที่คลาสที่เลือกเป็นคลาสหลังการปรับแก้ไขซอร์ซโค้ด ของคลาส OrderingRefactoring-DaoFileImpl โดยคลาส OrderingRefactoringDaoFileImpl จะดึงข้อมูลตั้งต้นของการจัดลำดับการใช้งานวิธีรีแฟคทอริงจากคลาส ConfigFileService จากนั้นจึงดึงข้อมูลรายละเอียดซอร์ซโค้ดและรายละเอียดค่ามาตรวัดผ่านทางเมท็อด getOriginalSourceCode หรือ getRefactoringSource-Code หลังจากได้รายละเอียดซอร์ซโค้ดแล้ว OrderingRefactoringDaoFileImpl จะส่งรายละเอียดซอร์ซโค้ดกลับไปยังคลาส OrderingRefactoringServiceImpl เพื่อให้คลาส OrderingRefactoring-SeriveImpl ส่งรายละเอียดซอร์ซโค้ดกลับไปยังหน้าจอส่วนแสดงผลต่อไป

5.2 ส่วนของรายละเอียดเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริง

5.2.1 โปรแกรมอีคลิปส์

เป็นเครื่องมือที่ใช้สำหรับเขียนโค้ดโปรแกรมในภาษาจาวา โดยทางวิทยานิพนธ์นี้ได้นำเอาปลั๊กอินเจดีโอโดแรนต์เพื่อเพิ่มคุณสมบัติในการค้นหาร่องรอยที่ผิดพลาดและวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไข และปลั๊กอินเมทริกในการคำนวณค่ามาตรวัดของซอร์ซโค้ด รายละเอียดหน้าจอของเครื่องมือมีดังนี้



ภาพที่ 45 หน้าจอหลักโปรแกรมอีคลิปส์

จากภาพที่ 45 หน้าจอหลักโปรแกรมอีคลิปส์ประกอบด้วย 4 ส่วนหลัก ได้แก่

- ส่วนของเมนูการจัดการ (หมายเลข 1) เป็นส่วนของการจัดการต่างๆ ภายในโปรแกรม

- ส่วนแสดงโครงสร้างโปรเจค (หมายเลข 2) เป็นส่วนที่ใช้สำหรับแสดงโครงสร้างของโปรเจคที่เขียนโปรแกรม สามารถเข้าถึงและจัดการไฟล์ต่างๆ ที่อยู่ภายใต้โปรเจคได้

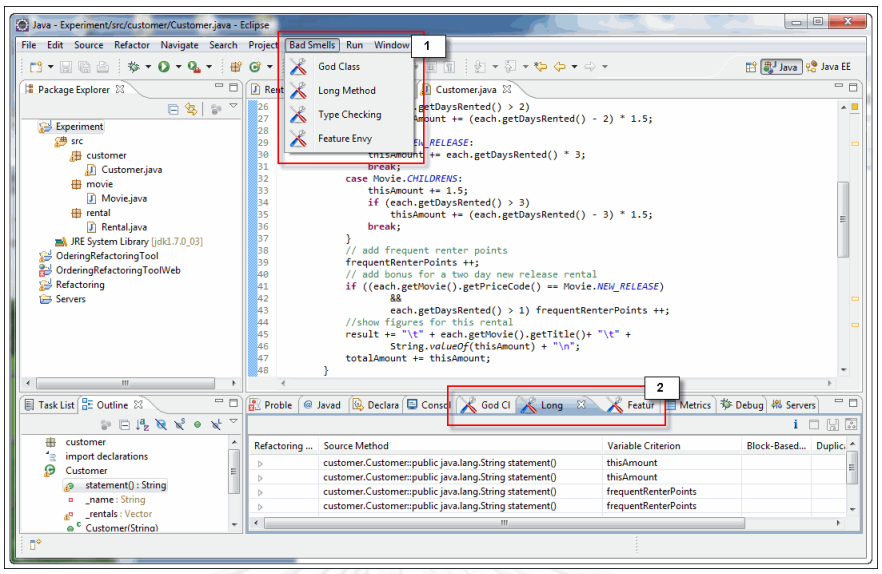
- ส่วนเขียนโค้ด (หมายเลข 3) เป็นส่วนที่ใช้ในการเขียนและปรับแก้ไขโค้ด

- ส่วนการแสดงผล (หมายเลข 4) เป็นส่วนที่ใช้แสดงผลของการค้นหาร่องรอยที่ผิดพลาด และค่ามาตรวัดของซอร์ซโค้ด

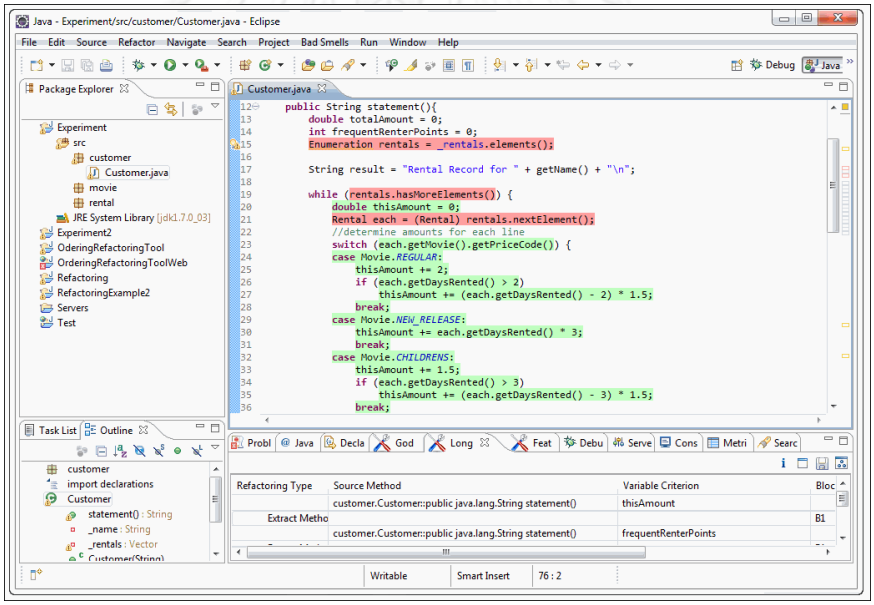
หน้าจอหลักโปรแกรมอีคลิปส์ในแต่ละส่วนมีรายละเอียดดังนี้

1. ส่วนของการค้นหาร่องรอยที่ผิดพลาด

ปลั๊กอินเจดีโอโดแรนต์ช่วยเพิ่มความสามารถในการค้นหาร่องรอยที่ผิดพลาดที่มีอยู่ในซอร์ซโค้ด และแนะนำวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขโค้ดให้กับโปรแกรมอีคลิปส์ จากภาพที่ 46 ปลั๊กอินเจดีโอโดแรนต์สามารถค้นหาร่องรอยที่ผิดพลาดได้ทั้งหมด 4 ชนิด คือ ร่องรอยที่ผิดพลาดแบบเมท็อดที่มีความยาวมาก (Large Class) โดยเจดีโอโดแรนต์จะแสดงด้วย GodClass ร่องรอยที่ผิดพลาดแบบแบบเมท็อดที่มีความยาวมาก (Long Method) การตรวจสอบชนิดตัวแปร (Type Checking) และร่องรอยที่ผิดพลาดแบบฟีเจอร์เอนวี (Feature Envy) ดังหมายเลขที่ 1 ใน ในส่วนของหมายเลขที่ 2 นั้นจะแสดงผลของการค้นหาร่องรอยที่ผิดพลาด โดยระบุเมท็อดและตัวแปรที่เป็นสาเหตุที่ทำให้เกิดข้อผิดพลาด และระบุวิธีรีแฟคทอริงในการปรับแก้ไขร่องรอยที่ผิดพลาดนั้นๆ



ภาพที่ 46 หน้าจอการค้นหาร่องรอยที่ผิดพลาดของโปรแกรมอีคลิปส์ โดยการใช้ปลั๊กอินเจดีโอโดแรนต์

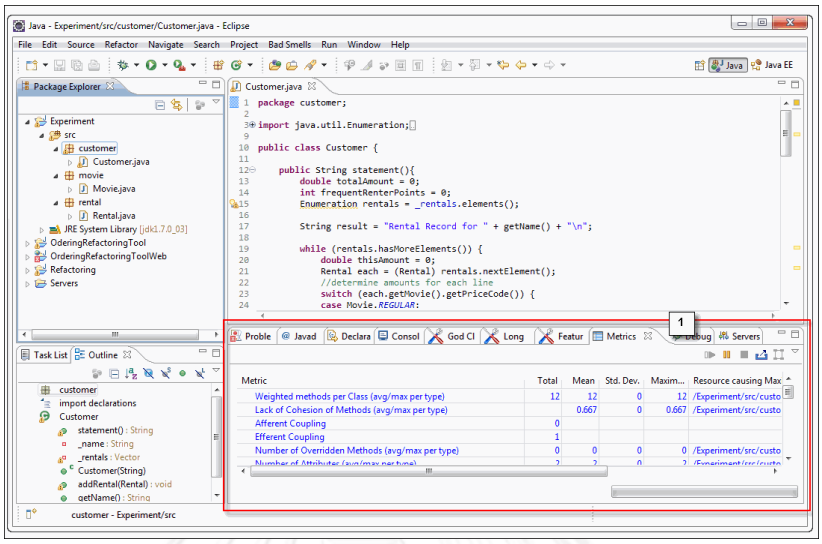


ภาพที่ 47 หน้าจอแสดงการไฮไลต์โค้ดส่วนที่ต้องทำการปรับแก้ไขด้วยวิธีรีแฟคทอริง จากการค้นหาร่องรอยที่ผิดพลาดของโปรแกรมอีคลิปส์ โดยการใช้ปลั๊กอินเจดีโอโดแรนต์

2. ส่วนของการคำนวณค่ามาตรวัด

ปลั๊กอินเมทริกช่วยเพิ่มความสามารถในการคำนวณค่ามาตรวัดของซอร์ซโค้ด โดยปลั๊กอินเมทริกสามารถคำนวณค่ามาตรวัดต่างๆ ได้แก่ มาตรวัดผลรวมค่าความซับซ้อนต่อคลาส (Weighted methods per Class) ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส (Lack of Cohesion of Methods) แอฟเฟอร์เรนคัปปลิง (Afferent Coupling)

เอฟเฟอร์เรนคัปปลิง (Efferent Coupling) เป็นต้น รายละเอียดที่จะแสดงผ่านทางหน้าจอ ได้แก่ จำนวนรวม ค่าเฉลี่ย ส่วนเบี่ยงเบนมาตรฐาน ค่าสูงสุด และคลาสที่ทำให้ค่ามาตรวัดสูงสุด รายละเอียดของหน้าจอแสดงได้ดังภาพที่ 48



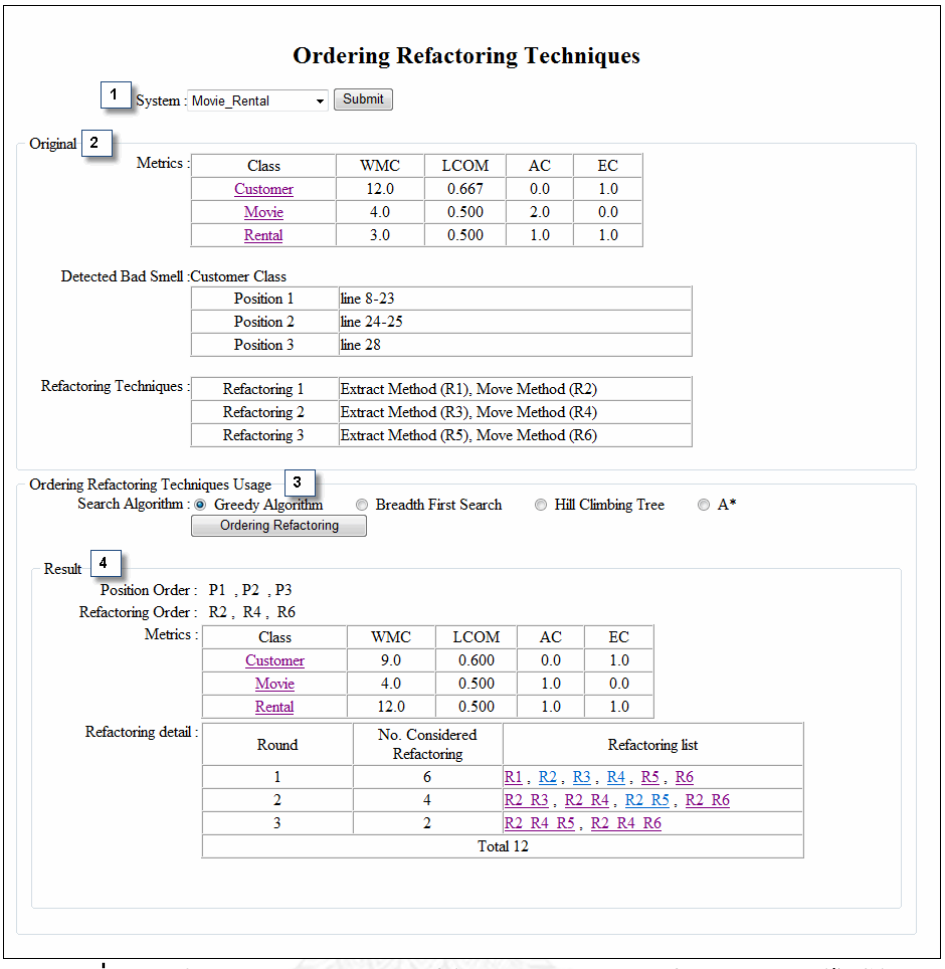
ภาพที่ 48 หน้าจอการคำนวณค่ามาตรวัดของโปรแกรมอีคลิปส์ โดยการใช้ปลั๊กอินเมทริก

5.2.2 ส่วนของการจัดลำดับการใช้งานวิธีรีแฟคทอริง

รายละเอียดของหน้าจอของการแสดงผลลัพธ์การจัดลำดับการใช้งานวิธีรีแฟคทอริงนั้น พัฒนาด้วยภาษาจาวาในรูปแบบของเว็บแอพลิเคชัน (Java Web Application) แสดงผลลัพธ์ผ่านทางหน้าบราวเซอร์ โดยรายละเอียดของหน้าจอแสดงดังภาพที่ 49 หน้าจอหลักของเครื่องมือประกอบด้วย 4 ส่วนหลัก ได้แก่

- ส่วนสำหรับเลือกโปรเจคที่เป็นข้อมูลนำเข้า (หมายเลข 1)
- ส่วนสำหรับแสดงผลรายละเอียดข้อมูลนำเข้าก่อนการจัดลำดับการใช้งานวิธีรีแฟคทอริง (หมายเลข 2)
- ส่วนสำหรับเลือกอัลกอริทึมในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด (หมายเลข 3)
- ส่วนสำหรับแสดงผลลัพธ์ภายหลังจากการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด (หมายเลข 4)

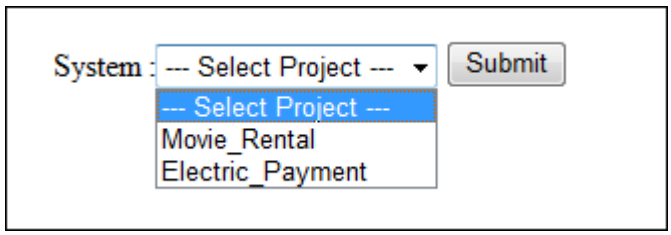
รายละเอียดและขั้นตอนการใช้งานของส่วนการจัดลำดับการใช้งานวิธีรีแฟคทอริงมีรายละเอียดดังนี้



ภาพที่ 49 หน้าจอการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด

1. ส่วนสำหรับเลือกโปรเจคที่เป็นข้อมูลนำเข้า

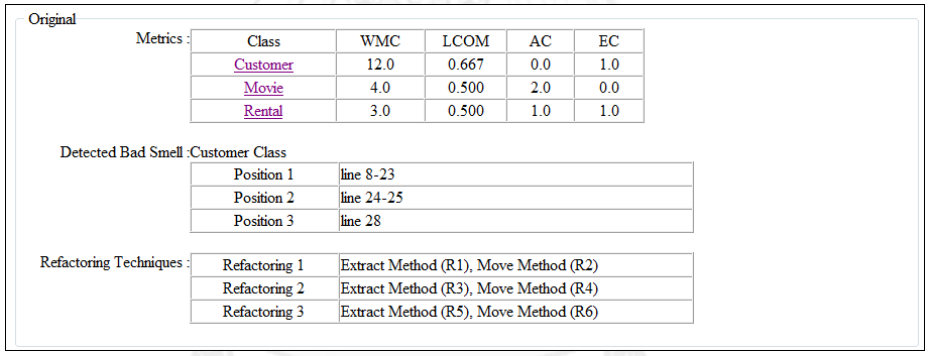
ใช้แสดงรายการโปรเจคที่เป็นข้อมูลนำเข้าทั้งหมดที่มีอยู่ภายในระบบ เพื่อใช้สำหรับการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด รายการทั้งหมดจะแสดงในรูปของลิสต์รายการ (Drop down list) ให้ผู้ใช้งานเลือกโปรเจคที่จะนำมาทำการจัดลำดับการใช้งานวิธีรีแฟคทอริง หน้าจอในส่วนแรกนั้นจะแสดงเพียงแค่ลิสต์รายการเท่านั้น ข้อมูลในส่วนอื่นๆ จะยังคงไม่แสดงจนกว่าผู้ใช้งานเลือกโปรเจคที่เป็นข้อมูลนำเข้าและกดปุ่มยืนยัน ภายหลังจากนั้นระบบจะแสดงข้อมูลในส่วนสำหรับแสดงผลรายละเอียดข้อมูลนำเข้าก่อนการจัดลำดับการใช้งานวิธีรีแฟคทอริงและส่วนสำหรับเลือกอัลกอริทึมในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดต่อไป โดยจากภาพที่ 50 ภายในลิสต์รายการมีระบบเช่าภาพยนตร์ (Movie_Rental) และระบบคำนวณค่าไฟฟ้า (Electric Payment) ให้เลือกนำมาจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดได้



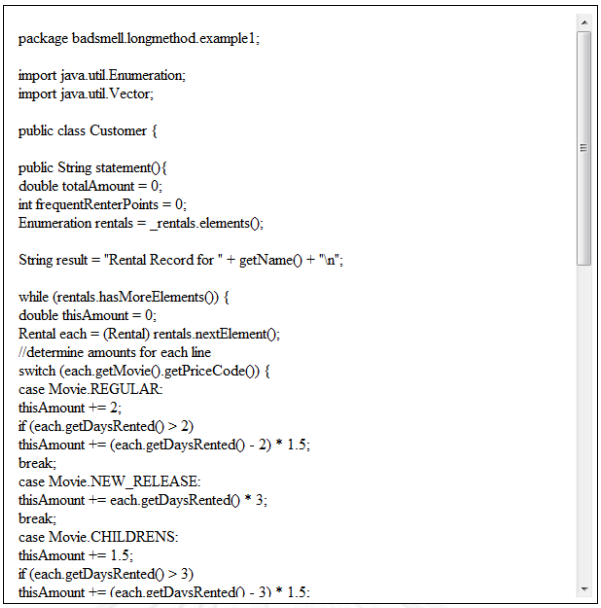
ภาพที่ 50 หน้าจอส่วนสำหรับเลือกโปรเจคที่เป็นข้อมูลนำเข้า

2. ส่วนสำหรับแสดงผลรายละเอียดข้อมูลนำเข้าก่อนการจัดลำดับการใช้งานวิธีรีแฟคทอริง

ใช้แสดงผลรายละเอียดของข้อมูลนำเข้าก่อนที่จะนำมาจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด ได้แก่ ค่ามาตรวัดของซอร์ซโค้ดของคลาส คลาสที่เป็นสาเหตุของร่องรอยที่ผิดพลาด ตำแหน่งของร่องรอยที่ผิดพลาด และวิธีรีแฟคทอริงที่ใช้สำหรับแก้ไขร่องรอยผิดที่พลาดนั้นๆ ในส่วนของรายละเอียดค่ามาตรวัดนั้น สามารถคลิกชื่อคลาสเพื่อดูรายละเอียดของซอร์ซโค้ดก่อนการจัดเรียงลำดับวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดได้ แสดงได้ดังภาพที่ 51 - ภาพที่ 52



ภาพที่ 51 หน้าจอส่วนสำหรับแสดงผลรายละเอียดข้อมูลนำเข้าก่อนการจัดลำดับการใช้งานวิธีรีแฟคทอริง



```
package badsmell.longmethod.example1;

import java.util.Enumeration;
import java.util.Vector;

public class Customer {

public String statement(){
double totalAmount = 0;
int frequentRenterPoints = 0;
Enumeration rentals = _rentals.elements();

String result = "Rental Record for " + getName() + "\n";

while (rentals.hasMoreElements()) {
double thisAmount = 0;
Rental each = (Rental) rentals.nextElement();
//determine amounts for each line
switch (each.getMovie().getPriceCode()) {
case Movie.REGULAR:
thisAmount += 2;
if (each.getDaysRented() > 2)
thisAmount += (each.getDaysRented() - 2) * 1.5;
break;
case Movie.NEW_RELEASE:
thisAmount += each.getDaysRented() * 3;
break;
case Movie.CHILDRENS:
thisAmount += 1.5;
if (each.getDaysRented() > 3)
thisAmount += (each.getDaysRented() - 3) * 1.5;
```

ภาพที่ 52 หน้าจอส่วนสำหรับแสดงรายละเอียดซอร์ซโค้ด ก่อนการจัดลำดับการใช้งานวิธีรีแฟคทอริง

3. ส่วนสำหรับเลือกอัลกอริทึมในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด

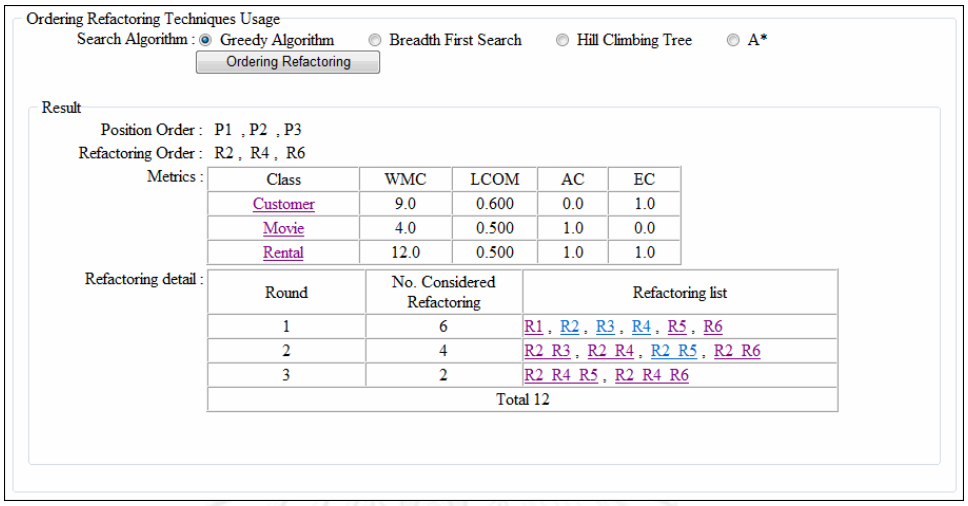
ใช้แสดงรายการอัลกอริทึมที่สามารถเลือกใช้งานในการจัดลำดับการใช้งานวิธีรีแฟคทอริงได้ ซึ่งอัลกอริทึมที่สามารถเลือกใช้งานได้ ได้แก่ อัลกอริทึมละโมบ (Greedy Algorithm) อัลกอริทึมค้นหาแนวกว้าง (Breath First Search) อัลกอริทึมปีนเขา (Hill Climbing) และอัลกอริทึมเอสตาร์ (A* - A Star) โดยมีปุ่มยืนยันการจัดลำดับการใช้งานวิธีรีแฟคทอริง เพื่อเริ่มทำการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด แสดงได้ดังภาพที่ 53

ภาพที่ 53 หน้าจอส่วนสำหรับเลือกอัลกอริทึมในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด

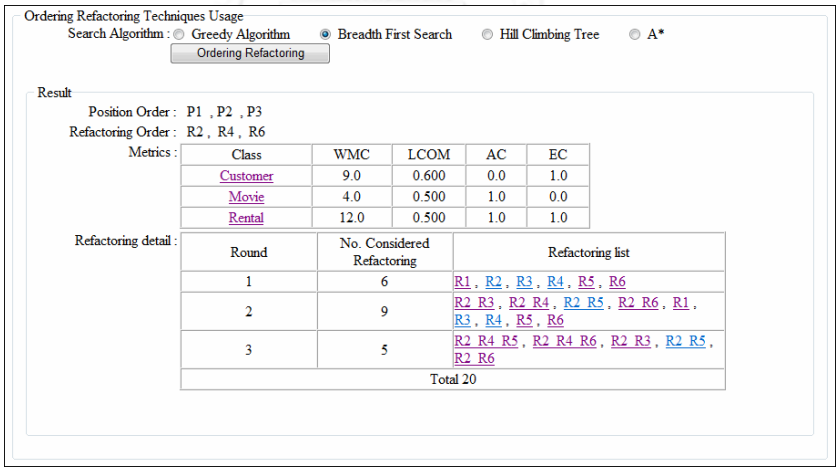
4. ส่วนสำหรับแสดงผลลัพธ์ภายหลังจากการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด

ใช้แสดงผลรายละเอียดของข้อมูลต่างๆ ภายหลังจากการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด ได้แก่ ลำดับของตำแหน่งที่ปรับแก้ไข ลำดับของวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไข ค่ามาตรวัดของซอร์ซโค้ด และรายละเอียดของการเข้าถึงโหนดข้อมูลในการจัดลำดับ ในส่วนของรายละเอียดของการเข้าถึงโหนดข้อมูลนั้นจะบอกจำนวนของโหนด

ทั้งหมดที่ต้องพิจารณา เพื่อเลือกโหนดที่ให้ค่ามาตรวัดทั้ง 4 ตัวที่น้อยที่สุดในแต่ละการวนรอบการทำงาน โดยที่ผู้ใช้งานสามารถคลิกที่โหนดลำดับและชื่อคลาสที่สนใจ เพื่อดูรายละเอียดของค่ามาตรวัดและซอร์ซโค้ดภายหลังจากจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด ซึ่งแสดงดังภาพที่ 54 - ภาพที่ 57 ตามลำดับ



ภาพที่ 54 หน้าจอส่วนสำหรับแสดงผลลัพธ์ภายหลังจากการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดด้วยการใช้อัลกอริทึมละโมบ



ภาพที่ 55 หน้าจอส่วนสำหรับแสดงผลลัพธ์ภายหลังจากการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดด้วยการใช้อัลกอริทึมค้นหาแนวกว้าง

Ordering : R2_R4_R6

Class	WMC	LCOM	AC	EC
Customer	9.0	0.600	0.0	1.0
Movie	4.0	0.500	1.0	0.0
Rental	12.0	0.500	1.0	1.0

ภาพที่ 56 หน้าจอส่วนสำหรับแสดงผลลัพธ์ค่ามาตรวัดที่ได้จากการเลือกลำดับการใช้งานวิธีรีแฟคทอริงที่สนใจ

```
package badsmell.longmethod.movierental_R2_R4_R6;

import java.util.Enumeration;
import java.util.Vector;

public class Customer {

public String statement(){
Enumeration rentals = _rentals.elements();
String result = "Rental Record for " + getName() + "\n";
while (rentals.hasMoreElements()) {
double thisAmount = 0;
Rental each = (Rental) rentals.nextElement();
thisAmount = each.getCharge();

result += "\t" + each.getMovie().getTitle()+ "\t" + String.valueOf(thisAmount) + "\n";

}
//add footer lines
result += "Amount owed is " + String.valueOf(getTotalCharge()) + "\n";
result += "You earned " + String.valueOf(getTotalFrequentRenterPoints()) + " frequent renter
points";
return result;

}

private double getTotalCharge() {
double result = 0;
Enumeration rentals = _rentals.elements();
while (rentals.hasMoreElements()) {
Rental each = (Rental) rentals.nextElement();
```

ภาพที่ 57 หน้าจอส่วนสำหรับแสดงรายละเอียดซอร์ซโค้ดภายหลังการจัดลำดับการใช้งานวิธีรีแฟคทอริง

บทที่ 6
สรุปผลการวิจัยและข้อเสนอแนะ

6.1 สรุปผลการวิจัย

วิทยานิพนธ์ฉบับนี้สามารถแบ่งเนื้อหาออกเป็นสองส่วนหลัก ๆ ซึ่งประกอบด้วย ส่วนแรกคือการนำเสนอวิธีการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดโดยใช้มาตรวัดเชิงวัตถุและอัลกอริทึมละโมบ และส่วนที่สองคือเครื่องมือสำหรับช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริง โดยส่วนแรกนั้นได้อธิบายลำดับขั้นตอนการจัดเรียงลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด ซึ่งประกอบด้วย 5 ขั้นตอนดังต่อไปนี้ ขั้นตอนการค้นหาร่องรอยที่ผิดพลาด ขั้นตอนการประยุกต์ใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด ขั้นตอนการเลือกลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด และขั้นตอนการประเมินผลลัพธ์ของลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด โดยเนื้อหาในส่วนของ 4 ขั้นตอนแรกนั้นอยู่ในบทที่ 3 และขั้นตอนของการประเมินผลลัพธ์ของลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดนั้นอยู่ในบทที่ 4 ในส่วนของเครื่องมือสำหรับช่วยในการจัดลำดับการใช้งานวิธีรีแฟคอทอริงจะอธิบายถึงรายละเอียดของเครื่องมือที่งานวิจัยนี้ได้นำมาใช้ในการทดลองและพัฒนาเครื่องมือเพิ่มเติมเพื่อช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด โดยรายละเอียดของส่วนนี้จะอยู่ในบทที่ 5

งานวิจัยนี้ได้นำเสนอวิธีการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดโดยใช้มาตรวัดเชิงวัตถุและอัลกอริทึมละโมบ เพื่อให้ได้ซอร์ซโค้ดภายหลังจากการปรับแก้ไขนั้นมีค่าความสามารถในการบำรุงรักษาที่มากที่สุด โดยการนำวิธีการจัดลำดับการใช้งานวิธีรีแฟคทอริงไปใช้แก้ไขซอร์ซโค้ดที่มีลักษณะของร่องรอยที่ผิดพลาดหลายลักษณะและหลายตำแหน่งที่ต้องปรับแก้ไข ลักษณะร่องรอยที่ผิดพลาดที่งานวิจัยนี้ได้นำมาทดสอบ ได้แก่ ร่องรอยที่ผิดพลาดแบบเมท็อดที่มีความยาวมาก ร่องรอยที่ผิดพลาดแบบคลาสที่มีขนาดใหญ่ และร่องรอยที่ผิดพลาดแบบฟีเจอร์เอนวี ภายหลังการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดดังกล่าว จะได้ซอร์ซโค้ดที่มีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์มากกว่าซอร์ซโค้ดที่ไม่พิจารณาเรื่องการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด

งานวิจัยนี้ได้ทำการเปรียบเทียบการทำงานในการจัดลำดับการใช้งานวิธีรีแฟคทอริงระหว่างอัลกอริทึมละโมบกับอัลกอริทึมชนิดอื่นๆ ได้แก่ อัลกอริทึมค้นหาแนวกว้าง อัลกอริทึมปีนเขา และอัลกอริทึมเอสตาร์ โดยพิจารณาในด้านของผลลัพธ์ค่าความสามารถของซอร์ซโค้ดภายหลังจากการปรับแก้ไขและเวลาที่ใช้ในการจัดลำดับการใช้งานวิธีรีแฟคทอริง ซึ่งผลลัพธ์ของการเปรียบเทียบนั้น ซอร์ซโค้ดที่ได้ภายหลังจากการจัดลำดับการใช้งานวิธีรีแฟคทอริงมีค่าความสามารถในการบำรุงรักษาที่สูงที่สุดเท่ากับอัลกอริทึมค้นหาแนวกว้างและอัลกอริทึมเอสตาร์ และใช้เวลาในการจัดลำดับน้อยที่สุดในบรรดาอัลกอริทึมทั้งหมด

6.2 ข้อจำกัด

การจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดโดยการใช้มาตรวัดเชิงวัตถุและอัลกอริทึมละโมบนั้น ในงานวิจัยนี้จะเน้นที่ผลลัพธ์ของซอร์ซโค้ดเป็นหลัก คือ ค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ของซอร์ซโค้ดจะต้องมากที่สุด ซึ่งในกรณีที่ซอร์ซโค้ดที่จะนำมาปรับแก้ไขนั้นในแต่ละตำแหน่งมีวิธีรีแฟคทอริงให้เลือกปรับแก้ไขเพียงวิธีเดียว และแต่ละตำแหน่งที่ปรับแก้ไขนั้นไม่มีความเกี่ยวข้องกันหรือการแก้ไขส่วนใดส่วนหนึ่งไม่กระทบซึ่งกันและกัน การสลับตำแหน่งหรือวิธีรีแฟคทอริงในการปรับแก้ไขผลสุดท้ายจะได้ซอร์ซโค้ดที่มีค่าความสามารถในการบำรุงรักษาที่เท่ากัน ซึ่งเป็นในมุมมองที่มองเพียงแค่ด้านของผลลัพธ์ของการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด แต่ถ้ามองในมุมของความง่ายในการปรับแก้ไขซอร์ซโค้ดในช่วงของการจัดลำดับการใช้งานวิธีรีแฟคทอริงนั้น การเลือกใช้งานวิธีรีแฟคทอริงที่ภายหลังจากปรับแก้ไขซอร์ซโค้ดแล้วได้ซอร์ซโค้ดที่มีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ที่สูงตั้งแต่ช่วงเริ่มต้นของการปรับแก้ไข ก็จะทำให้ซอร์ซโค้ดสำหรับการปรับแก้ไขในตำแหน่งต่อไปนั้นสามารถทำความเข้าใจและปรับแก้ไขได้ง่ายขึ้น ในทางตรงกันข้ามถ้าเลือกใช้งานวิธีรีแฟคทอริงที่ภายหลังจากการปรับแก้ไขแล้วได้ซอร์ซโค้ดภายหลังจากการปรับแก้ไขมีค่าความสามารถในการบำรุงรักษาที่ต่ำ ก็จะทำให้การปรับแก้ไขซอร์ซโค้ดในตำแหน่งที่เหลือนั้นทำได้ยาก ซึ่งในมุมมองส่วนนี้จะสามารถวัดได้จากค่าความพยายาม (Effort) ในการปรับแก้ไขโค้ด ซึ่งอยู่นอกเหนืองานวิจัยนี้ จึงไม่ได้พิจารณาในมุมมองของค่าความพยายามที่ใช้ในการปรับแก้ไขในแต่ละตำแหน่งระหว่างการปรับแก้ไขซอร์ซโค้ดด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง

6.3 แนวทางการวิจัยต่อไป

1. เพิ่มจำนวนมาตรวัดเชิงวัตถุชนิดอื่นๆ ที่ใช้เป็นเกณฑ์ในการตัดสินใจเลือกเส้นทางการใช้งานวิธีรีแฟคทอริงสำหรับการจัดลำดับการใช้งานวิธีรีแฟคทอริง เพื่อให้การเลือกเส้นทางนั้นมีความแม่นยำมากยิ่งขึ้น

2. เพิ่มการพิจารณาด้านค่าความพยายามที่ใช้สำหรับการปรับแก้ไขโค้ดในแต่ละตำแหน่งมาเป็นเกณฑ์ในการตัดสินใจเลือกเส้นทางการใช้งานวิธีรีแฟคทอริง เพื่อเป็นการยืนยันว่าการจัดลำดับการใช้งานวิธีรีแฟคทอริงนั้นจะช่วยให้การปรับแก้ไขโค้ดนั้นทำได้ง่ายขึ้น

3. เพิ่มการพิจารณาถึงผลกระทบภายหลังจากการปรับแก้ไขโค้ดด้วยวิธีรีแฟคทอริง ในด้านของความพยายามที่ใช้ในการปรับแก้ไข การกระทบกันของส่วนโค้ดการทำงาน และความพยายามที่ใช้ในการทดสอบการทำงานของโปรแกรมภายหลังจากการปรับแก้ไข เพื่อใช้เป็นเกณฑ์ในการตัดสินใจว่าคุ่มค่าที่จะทำการปรับแก้ไขด้วยวิธีรีแฟคทอริงหรือไม่

4. เพิ่มการพิจารณาเรื่องการถ่วงน้ำหนักของมาตรวัดแต่ละตัวที่ใช้เป็นเกณฑ์ในการตัดสินใจเลือกเส้นทาง เนื่องจากมาตรวัดแต่ละตัวมีสเกลที่แตกต่างกัน จึงจำเป็นต้องกำหนดลำดับความสำคัญของมาตรวัดที่สนใจและมีการเปลี่ยนแปลงเกิดขึ้น ตรงส่วนนี้จะทำให้เกณฑ์การพิจารณาเลือกซอร์ซโค้ดที่มีค่าความสามารถในการบำรุงรักษามากที่สุดนั้นมีความแม่นยำมากยิ่งขึ้น

5. ศึกษาถึงความสัมพันธ์ระหว่างวิธีรีแฟคทอริงกับมาตรวัดเชิงวัตถุ เพื่อใช้เป็นเกณฑ์การเลือกใช้งานมาตรวัดเชิงวัตถุในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด

รายการอ้างอิง

1. Fowler, M. , K. Beck, J. Brant, W. Opdyke and D Roberts, *Refactoring: Improving the Design Of Existing Code. Addision Wesley Professional*. 1990.
2. Meananeatra, P., S. Rongviriyapanish and T. Apiwattanapong, *In dentifying Refactoring Through Formal Model Based on Data Flow Graph*, in *The 5th Malaysian Conference in Software Engineering (MySEC)*. 2011: Malaysia. p. 113-118.
3. Meananeatra, P. , S. Rongviriyapanish and T. Apiwattanapong, *Using Software Metrics to Select Refactoring for Long Method Bad Smell*, in *The 8th Electrical Engineering/ Electronics, Computer, Telecommunications and Information Technology (ECTI)*. 2011: Khon Kaen, Thailand. p. 492-495.
4. Pienlert, T. and P. Muenchaisri, *Bad-Smell Detection Using OO SW Metrics*, in *International Conference on Computer Science, Software Engineering, Information Technology, e-Business and Applications (CSITeA)*. 2004: Cairo, Eypt.
5. Tsantalis, Nikololaos and Alexander Chatzigeorgiou. *Bad Smell Identification for Software Refactoring*. 2007 [cited 2012 September 15]; Available from: http://jdeodorant.com.
6. Mens, T. and T. Tourwe, *A Survey of Software Refactoring.* IEEE Transactions Software Engineering, 2004. **30**(2): p. 126-139.
7. Cormen, Leiserson and Rivest, *Introduction to Algorithms*. 1990.
8. Chidamber, S.R. and C.F. Kemerer, *A Metric Suit for Object-Oriented Design.* IEEE Transactions Software Engineering, 1994. **20**(6): p. 476-493.
9. Liu, Hui, Limei Yang, Zhendong Niu, Zhiyi Ma and Weizhong Shao, *Facilitating Software Refactoring with Appropriate Resolution Order of Bad Smells*, in *ESEC-FSE*. 2009: Amsterdam, Netherlands.
10. Mens, T., G. Taentzer and O. Runge, *Analysing Refactoring Dependencies Using Graph Transformation.* Software and System Modeling, 2007. **6**(3): p. 269-285.
11. Minhaz, F. Zibran, S. Chanchal and K. Roy, A Constraint Programming Approach to Conflict Aware Optimal Scheduling of Prioritized Code Clone Refactoring, in The 11th IEEE International Working Conference on Source Code Analysis and Manipulation. 2011. p. 105-114.
12. Piveta, Eduardo, Joao Araujo, Marcelo Pimenta, Ana Moreira, Pedro Guerreriro and R. Tom Price, *Searching for Opportunites of Refactoring Sequences : Reducing the Search Space.* Annual IEEE International Computer Software and Applications Conference, 2008: p. 319-326.

13. Sanjay, K.D and R. Ajay, *Assessment of Maintainability Metrics for Object-Oriented Software System.* ACM SIGSOFT Software Engineering Notes, 2011: p. 1-6.
14. Counsell, S. and E. Mendes, *Size and Frequence of Class Change from a Refactoring Perspective*, in *The 3rd IEEE Workshop on Software Evolvability*. 2007. p. 23-28.
15. Bansiya, J. and C.G. Davis, *A Hierarchical Model for Object-Oriented Design Quality Assessment.* IEEE Transactions on Software Engineering, 2002. **28**(1): p. 4-17.
16. Knuth and E. Donald, Artificial Intelligence: A Modern Approach 2nd edition, 1997: p. 590-597.
17. Russell, J. Stuar, Norvig and Peter, Artificial Intelligence: A Modern Approach 2nd edition, 2003: p. 111-114.
18. Delling, D., P. Sanders, D. Schultes and D. Wagner, *Engineering route planning algorithms.* Algorithmics of large and complex networks, 2009.

ภาคผนวก

ภาคผนวก ก
รายละเอียดการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด ด้วยอัลกอริทึมแบบอื่นๆ

1. อัลกอริทึมค้นหาแนวกว้าง (Breadth First Search) [16]

เป็นขั้นตอนวิธีในการท่องกราฟอย่างหนึ่ง โดยในขณะที่กำลังท่องกราฟมายังจุดยอดหนึ่งๆ นั้นจะมีการกระทำสองอย่างคือ เข้าเยี่ยมโหนดปัจจุบัน และเข้าถึงโหนดข้างเคียงของโหนดที่เข้าเยี่ยม โดยการพิจารณาโหนดข้างเคียงของโหนดที่ได้เลือกก่อนหน้านี้เพื่อที่จะสามารถย้อนเส้นกลับมาเส้นทางเดิมได้ในกรณีที่โหนดถัดไปที่ต่อจากโหนดปัจจุบันนั้นมีค่าผลลัพธ์ที่แย่กว่าโหนดข้างเคียงของโหนดที่ได้เลือกก่อนหน้านี้กลับไปยังเส้นทางที่มีผลลัพธ์ดีกว่า ขั้นตอนการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดด้วยอัลกอริทึมค้นหาแนวกว้าง มีดังตอนดังนี้

1. ประยุกต์ใช้งานวิธีรีแฟคทอริงแต่ละวิธีในการปรับแก้ไขโค้ดในแต่ละตำแหน่ง

2. คำนวณหาค่าความสามารถในการบำรุงรักษาซอฟต์แวร์สำหรับซอร์ซโค้ดที่ปรับแก้ไขด้วยวิธีรีแฟคทอริงในแต่ละแบบ เพื่อกำหนดให้เป็นค่าทรัพยากรที่ใช้ในการเดินไปยังแต่ละเส้นทาง

3. เลือกซอร์ซโค้ดหลังภายจากปรับแก้ไขด้วยวิธีรีแฟคทอริงที่มีค่าความสามารถในการบำรุงรักษาที่มากที่สุด โดยนำโหนดข้างเคียงของโหนดที่ได้เดินมาก่อนหน้านี้มาพิจารณาด้วย ให้ยกเว้นโหนดเริ่มต้นหรือรอบแรกของการค้นหา เพื่อนำมาเป็นซอร์ซโค้ดที่จะนำไปปรับแก้ไขสำหรับตำแหน่งที่เหลือต่อไป

4. ตรวจสอบว่ายังมีตำแหน่งที่ต้องปรับแก้ไขด้วยวิธีรีแฟคทอริงอีกหรือไม่

4.1 ถ้ายังมีตำแหน่งที่ต้องปรับแก้ไข ให้วนกลับไปทำขั้นตอนที่ 1 ถึง ขั้นตอนที่ 4 อีกครั้งจนกระทั้งไม่มีตำแหน่งที่ต้องปรับแก้ไขหรือไม่มีวิธีรีแฟคทอริงที่ใช้ปรับแก้ไข

4.2 ถ้าไม่มีตำแหน่งที่ต้องปรับแก้ไข ให้หยุดการค้นหา

5. ได้ลำดับการใช้งานวิธีรีแฟคทอริงที่ทำให้ซอร์ซโค้ดภายหลังจากปรับแก้ไขมีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ที่มากที่สุด

โดยจำนวนโหนดที่ต้องพิจารณาทั้งหมดในการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมค้นหาแนวกว้างหาได้จากสูตรดังนี้

จำนวนโหนดที่ต้องค้นหาทั้งหมด = ผลรวมของจำนวนวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขในรอบแรก ($\sum_{i=1}^{z} Pi$) + ผลรวมของจำนวนวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขในรอบที่สองเป็นต้นไป ($\sum_{r=2}^{z} \mathrm{Re}_r$)

กำหนดให้ :

- Re_r เป็นผลรวมของจำนวนวิธีรีแฟคทอริงที่ใช้ในการแก้ไขของแต่ละตำแหน่ง ($\sum_{i=1}^{z} Pi$) ของรอบการแก้ไขที่ r + (ผลรวมของจำนวนวิธีรีแฟคทอริงที่ต้องพิจารณารอบก่อนหน้า - 1)

- Pi เป็นจำนวนวิธีรีแฟคทอริงที่ใช้ในแก้ไขของตำแหน่งที่ i

- z เป็นจำนวนตำแหน่งที่จะปรับแก้ไข

- i เป็นตำแหน่งที่ปรับแก้ไข

- เมื่อสิ้นสุดการแก้ไขในตำแหน่งหนึ่งๆ จำนวนของตำแหน่งที่จะแก้ไขจะลดลง 1 ตำแหน่ง

2. อัลกอริทึมปีนเขา (Hill Climbing) [17]

อัลกอริทึมนี้เสมือนกับการปีนเขา ซึ่งจะค้นหาโดยเลือกโหนดที่มีค่าผลลัพธ์ดีกว่าโหนดที่เคยได้เลือกมา ทำให้การเดินไปยังโหนดถัดไปจะได้ค่าผลลัพธ์ที่ดีขึ้นเรื่อยๆ โดยจะไม่เดินย้อนกลับจุดเดิม และจะหยุดค้นหาเมื่อโหนดถัดไปนั้นไม่มีโหนดใดที่ให้ค่าผลลัพธ์ทีดีกว่าโหนดปัจจุบัน ขั้นตอนการจัดลำดับการใช้วิธีรีแฟคทอริงในการปรับแก้ไขโค้ดด้วยอัลกอริทึมปีนเขา มีดังนี้

1. ประยุกต์ใช้งานวิธีรีแฟคทอริงแต่ละวิธีในการปรับแก้ไขโค้ดในแต่ละตำแหน่ง

2. คำนวณหาค่าความสามารถในการบำรุงรักษาซอฟต์แวร์สำหรับซอร์ซโค้ดที่ปรับแก้ไขด้วยวิธีรีแฟคทอริงในแต่ละแบบ เพื่อกำหนดให้เป็นค่าทรัพยากรที่ใช้ในการเดินไปยังแต่ละเส้นทาง

3. เลือกซอร์ซโค้ดหลังภายจากปรับแก้ไขด้วยวิธีรีแฟคทอริงที่มีค่าความสามารถในการบำรุงรักษาที่มากที่สุดและต้องมากกว่าซอร์ซโค้ดหลังจากปรับแก้ไขในรอบที่แล้ว

4. ตรวจสอบว่ายังมีตำแหน่งที่ต้องปรับแก้ไขด้วยวิธีรีแฟคทอริงอีกหรือไม่และตำแหน่งต่างๆ นั้นมีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ที่ดีกว่าตำแหน่งปัจจุบันหรือไม่

4.1 ถ้ายังมีตำแหน่งที่ต้องปรับแก้ไข และตำแหน่งนั้นมีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ที่ดีกว่าตำแหน่งปัจจุบัน ให้วนกลับไปทำขั้นตอนที่ 1 ถึง ขั้นตอนที่ 4 อีกครั้งจนกระทั้งไม่มีตำแหน่งที่ต้องปรับแก้ไขหรือไม่มีวิธีรีแฟคทอริงที่ใช้ปรับแก้ไข หรือไม่มีตำแหน่งที่ปรับแก้ไขแล้วมีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ที่ดีกว่าตำแหน่งปัจจุบัน

4.2 ถ้าไม่มีตำแหน่งที่ต้องปรับแก้ไข หรือไม่มีตำแหน่งที่ปรับแก้ไขแล้วมีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ที่ดีกว่าตำแหน่งปัจจุบัน ให้หยุดการค้นหา

5. ได้ลำดับการใช้งานวิธีรีแฟคทอริงที่ทำให้ซอร์ซโค้ดภายหลังจากปรับแก้ไขมีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ที่มากที่สุด

จากขั้นตอนของการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดด้วยอัลกอริทึมปีนเขานั้น สามารถหยุดการค้นหาได้ถ้าไม่มีโหนดถัดไปที่มีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ที่

ดีกว่าโหนดปัจจุบัน จึงทำให้บางตำแหน่งอาจจะไม่ถูกปรับแก้ไขได้ ทำให้ซอร์ซโค้ดนั้นยังคงมีร่องรอยที่ผิดพลาดเหลืออยู่เมื่อสิ้นสุดการจัดลำดับ โดยจำนวนโหนดที่ต้องพิจารณาทั้งหมดในการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมปีนเขาหาได้จากสูตรดังนี้

จำนวนโหนดที่ต้องค้นหาทั้งหมด = ผลรวมของจำนวนวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขในแต่ละรอบ ($\sum_{r=1}^{z} \mathrm{Re}_r$)

กำหนดให้ :

- Re_r เป็นผลรวมของจำนวนวิธีรีแฟคทอริงที่ใช้ในการแก้ไขของแต่ละตำแหน่ง ($\sum_{i=1}^{z} Pi$) ของรอบการแก้ไขที่ r

- Pi เป็นจำนวนวิธีรีแฟคทอริงที่ใช้ในแก้ไขของตำแหน่งที่ i

- z เป็นจำนวนตำแหน่งที่จะปรับแก้ไข

- เมื่อสิ้นสุดการแก้ไขในตำแหน่งหนึ่งๆ จำนวนของตำแหน่งที่จะแก้ไขจะลดลง 1 ตำแหน่ง

3. อัลกอริทึมเอสตาร์ (A* - A Star) [18]

เป็นอัลกอริทึมที่ใช้ค้นหาเส้นทางที่ดีที่สุดและใช้ทรัพยากรน้อยที่สุดจากโหนดเริ่มต้นไปยังโหนดที่เป็นเป้าหมาย โดยการใช้ฟังก์ชันฮิวริสติกแบบค่าของระยะทาง (f(x)) ซึ่งเกิดจากผลรวมของสองฟังก์ชัน คือ ค่าระยะทางจากโหนดเริ่มต้นไปยังโหนดปัจจุบัน (g(x)) และค่าระยะทางจากโหนดปัจจุบันไปยังโหนดที่เป็นเป้าหมาย (h(x)) ในการใช้งานอัลกอริทึมเอสตาร์ในการเรียงลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดนั้น สามารถแทนรายละเอียดฟังก์ชันฮิวรีสติกต่างๆ ได้ดังนี้

$$f(x) = g(x) + h(x)$$

- f(x) เป็นฟังก์ชันฮิวริสติกที่บอกถึงค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ของซอร์ซโค้ดภายหลังจากปรับแก้ไขโค้ดทุกตำแหน่งแล้วของโหนดปัจจุบัน

- g(x) เป็นค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ของโหนดปัจจุบัน

- h(x) เป็นค่าเฉลี่ยความสามารถในการบำรุงรักษาซอฟต์แวร์ภายหลังจากการปรับแก้ไขของตำแหน่งอื่นๆ เพื่อแก้ไขให้ครบทุกตำแหน่ง

ขั้นตอนการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ดด้วยอัลกอริทึมเอสตาร์ มีดังนี้

1. ประยุกต์ใช้งานวิธีรีแฟคทอริงแต่ละวิธีในการปรับแก้ไขโค้ดในแต่ละตำแหน่ง

2. คำนวณหาค่าความสามารถในการบำรุงรักษาซอฟต์แวร์สำหรับซอร์ซโค้ดที่ปรับแก้ไขด้วยวิธีรีแฟคทอริงในแต่ละแบบที่ได้จากขั้นตอนที่ 1 เพื่อคำนวณหาค่า g(x)

3. ประยุกต์ใช้งานวิธีรีแฟคทอริงเพื่อปรับแก้ไขในตำแหน่งอื่นๆ ที่เหลือให้ครบทุกตำแหน่ง จากซอร์ซโค้ดที่ได้จากข้อ 1 แล้วคำนวณหาค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ของแต่ละซอร์ซโค้ดที่ได้แก้ไขครบทุกตำแหน่ง จากนั้นจึงนำมาหาค่าเฉลี่ยเพื่อคำนวณหาค่า h(x)

4. พิจารณาผลรวมของค่า g(x) กับค่า h(x) ของแต่ละตำแหน่งที่ได้จากข้อ 1 แล้วเลือกโหนดที่ให้มีค่าผลรวมมากที่สุดเป็นซอร์ซโค้ดที่จะนำไปปรับแก้ไขในตำแหน่งอื่นที่เหลือในรอบถัดไป

5. ตรวจสอบว่ายังมีตำแหน่งที่ต้องปรับแก้ไขด้วยวิธีรีแฟคทอริงอีกหรือไม่

5.1 ถ้ายังมีตำแหน่งที่ต้องปรับแก้ไข ให้วนกลับไปทำขั้นตอนที่ 1 ถึง ขั้นตอนที่ 5 อีกครั้งจนกระทั้งไม่มีตำแหน่งที่ต้องปรับแก้ไขหรือไม่มีวิธีรีแฟคทอริงที่ใช้ปรับแก้ไข

5.2 ถ้าไม่มีตำแหน่งที่ต้องปรับแก้ไข ให้หยุดการค้นหา

6. ได้ลำดับการใช้งานวิธีรีแฟคทอริงที่ทำให้ซอร์ซโค้ดภายหลังจากปรับแก้ไขมีค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ที่มากที่สุด

โดยจำนวนโหนดที่ต้องพิจารณาทั้งหมดในการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมเอสตาร์หาได้จากสูตรดังนี้

จำนวนโหนดที่ต้องค้นหาทั้งหมด = ผลรวมของจำนวนวิธีรีแฟคทอริงที่ใช้ในการปรับแก้ไขในแต่ละรอบ ($\sum_{r=1}^{z} \mathrm{Re}_r$)

กำหนดให้

- Re_r เป็นผลรวมของจำนวนโหนดทั้งหมดที่เกิดจากการปรับแก้ไขด้วยวิธีรีแฟคทอริงที่ใช้ปรับแก้ไขที่ตำแหน่งที่ Pi ร่วมกับที่ตำแหน่งอื่นๆ

- Pi เป็นจำนวนวิธีรีแฟคทอริงที่ใช้ในแก้ไขของตำแหน่งที่ i

- i เป็นตำแหน่งที่ปรับแก้ไข

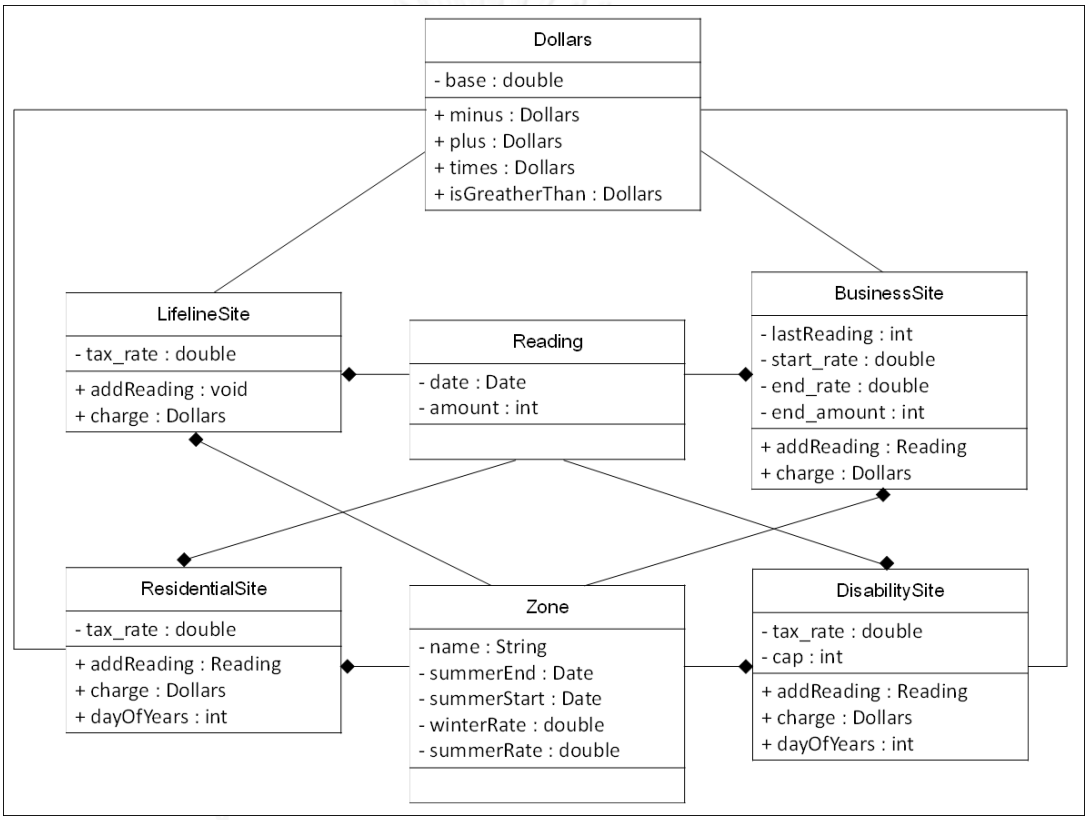
- z เป็นจำนวนตำแหน่งที่จะปรับแก้ไข

- เมื่อสิ้นสุดการแก้ไขในตำแหน่งหนึ่งๆ จำนวนของตำแหน่งที่จะแก้ไขจะลดลง 1 ตำแหน่ง

ภาคผนวก ข
รายละเอียดซอร์ซโค้ดระบบคำนวณค่าไฟฟ้า

ระบบคำนวณค่าไฟฟ้าเป็นระบบที่ใช้สำหรับการคำนวณค่าไฟฟ้าประจำรอบเดือนของลูกค้า ซึ่งระบบจะประกอบด้วยคลาสหลักจำนวน 7 คลาส ได้แก่ คลาส BusinessSite คลาส DisabilitySite คลาส Dollars คลาส LifelineSite คลาส Reading คลาส ResidentialSite คลาส Zone

แผนภาพความสัมพันธ์ของคลาสของระบบคำนวณค่าไฟฟ้า แสดงได้ดังภาพที่ 58



ภาพที่ 58 แผนภาพความสัมพันธ์ของคลาสของระบบเช่าภาพยนตร์

รายละเอียดของแต่ละคลาสมีดังนี้

1. คลาส Dollars

เป็นคลาสที่เก็บข้อมูลของจำนวนเงิน รายละเอียดของตัวแปรและเมท็อดของคลาสมีดังนี้

- ตัวแปร base ใช้สำหรับเก็บจำนวนเงิน
- เมท็อด minus ใช้สำหรับลดจำนวนเงิน
- เมท็อด plus ใช้สำหรับเพิ่มจำนวนเงิน
- เมท็อด times ใช้สำหรับการเพิ่มจำนวนเงินตามอัตราของภาษี

- เมท็อด isGreaterThan ใช้สำหรับเปรียบเทียบจำนวนเงิน

2. คลาส Reading

เป็นคลาสที่เก็บข้อมูลรายการการใช้ไฟฟ้าของผู้ใช้งาน รายละเอียดของตัวแปรและเมท็อดของคลาสมีดังนี้

- ตัวแปร date ใช้สำหรับเก็บวันที่บันทึกรายการการใช้ไฟฟ้า
- ตัวแปร amount ใช้สำหรับเก็บจำนวนหน่วยการใช้ไฟฟ้า

3. คลาส Zone

เป็นคลาสที่เก็บข้อมูลของย่าน รายละเอียดของตัวแปรของคลาสมีดังนี้

- ตัวแปร name ใช้สำหรับเก็บชื่อของย่าน
- ตัวแปร summerEnd ใช้สำหรับวันที่สิ้นสุดของหน้าร้อน
- ตัวแปร summerStart ใช้สำหรับเก็บวันที่เริ่มต้นของหน้าร้อน
- ตัวแปร winterRate ใช้สำหรับเก็บอัตราค่าไฟฟ้าของหน้าหนาว
- ตัวแปร summerRate ใช้สำหรับเก็บอัตราค่าไฟฟ้าของหน้าร้อน

4. คลาส LifeLine

เป็นคลาสใช้สำหรับคำนวณข้อมูลของย่านชนบท รายละเอียดของตัวแปรและเมท็อดของคลาสมีดังนี้

- ตัวแปร tax_rate ใช้สำหรับเก็บค่าภาษี
- เมท็อด addReading ใช้สำหรับเพิ่มรายการการใช้ไฟฟ้าของผู้ใช้งาน
- เมท็อด Charge ใช้สำหรับคำนวณค่าใช้ไฟฟ้า

5. คลาส ResidentialSite

เป็นคลาสใช้สำหรับคำนวณข้อมูลของย่านที่อยู่อาศัย รายละเอียดของตัวแปรและเมท็อดของคลาสมีดังนี้

- ตัวแปร tax_rate ใช้สำหรับเก็บค่าภาษี
- เมท็อด addReading ใช้สำหรับเพิ่มรายการการใช้ไฟฟ้าของผู้ใช้งาน
- เมท็อด Charge ใช้สำหรับคำนวณค่าใช้ไฟฟ้า
- เมท็อด dayOfYear ใช้สำหรับระบุค่าวันในรอบปี

6. คลาส BusinessSite

เป็นคลาสใช้สำหรับคำนวณข้อมูลของย่านธุระกิจ รายละเอียดของตัวแปรและเมท็อดของคลาสมีดังนี้

- ตัวแปร lastReading ใช้สำหรับเก็บค่าหน่วยการใช้ไฟฟ้าครั้งล่าสุด
- ตัวแปร start_rate ใช้สำหรับเก็บอัตราเริ่มต้น
- ตัวแปร end_rate ใช้สำหรับเก็บอัตราสิ้นสุด
- ตัวแปร end_amount ใช้สำหรับเก็บค่าจำนวนเงินสิ้นสุดการใช้ไฟฟ้า

7. คลาส DisabilitySite

เป็นคลาสใช้สำหรับคำนวณข้อมูลของย่านผู้พิการ รายละเอียดของตัวแปรและเมท็อดของคลาสมีดังนี้

- ตัวแปร tax_rate ใช้สำหรับเก็บค่าภาษี
- เมท็อด addReading ใช้สำหรับเพิ่มรายการการใช้ไฟฟ้าของผู้ใช้งาน
- เมท็อด Charge ใช้สำหรับคำนวณค่าใช้ไฟฟ้า
- เมท็อด dayOfYear ใช้สำหรับระบุค่าวันในรอบปี

1. รายละเอียดซอร์ซโค้ดก่อนการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ

1.1 คลาส DisabilitySite

```
01 public class DisabilitySite {
02
03        private Dollars charge(int fullUsage, Date start, Date end) {
04                Dollars result;
05                double summerFraction;
06                int usage = Math.min(fullUsage, CAP);
07                if(start.after(_zone.get_summerEnd()) || end.before(_zone.get_summerStart()))
08                        summerFraction = 0;
09                else if(!start.before(_zone.get_summerStart())
10                                && !start.after(_zone.get_summerEnd()) && !end.before(_zone.get_summerStart())
11                                && !end.after(_zone.get_summerEnd()))
12                        summerFraction = 1;
13                else{
14                        double summerDays;
15                        if(start.before(_zone.get_summerStart()) || start.after(_zone.get_summerEnd())){
16                                summerDays = dayOfYear(end) - dayOfYear(_zone.get_summerStart()) + 1;
17
18                        }else {
19                                summerDays = dayOfYear(_zone.get_summerEnd()) - dayOfYear(start) + 1;
20                        }
21                        summerFraction = summerDays/(dayOfYear(end) - dayOfYear(start) + 1);
22                }
23
24                result = new Dollars((usage * _zone.get_summerRate() * summerFraction) + (usage * _zone.get_winterRate() * (1-summerFraction)));
25                result = result.plus(new Dollars(Math.max(fullUsage - usage, 0) * 0.062));
26                result = result.plus(new Dollars(result.times(TAX_RATE)));
27                Dollars fuel = new Dollars(fullUsage * 0.0175);
28                result = result.plus(fuel);
29                result = new Dollars(result.plus(fuel.times(TAX_RATE)));
30                return result;
31
32        }
33
34}
```

ภาพที่ 59 ซอร์ซโค้ดเมท็อด Charge ของคลาส DisabilitySite ก่อนการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง

1.2 คลาส ResidentialSite

```
01 public class DisabilitySite {
02
03        private Dollars charge(int fullUsage, Date start, Date end) {
04                Dollars result;
05                double summerFraction;
06                int usage = Math.min(fullUsage, CAP);
07                if(start.after(_zone.get_summerEnd()) || end.before(_zone.get_summerStart()))
08                        summerFraction = 0;
09                else if(!start.before(_zone.get_summerStart())
10                                && !start.after(_zone.get_summerEnd()) && !end.before(_zone.get_summerStart())
11                                && !end.after(_zone.get_summerEnd()))
12                        summerFraction = 1;
13                else{
14                        double summerDays;
15                        if(start.before(_zone.get_summerStart()) || start.after(_zone.get_summerEnd())){
16                                summerDays = dayOfYear(end) - dayOfYear(_zone.get_summerStart()) + 1;
17
18                        }else {
19                                summerDays = dayOfYear(_zone.get_summerEnd()) - dayOfYear(start) + 1;
20                        }
21                        summerFraction = summerDays/(dayOfYear(end) - dayOfYear(start) + 1);
22                }
23
24                result = new Dollars((usage * _zone.get_summerRate() * summerFraction) + (usage * _zone.get_winterRate() * (1-summerFraction)));
25                result = result.plus(new Dollars(Math.max(fullUsage - usage, 0) * 0.062));
26                result = result.plus(new Dollars(result.times(TAX_RATE)));
27                Dollars fuel = new Dollars(fullUsage * 0.0175);
28                result = result.plus(fuel);
29                result = new Dollars(result.plus(fuel.times(TAX_RATE)));
30                return result;
31
32        }
33
34}
```

ภาพที่ 60 ซอร์ซโค้ดเมท็อด Charge ของคลาส ResidentialSite ก่อนการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริง

2. รายละเอียดซอร์ซโค้ดหลังการปรับแก้ไขด้วยการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ

2.1 คลาส DisabilitySite

```
01 public class DisabilitySite {
02
03        private Dollars charge(int fullUsage, Date start, Date end) {
04                Dollars result;
05                int usage = Math.min(fullUsage, CAP);
06                result = new Dollars((usage + _zone.get_summerRate() * summerFraction(start, end)) +
07                                (usage * _zone.get_winterRate() * (1-summerFraction(start, end))));
08                result = result.plus(new Dollars(Math.max(fullUsage - usage, 0) * 0.062));
09                result = result.plus(new Dollars(result.times(TAX_RATE)));
10                Dollars fuel = new Dollars(fullUsage * 0.0175);
11                result = result.plus(fuel);
12                result = new Dollars(result.plus(fuel.times(TAX_RATE)));
13                return result;
14
15        }
16
17        private double summerFraction(Date start, Date end){
18                double summerFraction;
19                if(start.after(_zone.get_summerEnd()) || end.before(_zone.get_summerStart()))
20                        summerFraction = 0;
21                else if(!start.before(_zone.get_summerStart())
22                                && !start.after(_zone.get_summerEnd()) && !end.before(_zone.get_summerStart())
23                                && !end.after(_zone.get_summerEnd()))
24                        summerFraction = 1;
25                else{
26                        double summerDays;
27                        if(start.before(_zone.get_summerStart()) || start.after(_zone.get_summerEnd())){
28                                summerDays = dayOfYear(end) - dayOfYear(_zone.get_summerStart()) + 1;
29
30                        }else {
31                                summerDays = dayOfYear(_zone.get_summerEnd()) - dayOfYear(start) + 1;
32                        }
33                        summerFraction = summerDays/(dayOfYear(end) - dayOfYear(start) + 1);
34                }
35
36                return summerFraction;
37
38        }
39
40}
```

ภาพที่ 61 ซอร์ซโค้ดเมท็อด Charge ของคลาส DisabilitySite ภายหลังจากปรับแก้ไขที่ตำแหน่งที่ P2 ด้วยวิธีรีแฟคทอริงแบบ R2

```
01 public class DisabilitySite {
02
03        private Dollars charge(int fullUsage, Date start, Date end) {
04                  Dollars result;
05                  int usage = Math.min(fullUsage, CAP);
06                  result = new Dollars((usage + _zone.get_summerRate() * summerFraction(start, end)) +
07                              (usage * _zone.get_winterRate() * (1-summerFraction(start, end))));
08                  result = result.plus(new Dollars(Math.max(fullUsage - usage, 0) * 0.062));
09                  result = result.plus(new Dollars(result.times(TAX_RATE)));
10                  Dollars fuel = new Dollars(fullUsage * 0.0175);
11                  result = result.plus(fuel);
12                  result = new Dollars(result.plus(result.plus(fuelCharge(fullUsage).times(TAX_RATE))));
13                  return result;
14
15        }
16
17        protected Dollars fuelCharge(int fullUsage){
18                  return new Dollars(fullUsage * FUEL_CHARGE_RATE);
19
20        }
21
22        private double summerFraction(Date start, Date end){
23                  double summerFraction;
24                  if(start.after(_zone.get_summerEnd()) || end.before(_zone.get_summerStart()))
25                              summerFraction = 0;
26                  else if(!start.before(_zone.get_summerStart())
27                              && !start.after(_zone.get_summerEnd()) && !end.before(_zone.get_summerStart())
28                              && !end.after(_zone.get_summerEnd()))
29                        summerFraction = 1;
30                  else{
31                        double summerDays;
32                        if(start.before(_zone.get_summerStart()) || start.after(_zone.get_summerEnd())){
33                              summerDays = dayOfYear(end) - dayOfYear(_zone.get_summerStart()) + 1;
34
35                        }else {
36                              summerDays = dayOfYear(_zone.get_summerEnd()) - dayOfYear(start) + 1;
37                        }
38                        summerFraction = summerDays/(dayOfYear(end) - dayOfYear(start) + 1);
39                  }
40
41                  return summerFraction;
42
43        }
44
45}
```

ภาพที่ 62 ซอร์ซโค้ดเมท็อด Charge ของคลาส DisabilitySite

ภายหลังจากปรับแก้ไขที่ตำแหน่งที่ P1 ด้วยวิธีรีแฟคทอริงแบบ R2 และ R1 ตามลำดับ

```
01 public class DisabilitySite {
02
03        protected static final double FUEL_CHARGE_RATE = 0.0175;
04
05        private Dollars charge(Date start, Date end) {
06                  Dollars result;
07                  int usage = Math.min(lastUsage(), CAP);
08                  result = new Dollars((usage + _zone.get_summerRate() * summerFraction(start, end)) +
09                              (usage * _zone.get_winterRate() * (1-summerFraction(start, end))));
10                  result = result.plus(new Dollars(Math.max(lastUsage() - usage, 0) * 0.062));
11                  result = result.plus(new Dollars(result.times(TAX_RATE)));
12                  Dollars fuel = new Dollars(lastUsage() * 0.0175);
13                  result = result.plus(fuel);
14                  result = new Dollars(result.plus(result.plus(fuelCharge(lastUsage()).times(TAX_RATE))));
15                  return result;
16
17        }
18
19        protected Dollars fuelCharge(int fullUsage){
20                  return new Dollars(fullUsage * FUEL_CHARGE_RATE);
21
22        }
23
24        private int lastUsage(){
25                  int i=0;
26                  while(_readings[i] != null) i++;
27                  return _readings[i-1].amount() - _readings[i-2].amount();
28
29        }
30
31        private double summerFraction(Date start, Date end){
32                  double summerFraction;
33                  if(start.after(_zone.get_summerEnd()) || end.before(_zone.get_summerStart()))
34                              summerFraction = 0;
35                  else if(!start.before(_zone.get_summerStart())
36                              && !start.after(_zone.get_summerEnd()) && !end.before(_zone.get_summerStart())
37                              && !end.after(_zone.get_summerEnd()))
38                        summerFraction = 1;
39                  else{
40                        double summerDays;
41                        if(start.before(_zone.get_summerStart()) || start.after(_zone.get_summerEnd())){
42                              summerDays = dayOfYear(end) - dayOfYear(_zone.get_summerStart()) + 1;
43
44                        }else {
45                              summerDays = dayOfYear(_zone.get_summerEnd()) - dayOfYear(start) + 1;
46                        }
47                        summerFraction = summerDays/(dayOfYear(end) - dayOfYear(start) + 1);
48                  }
49
50                  return summerFraction;
51
52        }
53
54 }
```

ภาพที่ 63 ซอร์ซโค้ดเมท็อด Charge ของคลาส DisabilitySite

ภายหลังจากปรับแก้ไขที่ตำแหน่งที่ P3 ด้วยวิธีรีแฟคทอริงแบบ R2 R1 และ R3 ตามลำดับ

2.2 คลาส ResidentialSite

```
01 public class ResidentialSite {
02
03        private Dollars charge(int usage, Date start, Date end){
04                Dollars result;
05                result = new Dollars((usage * _zone.get_summerRate() * summerFraction(start, end)) + (usage * _zone.get_winterRate() *
06                                (1 - summerFraction(start, end))));
07
08                result = result.plus(new Dollars(result.times(TAX_RATE)));
09                Dollars fuel = new Dollars(usage * 0.0175);
10                result = result.plus(fuel);
11                result = new Dollars(result.plus(fuelChargeTaxes(usage)));
12
13                return result;
14
15        }
16
17        private double summerFraction(Date start, Date end){
18                double summerFraction;
19
20                if((start.after(_zone.get_summerEnd())) || end.before(_zone.get_summerStart()))
21                        summerFraction = 0;
22                else if(!start.before(_zone.get_summerStart()) && !start.after(_zone.get_summerEnd()) &&
23                                !end.before(_zone.get_summerStart()) && !end.after(_zone.get_summerEnd()))
24                                        summerFraction = 1;
25                else{
26                        double summerDays;
27
28                        if(start.before(_zone.get_summerStart()) || start.after(_zone.get_summerEnd())){
29                                summerDays = dayOfYear(end) - dayOfYear(_zone.get_summerStart()) +1;
30
31                        }else{
32                                summerDays = dayOfYear(_zone.get_summerEnd()) - dayOfYear(start) +1;
33
34                        }
35
36                        summerFraction = summerDays/(dayOfYear(end) - dayOfYear(start) +1);
37
38                }
39                return summerFraction;
40        }
41
42        protected Dollars fuelChargeTaxes(int usage){
43                return new Dollars(fuelChargeTaxes(usage).times(TAX_RATE));
44
45        }
46
47}
```

ภาพที่ 64 ซอร์ซโค้ดเมท็อด Charge ของคลาส ResidentialSite ภายหลังจากปรับแก้ไขที่ตำแหน่งที่ P5 ด้วยวิธีรีแฟคทอริงแบบ R5

```
01 public class ResidentialSite {
02
03        private Dollars charge(int usage, Date start, Date end){
04                Dollars result;
05                result = new Dollars((usage * _zone.get_summerRate() * summerFraction(start, end)) + (usage * _zone.get_winterRate() *
06                                (1 - summerFraction(start, end))));
07
08                result = result.plus(new Dollars(result.times(TAX_RATE)));
09                result = result.plus(fuelCharge(usage));
10                result = new Dollars(result.plus(fuelChargeTaxes(usage)));
11
12                return result;
13
14        }
15
16        protected Dollars fuelChargeTaxes(int usage){
17                return new Dollars(fuelChargeTaxes(usage).times(TAX_RATE));
18
19        }
20
21        private double summerFraction(Date start, Date end){
22                double summerFraction;
23
24                if((start.after(_zone.get_summerEnd())) || end.before(_zone.get_summerStart()))
25                        summerFraction = 0;
26                else if(!start.before(_zone.get_summerStart()) && !start.after(_zone.get_summerEnd()) &&
27                                !end.before(_zone.get_summerStart()) && !end.after(_zone.get_summerEnd()))
28                                        summerFraction = 1;
29                else{
30                        double summerDays;
31
32                        if(start.before(_zone.get_summerStart()) || start.after(_zone.get_summerEnd())){
33                                summerDays = dayOfYear(end) - dayOfYear(_zone.get_summerStart()) +1;
34
35                        }else{
36                                summerDays = dayOfYear(_zone.get_summerEnd()) - dayOfYear(start) +1;
37
38                        }
39
40                        summerFraction = summerDays/(dayOfYear(end) - dayOfYear(start) +1);
41
42                }
43                return summerFraction;
44        }
45
46        protected Dollars fuelCharge(int fullUsage){
47                return new Dollars(fullUsage * FUEL_CHARGE_RATE);
48        }
49
50        protected static final double FUEL_CHARGE_RATE = 0.0175;
51
52}
```

ภาพที่ 65 ซอร์ซโค้ดเมท็อด Charge ของคลาส ResidentialSite ภายหลังจากปรับแก้ไขที่ตำแหน่งที่ P4 ด้วยวิธีรีแฟคทอริงแบบ R5 และ R4 ตามลำดับ

```
01 public class ResidentialSite {
02
03        private Dollars charge(Date start, Date end){
04                Dollars result;
05                result = new Dollars((lastUsage() * _zone.get_summerRate() * summerFraction(start, end)) + (lastUsage() * _zone.get_winterRate() *
06                                (1 - summerFraction(start, end))));
07
08                result = result.plus(new Dollars(result.times(TAX_RATE)));
09                result = result.plus(fuelCharge(lastUsage()));
10                result = new Dollars(result.plus(fuelChargeTaxes(lastUsage())));
11
12                return result;
13
14        }
15
16        protected Dollars fuelChargeTaxes(int usage){
17                return new Dollars(fuelChargeTaxes(usage).times(TAX_RATE));
18
19        }
20
21        private double summerFraction(Date start, Date end){
22                double summerFraction;
23
24                if((start.after(_zone.get_summerEnd())) || end.before(_zone.get_summerStart()))
25                        summerFraction = 0;
26                else if(!start.before(_zone.get_summerStart()) && !start.after(_zone.get_summerEnd()) &&
27                                !end.before(_zone.get_summerStart()) && !end.after(_zone.get_summerEnd()))
28                                        summerFraction = 1;
29                else{
30                        double summerDays;
31
32                        if(start.before(_zone.get_summerStart()) || start.after(_zone.get_summerEnd())){
33                                summerDays = dayOfYear(end) - dayOfYear(_zone.get_summerStart()) +1;
34
35                        }else{
36                                summerDays = dayOfYear(_zone.get_summerEnd()) - dayOfYear(start) +1;
37
38                        }
39
40                        summerFraction = summerDays/(dayOfYear(end) - dayOfYear(start) +1);
41
42                }
43                return summerFraction;
44        }
45
46        protected Dollars fuelCharge(int fullUsage){
47                return new Dollars(fullUsage * FUEL_CHARGE_RATE);
48        }
49
50        protected static final double FUEL_CHARGE_RATE = 0.0175;
51
52        private int lastUsage(){
53                int i=0;
54                while(_readings[i] != null) i++;
55                return _readings[i-1].amount() - _readings[i-2].amount();
56
57        }
58
59}
```

ภาพที่ 66 ซอร์ซโค้ดเมท็อด Charge ของคลาส ResidentialSite
ภายหลังจากปรับแก้ไขที่ตำแหน่งที่ P6 ด้วยวิธีรีแฟคทอริงแบบ R5 R4 และ R6 ตามลำดับ

3. รายละเอียดค่ามาตรวัดของการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ

3.1 คลาส DisabilitySite

ตารางที่ 62 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส DisabilitySite
สำหรับการปรับแก้ไขโค้ดรอบที่ 1

โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ค่าความซับซ้อนของเมคเคปของเมท็อด Charge	จำนวนบรรทัดของเมท็อด Charge	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
P1[R1]	31	9	23	0.5	0	1
P2[R2]	31	2	10	0.5	0	1
P3[R3]	32	9	25	0.5	0	1

ตารางที่ 63 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส DisabilitySite สำหรับการปรับแก้ไขโค้ดรอบที่ 2

โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ค่าความซับซ้อนของเมคเคปของเมท็อด Charge	จำนวนบรรทัดของเมท็อด Charge	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
P1[R2,R1]	32	2	10	0.5	0	1
P3[R2,R3]	33	2	10	0.5	0	1

ตารางที่ 64 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส DisabilitySite เมื่อสิ้นสุดการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ

โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ค่าความซับซ้อนของเมคเคปของเมท็อด Charge	จำนวนบรรทัดของเมท็อด Charge	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
D[R2,R1,R3]	33	2	10	0.5	0	1

3.2 คลาส ResidentialSite

ตารางที่ 65 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส ResidentialSite สำหรับการปรับแก้ไขโค้ดรอบที่ 1

โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ค่าความซับซ้อนของเมคเคปของเมท็อด Charge	จำนวนบรรทัดของเมท็อด Charge	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
P4[R4]	31	9	21	0.5	0	1
P5[R5]	32	2	8	0.5	0	1
P6[R6]	31	9	25	0.5	0	1

ตารางที่ 66 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส ResidentialSite สำหรับการปรับแก้ไขโค้ดรอบที่ 2

โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ค่าความซับซ้อนของเมคเคปของเมท็อด Charge	จำนวนบรรทัดของเมท็อด Charge	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
P4[R5,R4]	33	2	7	0.5	0	1
P6[R5,R6]	34	2	8	0.5	0	1

ตารางที่ 67 แสดงค่ามาตรวัดของเมท็อด Charge ของคลาส ResidentialSite เมื่อสิ้นสุดการจัดลำดับการใช้งานวิธีรีแฟคทอริงด้วยอัลกอริทึมละโมบ

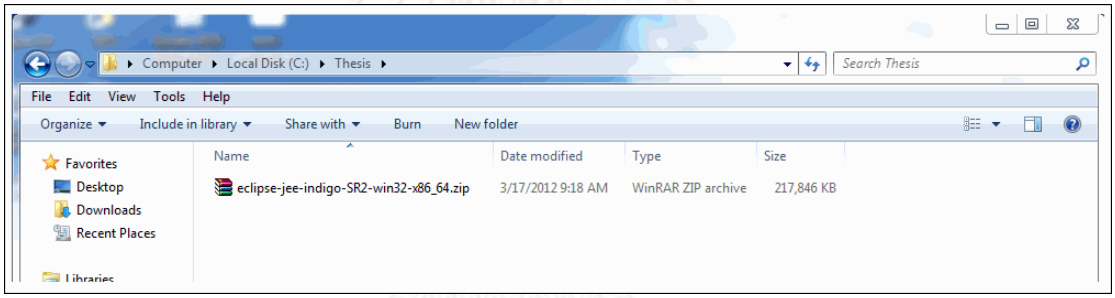
โหนด	ผลรวมค่าความซับซ้อนต่อคลาส	ค่าความซับซ้อนของเมคเคปของเมท็อด Charge	จำนวนบรรทัดของเมท็อด Charge	ระดับของการขาดการเกาะกันเป็นก้อนของเมท็อดภายในคลาส	แอฟเฟอร์เรนคัปปลิง	เอฟเฟอร์เรนคัปปลิง
R[R5,R4,R6]	35	2	7	0.5	0	1

ภาคผนวก ค
การติดตั้งเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด

เครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด ประกอบด้วยเครื่องมือ 4 ตัวที่ต้องทำการติดตั้งก่อนใช้งาน ได้แก่ โปรแกรมอีคลิปส์ ปลั๊กอินเจดีโอโดแรนต์ ปลั๊กอินเมทริก และเครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด ซึ่งรายละเอียดการติดตั้งแต่ละโปรแกรมมีรายละเอียดดังต่อไปนี้

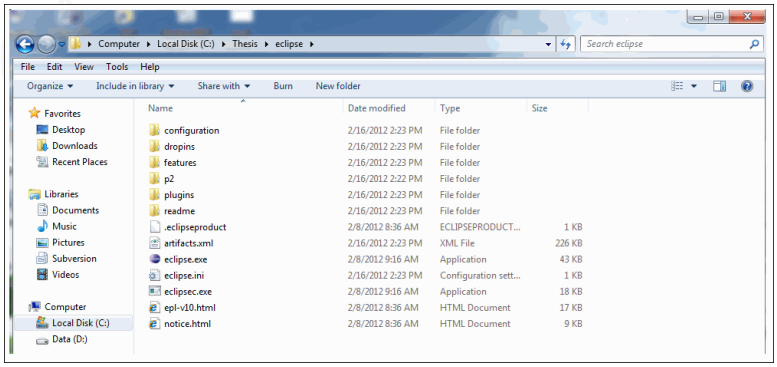
1. โปรแกรมอีคลิปส์

โปรแกรมอีคลิปส์สามารถติดตั้งได้โดยใช้วิธีการแตกซิปไฟล์ (Zip file) โดยผู้ใช้งานสารถดาวโหลดได้ที่เว็บไซต์ https://www.eclipse.org/downloads เลือกเวอร์ชันให้สนับสนุนกับระบบปฏิบัติการของเครื่องที่ใช้ ภายหลังจากที่ดาวโหลดเสร็จจะได้ซิปไฟล์ของโปรแกรมอีคลิปส์ดังภาพที่ 67



ภาพที่ 67 แสดงซิปไฟล์ของโปรแกรมอีคลิปส์ภายหลังจากดาวโหลดเสร็จสิ้น

เมื่อแตกซิปไฟล์ (Extract) จะได้โฟลเดอร์ชื่อ eclipse ซึ่งภายในประกอบด้วยโครงสร้างไฟล์ดังภาพที่ 68 โดยผู้ใช้งานสามารถเริ่มใช้งานโปรแกรมได้โดยการดับเบิ้ลคลิกที่ไฟล์ eclipse.exe ดังภาพที่ 69

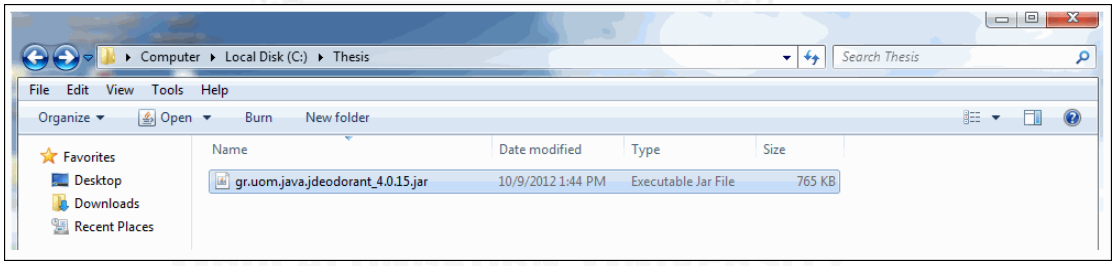


ภาพที่ 68 แสดงโครงสร้างไฟล์ภายหลังจากการแตกซิปไฟล์ของโปรแกรมอีคลิปส์

ภาพที่ 69 แสดงการเริ่มใช้งานโปรแกรมอีคลิปส์

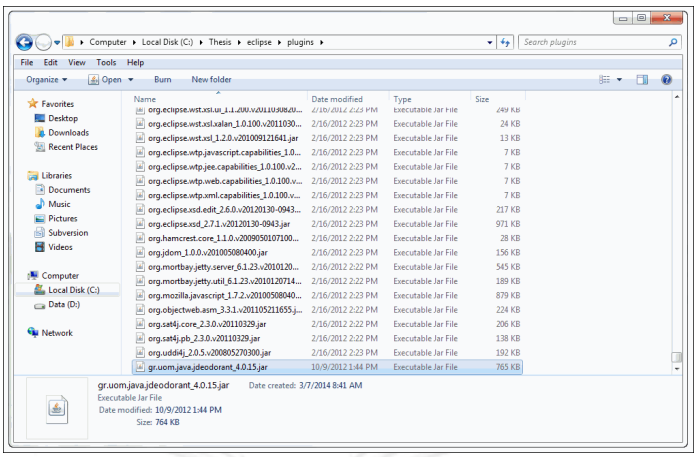
2. ปลั๊กอินเจดีโอโดแรนต์

ปลั๊กอินเจดีโอโดแรนต์เป็นปลั๊กอินที่สามารถใช้งานร่วมกับโปรแกรมอีคลิปส์ได้ โดยใช้วิธีการวางไฟล์จา (Jar file) ภายในโฟลเดอร์ plugin ของโปรแกรมอีคลิปส์ โดยผู้ใช้งานสามารถดาวโหลดได้ที่ http://www.jdeodorant.com เลือกเวอร์ชันของปลั๊กอินเจดีโอโดแรนต์ให้รองรับการใช้งานร่วมกับโปรแกรมอีคลิปส์ ภายหลังจากที่ดาวโหลดเสร็จจะได้ไฟล์จาของปลั๊กอิน ดังภาพที่ 70

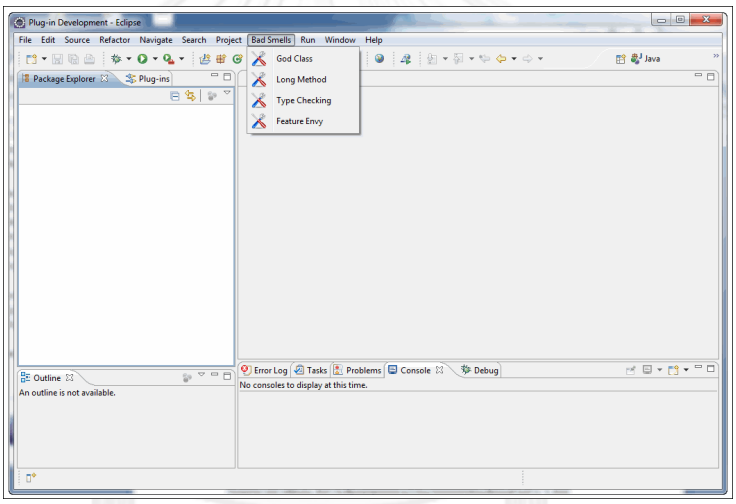


ภาพที่ 70 แสดงไฟล์จาของปลั๊กอินเจดีโอโดแรนต์ภายหลังจากดาวโหลดเสร็จสิ้น

จากนั้นจึงวางไฟล์จาของปลั๊กอินเจดีโอโดแรนต์ไว้ภายในโฟลเดอร์ plugin ของโปรแกรมอีคลิปส์ดังภาพที่ 71 จากนั้นจึงเปิดโปรแกรมอีคลิปส์เพื่อให้ทางโปรแกรมโหลดปลั๊กอินเจดีโอโดแรนต์มาไว้ที่ตัวเอง เพื่อพร้อมใช้งานปลั๊กอินดังภาพที่ 72



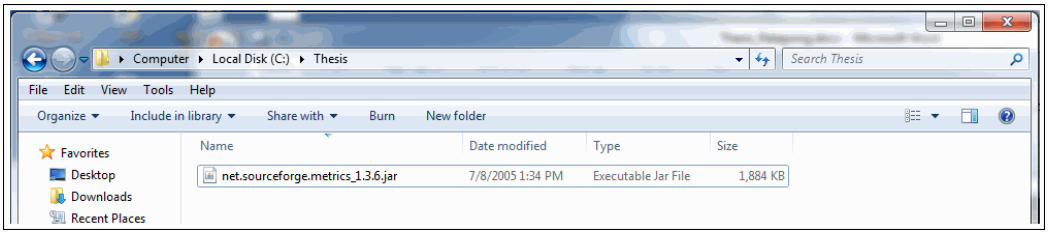
ภาพที่ 71 แสดงการวางไฟล์จาของปลั๊กอินเจดีโอโดแรนต์เพื่อใช้งานร่วมกับโปรแกรมอีคลิปส์



ภาพที่ 72 แสดงปลั๊กอินเจดีโอโดแรนต์พร้อมใช้งานร่วมกับโปรแกรมอีคลิปส์

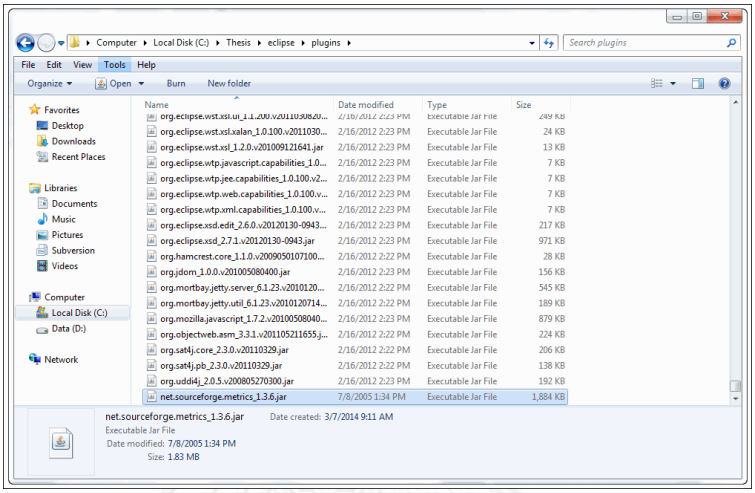
3. ปลั๊กอินเมทริก

ปลั๊กอินเมทิกเป็นปลั๊กอินที่สามารถใช้งานร่วมกับโปรแกรมอีคลิปส์ได้ โดยใช้วิธีการวางไฟล์จาเช่นเดียวกันกับปลั๊กอินเจดีโอโดแรนต์ คือ วางไฟล์ไว้ภายในโฟลเดอร์ plugin ของโปรแกรมอีคลิปส์ โดยผู้ใช้งานสามารถดาวโหลดได้ที่ http:// http://sourceforge.net/projects/metrics/ ภายหลังจากที่ดาวโหลดเสร็จจะได้ไฟล์จาของปลั๊กอิน ดังภาพที่ 73

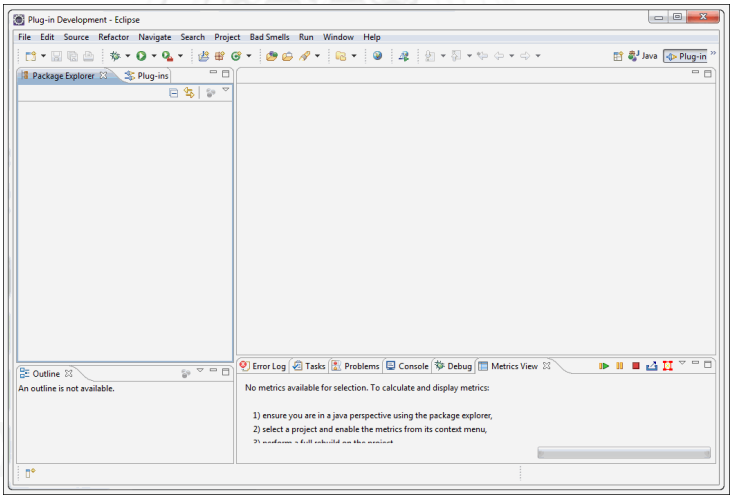


ภาพที่ 73 แสดงไฟล์จาของปลั๊กอินเมทริกภายหลังจากดาวโหลดเสร็จสิ้น

จากนั้นจึงวางไฟล์จาของปลั๊กอินเมทริกไว้ภายในโฟลเดอร์ plugin ของโปรแกรมอีคลิปส์ดังภาพที่ 74 จากนั้นจึงเปิดโปรแกรมอีคลิปส์เพื่อให้ทางโปรแกรมโหลดปลั๊กอินเมทริกมาไว้ที่ตัวเอง เพื่อพร้อมใช้งานปลั๊กอินดังภาพที่ 75



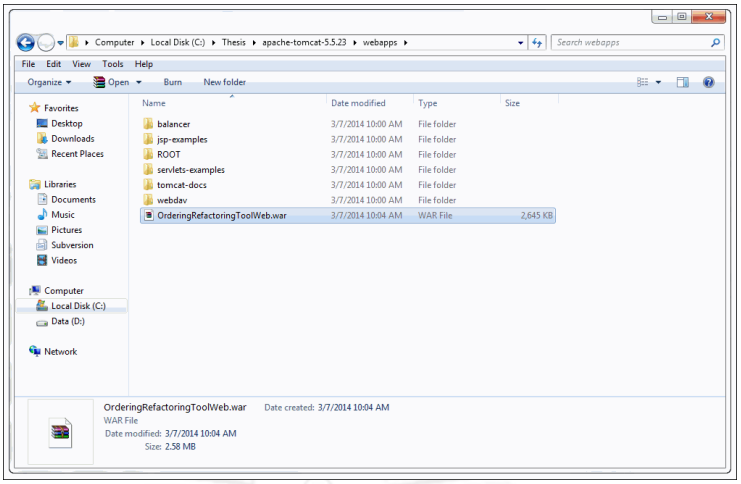
ภาพที่ 74 แสดงไฟล์จาของปลั๊กอินเมทริกภายหลังวางไว้ภายในโฟลเดอร์ plugin ของโปรแกรมอีคลิปส์



ภาพที่ 75 แสดงปลั๊กอินเมทริกพร้อมใช้งานร่วมกับโปรแกรมอีคลิปส์

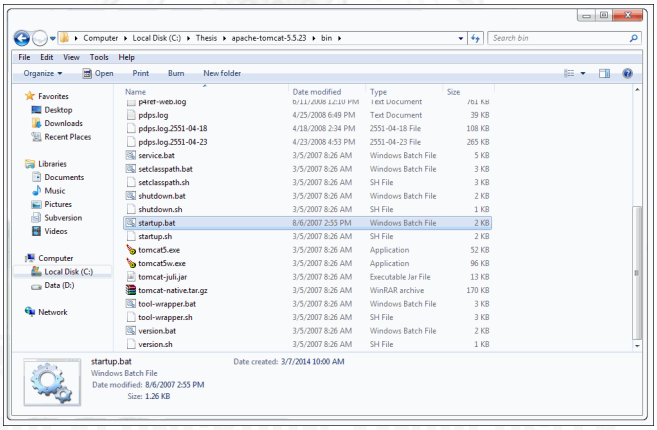
4. เครื่องมือช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริงในการปรับแก้ไขโค้ด

เป็นเครื่องมือที่งานวิจัยนี้ได้พัฒนาขึ้นในรูปแบบของเวปแอบพลิเคชั่น สามารถติดตั้งการทำงานผ่านทางเวบเซอร์เวอร์ (Web Server) ได้ โดยในงานวิจัยนี้ได้เลือกใช้งานเวบเซอร์เวอร์ของอาพาเช่ทอมแคท เวอร์ชัน 5.5 (Apache Tomcat 5.5) ในการรันการทำงาน โดยการนำเอาวอร์ไฟล์ของโปรแกรมที่ชื่อว่า OrderingRefactoringToolWeb.war ไปวางไว้ที่พาธ webapps ภายใต้โฟลเดอร์ของอาพาเช่ทอมแคท ดังภาพที่ 76

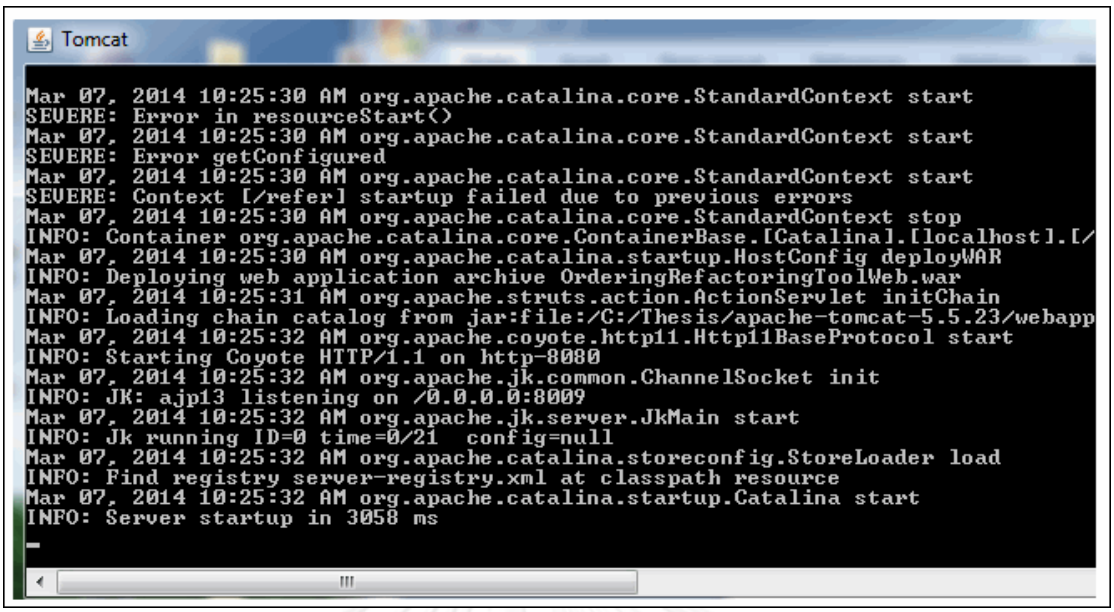


ภาพที่ 76 แสดงการวางไฟล์ OrderingRefactoringToolWeb.war ไว้ภายในเซอร์เวอร์อาพาเช่ทอมแคท

ภายหลังจากที่วางไฟล์เสร็จแล้ว จึงสามารถเริ่มการทำงานของเซอร์เวอร์อาพาเช่ทอมแคทได้ โดยดับเบิ้ลคลิกที่ไฟล์ startup.bat ที่พาธ bin ภายใต้โฟลเดอร์ของเซอร์เวอร์อาพาเช่ทอมแคทดังภาพที่ 77 หลังจากนั้นเซอร์เวอร์อาพาเช่คอมแคทจะแสดงหน้าจอรันการทำงาน เมื่อเซอร์เวอร์พร้อมทำงานจะแสดงดังภาพที่ 78

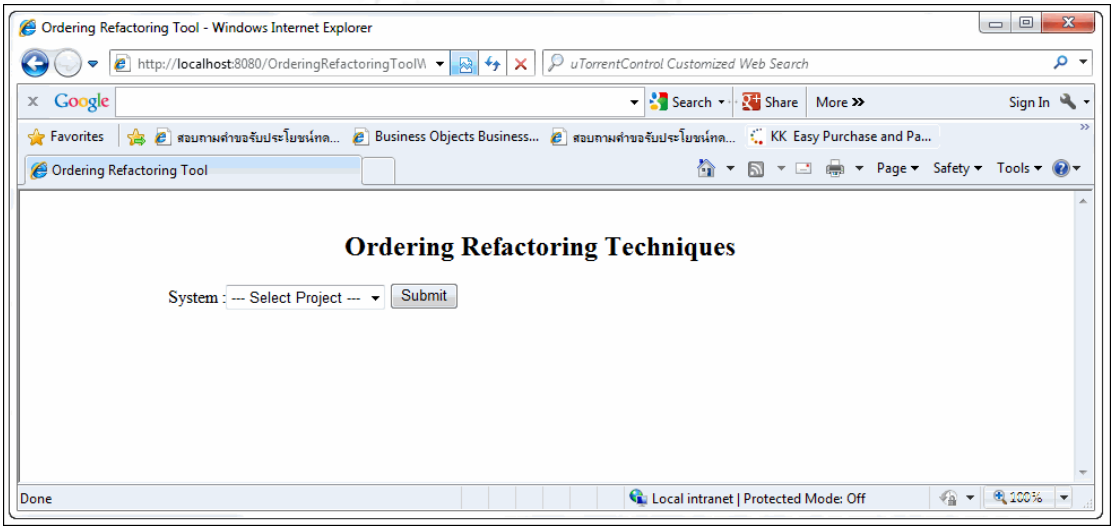


ภาพที่ 77 แสดงไฟล์ startup.bat สำหรับใช้ในการเริ่มการทำงานของเซอร์เวอร์อาพาเช่ทอมแคท



ภาพที่ 78 แสดงการรันการทำงานของเซอร์เวอร์อาพาเช่ทอมแคท

ภายหลังจากที่เซอร์เวอร์พร้อมใช้งานแล้ว ผู้ใช้งานสามารถเข้าสู่ระบบผ่านทางเวบเบอรเซอร์ได้โดยพิมพ์ยูอาร์แอล (Url) ดังนี้ http://localhost:8080/OrderingRefactotingTooWeb/ ระบบจะแสดงหน้าจอพร้อมใช้งานดังภาพที่ 79



ภาพที่ 79 แสดงระบบช่วยในการจัดลำดับการใช้งานวิธีรีแฟคทอริง ในการปรับแก้ไขโค้ดพร้อมใช้งาน

ประวัติผู้เขียนวิทยานิพนธ์

นายรัฐพงษ์ วงศ์เปียง เกิดเมื่อวันที่ 22 มิถุนายน พ.ศ. 2528 ที่จังหวัดกรุงเทพมหานคร สำเร็จการศึกษาระดับประถมศึกษาจากโรงเรียนพันธะวัฒนา จังหวัดกรุงเทพมหานคร สำเร็จการศึกษาระดับมัธยมศึกษาจากโรงเรียนมัธยมวัดเบญจมบพิตร กรุงเทพมหานคร สำเร็จการศึกษาปริญญาวิทยาศาสตร์บัณฑิต สาขาวิชาวิทยาศาสตร์ ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยธรรมศาสตร์ ในปีการศึกษา 2550 และเข้าศึกษาในหลักสูตรวิทยาศาสตร์มหาบัณฑิต สาขาวิศวกรรมซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2554