การตรวจจับร่องรอยที่ไม่ดีสำหรับรีแฟคทอริงโดยใช้มาตรวัดซอฟต์แวร์เชิงวัตถุ

นางสาวธิษณา เพียรเลิศ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2547

ISBN 974-17-6872-9

BAD-SMELL DETECTION FOR REFACTORING USING OBJECT-ORIENTED SOFTWARE METRICS

Miss Thisana Peanlert

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Computer Science
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2004

ISBN 974-17-6872-9

หัวข้อวิทยานิพนธ์	การตรวจจับร่องรอยที่ไม่ดีสำหรับรีแฟคทอริงโดยใช้มาตรวัดซอฟต์แวร์เชิงวัตถุ
โดย	นางสาวธิษณา เพียรเลิศ
สาขาวิชา	วิทยาศาสตร์คอมพิวเตอร์
อาจารย์ที่ปรึกษา	ผู้ช่วยศาสตราจารย์ ดร.พรศิริ หมื่นไชยศรี

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

…….. คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.ดิเรก ลาวัณย์ศิริ)

คณะกรรมการสอบวิทยานิพนธ์

…… ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.วิวัฒน์ วัฒนาวุฒิ)

……... อาจารย์ที่ปรึกษา
(ผู้ช่วยศาสตราจารย์ ดร.พรศิริ หมื่นไชยศรี)

……กรรมการ
(อาจารย์ ดร.ญาใจ ลิ่มปิยะกรณ์)

……กรรมการ
(อาจารย์นครทิพย์ พร้อมพูล)

นางสาวธิษณา เพียรเลิศ : การตรวจจับร่องรอยที่ไม่ดีสำหรับรีแฟคทอริงโดยใช้มาตรวัดซอฟต์แวร์เชิงวัตถุ. (BAD-SMELL DETECTION FOR REFACTORING USING OBJECT-ORIENTED SOFTWARE METRICS) อ. ที่ปรึกษา: ผศ.ดร.พรศิริ หมื่นไชยศรี, 202 หน้า. ISBN 974-17-6872-9.

วิทยานิพนธ์นี้นำเสนอวิธีการตรวจจับร่องรอยที่ไม่ดีสำหรับซอร์สโค้ด 6 ประเภท คือ Feature Envy, Large Class, Lazy Class, Long Method, Long Parameter Lists และ Switch Statement โดยใช้มาตรวัดซอฟต์แวร์เชิงวัตถุ ซึ่งมาตรวัดที่นำเสนอเหล่านี้ใช้ เพื่อช่วยตรวจสอบว่าส่วนใดของโค้ดเป็นร่องรอยที่ไม่ดี และวิธีการนี้ได้เสนอแนะวิธีการรีแฟคทอริง เพื่อแก้ไขร่องรอยที่ไม่ดีที่พบในซอร์สโค้ด จากนั้นทำการประเมินความสามารถของมาตรวัดร่องรอยที่ไม่ดี โดยเปรียบเทียบค่ามาตรวัดร่องรอยที่ไม่ดี ก่อนและหลังการประยุกต์ใช้วิธีรีแฟคทอริง ผู้วิจัยได้พัฒนาเครื่องมือสำหรับตรวจจับร่องรอยที่ไม่ดีของซอร์สโค้ดภาษาจาวา โดยคำนวณค่ามาตรวัดร่องรอยที่เกี่ยวข้องกับร่องรอยที่ไม่ดีทั้ง 6 ประเภท

ผลการทดสอบพบว่า ค่ามาตรวัดร่องรอยที่ไม่ดีนั้นดีขึ้น หลังจากประยุกต์ใช้วิธีรีแฟค-ทอริงแล้ว แสดงให้เห็นว่า มาตรวัดร่องรอยที่ไม่ดีที่เสนอในงานวิจัยนี้ ช่วยในการตรวจจับร่องรอยที่ไม่ดีได้

ภาควิชา.... วิศวกรรมคอมพิวเตอร์.....ลายมือชื่อนิสิต.....
สาขาวิชา....วิทยาศาสตร์คอมพิวเตอร์.....ลายมือชื่ออาจารย์ที่ปรึกษา.....
ปีการศึกษา .....2547.....

4570354221 : MAJOR COMPUTER SCIENCE

KEY WORD: OBJECT-ORIENTED SOFTWARE METRIC / REFACTORING / BAD-SMELL

THISANA PIENLERT: BAD-SMELL DETECTION FOR REFACTORING USING OBJECT-ORIENTED SOFTWARE METRICS.

THESIS ADVISOR: ASSISTANT PROFESSOR PORNSIRI MUENCHAISRI, PH.D., 202 pp. ISBN 974-17-6872-9.

This thesis proposes an approach for detecting and locating six bad-smells (such as Feature Envy, Large Class, Lazy Class, Long Method, Long Parameter Lists, and Switch Statement) from source code using object-oriented software metrics. The metrics are proposed to be used as indicators for determining whether a particular fraction of code contains the bad-smell or not. The approach also provides suggestion to modify the code by particular refactoring techniques, and then it is evaluated by comparing bad-smell metrics before and after applying the refactoring techniques. This research work also constructs an automated tool for detecting the bad-smell from java source code by measuring bad-smell metrics that are related to the six bad-smell.

The result shows that these bad-smell metrics are enhanced after we apply the refactoring. All proposed bad-smell metrics can be used as indicators for detecting and locating bad-smells.

Department…. Computer Engineering...Student's signature.....
Field of study…. Computer Science……Advisor's signature.....
Academic year …2004……

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยความช่วยเหลือจาก ผู้ช่วยศาสตราจารย์ ดร. พรศิริ หมื่นไชยศรี อาจารย์ที่ปรึกษาวิทยานิพนธ์ของข้าพเจ้า ขอกราบขอบพระคุณอาจารย์ที่ให้คำแนะนำ และข้อเสนอแนะต่างๆ ตลอดระยะเวลาของการทำวิทยานิพนธ์ของข้าพเจ้า จนสำเร็จลุล่วงได้ด้วยดี

ขอกราบขอบพระคุณผู้ช่วยศาสตราจารย์ ดร.วิวัฒน์ วัฒนาวุฒิ เป็นประธานกรรมการ อาจารย์ ดร.ญาใจ ลิ่มปิยะกรณ์ และอาจารย์นครทิพย์ พร้อมพูล เป็นกรรมการสอบวิทยานิพนธ์ ซึ่งได้สละเวลาและให้คำแนะนำต่างๆ ในการสอบวิทยานิพนธ์ของข้าพเจ้า

ท้ายที่สุดนี้ ขอกราบขอบพระคุณบิดา มารดา พี่ และขอบคุณเพื่อนๆ ทุกคนที่เป็นกำลังใจ และให้ความสนับสนุนมาโดยตลอด

สารบัญ

หน้า

บทคัดย่อภาษาไทย ..... ง

บทคัดย่อภาษาอังกฤษ ..... จ

กิตติกรรมประกาศ ..... ฉ

สารบัญ ..... ช

สารบัญภาพ ..... ญ

สารบัญตาราง ..... ฑ

บทที่ 1 บทนำ 1

1.1 ความเป็นมาและความสำคัญของปัญหา 1

1.2 วัตถุประสงค์ 2

1.3 ขอบเขตงานวิจัย 2

1.4 ขั้นตอนและวิธีดำเนินงานวิจัย 3

1.5 ประโยชน์ที่คาดว่าจะได้รับ 3

บทที่ 2 ทฤษฎีและงานวิจัยที่เกี่ยวข้อง 4

2.1 ทฤษฎีที่เกี่ยวข้อง 4

2.1.1 การวัดซอฟต์แวร์เชิงวัตถุ 4

2.1.2 รีแฟคทอริง 5

2.2 งานวิจัยที่เกี่ยวข้อง 8

2.2.1 งานวิจัย “Detecting Design Flaws via Metrics in Object-Oriented System” 8

2.2.2 งานวิจัย “A Quantitative Evaluation of Maintainability Enhancement by Refactoring” 12

2.2.3 งานวิจัย “A Systematic Class Refactoring Approach based on Metrics” 12

บทที่ 3 วิธีการตรวจจับร่องรอยที่ไม่ดี 15

3.1 การออกแบบวิธีการตรวจจับร่องรอยที่ไม่ดี 16

3.2 การออกแบบมาตรวัดร่องรอยที่ไม่ดี 17

บทที่ 4 การออกแบบและพัฒนาเครื่องมือช่วยในการตรวจจับร่องรอยที่ไม่ดี 34

4.1 แผนภาพยูสเคส 35

หน้า

4.2 แผนภาพคลาส 35
4.2.1 แพ็กเกจส่วนติดต่อกับผู้ใช้ 36
4.2.2 แพ็กเกจคอมไพเลอร์ 36
4.2.3 แพ็กเกจคำนวณค่าตัววัด 37
4.2.4 แพ็กเกจจัดรูปแบบการแสดงผลค่าตัววัด 38
4.3 แผนภาพซีเควนซ์ 38
4.3.1 แผนภาพซีเควนซ์ของกำหนดโปรแกรมต้นฉบับ 39
4.3.2 แผนภาพซีเควนซ์ของการสร้างพาร์เซอร์ 39
4.3.3 แผนภาพซีเควนซ์ของการคำนวณค่ามาตรวัด 40
4.3.4 แผนภาพซีเควนซ์ของการเรียกดูมาตรวัดร่องรอยที่ไม่ดีและวิธีการรีแฟคทอริง ของ Feature Envy 41
4.3.5 แผนภาพซีเควนซ์ของการเรียกดูมาตรวัดร่องรอยที่ไม่ดีและวิธีการรีแฟคทอริง ของ Large Class 41
4.3.6 แผนภาพซีเควนซ์ของการเรียกดูมาตรวัดร่องรอยที่ไม่ดีและวิธีการรีแฟคทอริง ของ Lazy Class 42
4.3.7 แผนภาพซีเควนซ์ของการเรียกดูมาตรวัดร่องรอยที่ไม่ดีและวิธีการรีแฟคทอริง ของ Long Method 42
4.3.8 แผนภาพซีเควนซ์ของการเรียกดูมาตรวัดร่องรอยที่ไม่ดีและวิธีการรีแฟคทอริง ของ Long Parameter Lists 42
4.3.9 แผนภาพซีเควนซ์ของการเรียกดูมาตรวัดร่องรอยที่ไม่ดีและวิธีการรีแฟคทอริง ของ Switch Statement 42
4.3.10 แผนภาพซีเควนซ์ของการเรียกดูค่ามาตรวัดร่องรอยที่ไม่ดีทั้งหมด 47
บทที่ 5 การทดสอบความน่าเชื่อถือของมาตรวัดร่องรอยที่ไม่ดี 48
5.1 การคำนวณค่ามาตรวัดร่องรอยที่ไม่ดี 48
5.1.1 ผลการคำนวณของโปรแกรมที่ 1 48
5.1.2 ผลการคำนวณของโปรแกรมที่ 2 57
5.1.3 ผลการคำนวณของโปรแกรมที่ 3 64
5.2 การประเมินความสามารถของมาตรวัดร่องรอยที่ไม่ดี 65
5.2.1 ผลการทดสอบโปรแกรมที่ 1 65
5.2.2 ผลการทดสอบโปรแกรมที่ 2 76

หน้า

5.2.3 ผลการทดสอบโปรแกรมที่ 3..... 106
5.3 การเลือกวิธีรีแฟคทอริง..... 110
5.4 การตรวจสอบค่ามาตรวัดที่ได้จากเครื่องมือ..... 110
บทที่ 6 บทสรุปและข้อเสนอแนะ..... 111
6.1 บทสรุป..... 111
6.2 ข้อเสนอแนะ..... 111
6.3 ผลงานตีพิมพ์..... 112
รายการอ้างอิง..... 113
ภาคผนวก..... 115
ภาคผนวก ก วิธีการรีแฟคทอริง.....114
ภาคผนวก ข ขั้นตอนการทำวิธีรีแฟคทอริงที่ใช้ในงานวิจัย.....120
ภาคผนวก ค ซอร์ดโค้ดของโปรแกรมที่นำมาทดสอบก่อนและหลังการทำรีแฟคทอริง.....147
ภาคผนวก ง ผลของค่ามาตรวัดร่องรอยที่ไม่ดีของโปรแกรมที่ 3.....162
ภาคผนวก จ การใช้งานเครื่องมือเพื่อช่วยในการตรวจจับร่องรอยที่ไม่ดี.....185
ภาคผนวก ฉ ผลงานตีพิมพ์.....195
ประวัติผู้เขียนวิทยานิพนธ์..... 202

สารบัญภาพ

หน้า

รูปที่ 2.1 กระบวนการรีแฟคทอริง 6
รูปที่ 3.1 แผนภาพขั้นตอนการวิจัย 15
รูปที่ 3.2 แผนภาพขั้นตอนตรวจจับร่องรอยที่ไม่ดี 16
รูปที่ 3.3 การเรียกใช้เมทธอดหรือคุณลักษณะภายในเมทธอดระหว่างคลาส *A* และ *B* ก่อนการประยุกต์ใช้วิธี Move Method 20
รูปที่ 3.4 การเรียกใช้เมทธอดหรือคุณลักษณะภายในเมทธอดระหว่างคลาส *A* และ *B* หลังการประยุกต์ใช้วิธี Move Method 21
รูปที่ 3.5 การเรียกใช้ตัวแปรอินสแตนท์จากเมทธอดภายในคลาส *A* 25
รูปที่ 3.6 การประยุกต์ใช้วิธี Extract Class 26
รูปที่ 3.7 การเรียกใช้เมทธอดและคุณลักษณะที่ตัวแปรอินสแตนท์ของคลาส *A* เรียกใช้ 26
รูปที่ 3.8 แผนภาพคลาสหลังการประยุกต์ใช้วิธี Extract Subclass 27
รูปที่ 4.1 แผนภาพยูสเคสของเครื่องมือช่วยในการตรวจจับร่องรอยที่ไม่ดี 34
รูปที่ 4.2 แผนภาพคลาสแสดงความสัมพันธ์ระหว่างแพ็กเกจต่าง ๆ ของระบบ 35
รูปที่ 4.4 แผนภาพคลาสของแพ็กเกจคอมไพเลอร์ 37
รูปที่ 4.6 แผนภาพคลาสของแพ็กเกจจัดรูปแบบการแสดงผลค่าตัววัด 38
รูปที่ 4.7 แผนภาพซีเควนซ์ของการกำหนดโปรแกรมต้นฉบับ 39
รูปที่ 4.8 แผนภาพซีเควนซ์ของการสร้างพาร์เซอร์ 40
รูปที่ 4.9 แผนภาพซีเควนซ์ของการคำนวณค่ามาตรวัด 41
รูปที่ 4.10 แผนภาพซีเควนซ์ของการเรียกดูค่ามาตรวัด และวิธีรีแฟคทอริงของ Feature Envy ... 43
รูปที่ 4.11 แผนภาพซีเควนซ์ของการเรียกดูค่ามาตรวัด และวิธีรีแฟคทอริงของ Large Class 44
รูปที่ 4.12 แผนภาพซีเควนซ์ของการเรียกดูค่ามาตรวัด และวิธีรีแฟคทอริงของ Lazy Class 45
รูปที่ 4.13 แผนภาพซีเควนซ์ของการเรียกดูค่ามาตรวัด และวิธีรีแฟคทอริงของ Long Method .. 45
รูปที่ 4.14 แผนภาพซีเควนซ์ของการเรียกดูค่ามาตรวัด และวิธีรีแฟคทอริงของ Long Parameter Lists 46
รูปที่ 4.15 แผนภาพซีเควนซ์ของการเรียกดูค่ามาตรวัด และวิธีรีแฟคทอริงของ Switch Statement 46
รูปที่ 4.16 แผนภาพซีเควนซ์ของการเรียกดูค่ามาตรวัดสำหรับคลาส เมทธอด และคุณลักษณะ ของโปรแกรม 47

หน้า

รูปที่ 5.1 แผนภาพคลาสของโปรแกรมที่ 1 49
รูปที่ 5.2 ซอร์สโค้ดของโปรแกรมที่ 1 ก่อนการทำรีแฟคทอริง 50
รูปที่ 5.3 ซอร์สโค้ดของโปรแกรมที่ 1 หลังการประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 1 55
รูปที่ 5.4 ซอร์สโค้ดของโปรแกรมที่ 1 หลังการประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 2 56
รูปที่ 5.5 ซอร์สโค้ดของโปรแกรมที่ 1 หลังการประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 3 56
รูปที่ 5.6 ซอร์สโค้ดของโปรแกรมที่ 1 หลังการประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 4 57
รูปที่ 5.7 แผนภาพคลาสของโปรแกรมที่ 2 58
รูปที่ ข.1 แผนภาพคลาสแสดงตัวอย่างการทำวิธี Collapse Hierarchy 122
รูปที่ ข.2 ตัวอย่างโค้ดก่อนการทำวิธี Decompose Conditional 123
รูปที่ ข.3 ตัวอย่างโค้ดหลังการทำวิธี Decompose Conditional 123
รูปที่ ข.4 ตัวอย่างโค้ดก่อนการทำวิธี Duplicate Observed Data 123
รูปที่ ข.5 ตัวอย่างโค้ดหลังการทำวิธี Duplicate Observed Data 125
รูปที่ ข.6 ตัวอย่างโค้ดก่อนการทำวิธี Extract Class 126
รูปที่ ข.7 ตัวอย่างโค้ดหลังการทำวิธี Extract Class 126
รูปที่ ข.8 ตัวอย่างโค้ดก่อนการทำวิธี Extract Interface 127
รูปที่ ข.9 ตัวอย่างโค้ดหลังการทำวิธี Extract Interface 127
รูปที่ ข.10 ตัวอย่างโค้ดก่อนการทำวิธี Extract Method 127
รูปที่ ข.11 ตัวอย่างโค้ดหลังการทำวิธี Extract Method 128
รูปที่ ข.12 ตัวอย่างโค้ดก่อนการทำวิธี Extract Subclass 128
รูปที่ ข.13 ตัวอย่างโค้ดหลังการทำวิธี Extract Subclass 130
รูปที่ ข.14 ตัวอย่างโค้ดก่อนการทำวิธี Inline Class 131
รูปที่ ข.15 ตัวอย่างโค้ดหลังการทำวิธี Inline Class 131
รูปที่ ข.16 ตัวอย่างโค้ดก่อนการทำวิธี Introduce Null Object 131
รูปที่ ข.17 ตัวอย่างโค้ดหลังการทำวิธี Introduce Null Object 133
รูปที่ ข.18 ตัวอย่างโค้ดก่อนการทำวิธี Introduce Parameter Object 133
รูปที่ ข.19 ตัวอย่างโค้ดหลังการทำวิธี Introduce Parameter Object 135
รูปที่ ข.20 ตัวอย่างโค้ดก่อนการทำวิธี Move Attribute 135
รูปที่ ข.21 ตัวอย่างโค้ดหลังการทำวิธี Move Attribute 136
รูปที่ ข.22 ตัวอย่างโค้ดก่อนการทำวิธี Move Method 137
รูปที่ ข.23 ตัวอย่างโค้ดหลังการทำวิธี Move Method 138

หน้า

รูปที่ ข.24 ตัวอย่างโค้ดก่อนการทำวิธี Preserve Whole Object 138
รูปที่ ข.25 ตัวอย่างโค้ดหลังการทำวิธี Preserve Whole Object..... 139
รูปที่ ข.26 ตัวอย่างโค้ดก่อนการทำวิธี Replace Conditional with Polymorphism..... 139
รูปที่ ข.27 แผนภาพคลาสโครงสร้างการสืบทอดคุณสมบัติ 140
รูปที่ ข.28 ตัวอย่างโค้ดก่อนการทำวิธี Replace Method with Method Object..... 140
รูปที่ ข.29 ตัวอย่างโค้ดหลังการทำวิธี Replace Method with Method Object..... 141
รูปที่ ข.30 ตัวอย่างโค้ดก่อนการทำวิธี Replace Parameter with Explicit Method 142
รูปที่ ข.31 ตัวอย่างโค้ดหลังการทำวิธี Replace Parameter with Explicit Method..... 142
รูปที่ ข.32 ตัวอย่างโค้ดก่อนการทำวิธี Replace Parameters with Method 142
รูปที่ ข.33 ตัวอย่างโค้ดหลังทำวิธี Replace Parameters with Method 143
รูปที่ ข.34 ตัวอย่างโค้ดก่อนทำวิธี Replace Temp with Query 143
รูปที่ ข.35 ตัวอย่างโค้ดหลังจากทำวิธี Replace Temp with Query..... 144
รูปที่ ข.36 ตัวอย่างโค้ดก่อนการทำวิธี Replace Type Code with State/Strategy 144
รูปที่ ข.37 ตัวอย่างโค้ดหลังการทำ Replace Type Code with State/Strategy..... 146
รูปที่ ข.38 ตัวอย่างโค้ดก่อนการทำวิธี Replace Type Code with Subclasses 147
รูปที่ ข.39 ตัวอย่างโค้ดหลังการทำวิธี Replace Type Code with Subclasses..... 148
รูปที่ ค.1 แผนภาพคลาสแสดงความสัมพันธ์ระหว่างแพ็กเกจต่าง ๆ ของโปรแกรมที่ 3 160
รูปที่ ค.2 แผนภาพคลาสแสดงความสัมพันธ์ภายในแพ็กเกจแสดงผล 160
รูปที่ ค.3 แผนภาพคลาสแสดงความสัมพันธ์ภายในแพ็กเกจแบบจำลอง..... 161
รูปที่ ค.4 แผนภาพคลาสแสดงความสัมพันธ์ภายในแพ็กเกจควบคุม..... 161
รูปที่ ค.5 แผนภาพคลาสแสดงความสัมพันธ์ภายในแพ็กเกจเหตุการณ์..... 161
รูปที่ ค.6 แผนภาพคลาสแสดงความสัมพันธ์ภายในแพ็กเกจเหตุการณ์..... 163
รูปที่ จ.1 เฟรมเพื่อเลือกโปรแกรมต้นฉบับจาวา 187
รูปที่ จ.2 ทรีและโหนดต่างๆ ที่ได้จากการเลือกโปรแกรมต้นฉบับ 188
รูปที่ จ.3 ค่ามาตรวัดร่องรอยที่ไม่ดีของ Feature Envy สำหรับเมทธอด..... 189
รูปที่ จ.4 ค่ามาตรวัดร่องรอยที่ไม่ดีของ Feature Envy สำหรับคุณลักษณะ..... 189
รูปที่ จ.5 ค่าวิธีรีแฟคทอริงของ Feature Envy 190
รูปที่ จ.6 ค่ามาตรวัดร่องรอยที่ไม่ดีของ Large Class..... 190
รูปที่ จ.7 วิธีรีแฟคทอริงของ Large Class 191
รูปที่ จ.8 ค่ามาตรวัดร่องรอยที่ไม่ดีของ Lazy Class..... 191

หน้า

รูปที่ จ.9 วิธีรีแฟคทอริงของ Lazy Class 192
รูปที่ จ.10 ค่ามาตรวัดร่องรอยที่ไม่ดีของ Long Method 192
รูปที่ จ.11 วิธีรีแฟคทอริงของ Long Method 193
รูปที่ จ.12 ค่ามาตรวัดร่องรอยที่ไม่ดีของ Long Parameter Lists 193
รูปที่ จ.13 วิธีรีแฟคทอริงของ Long Parameter Lists 194
รูปที่ จ.14 ค่ามาตรวัดร่องรอยที่ไม่ดีของ Switch Statement 194
รูปที่ จ.15 วิธีรีแฟคทอริงของ Switch Statement 195
รูปที่ จ.16 ค่ามาตรวัดร่องรอยที่ไม่ดีของคลาส 195
รูปที่ จ.17 ค่ามาตรวัดร่องรอยที่ไม่ดีของเมทธอด 196
รูปที่ จ.18 ค่ามาตรวัดร่องรอยที่ไม่ดีของคุณลักษณะ 196

สารบัญตาราง

หน้า

ตารางที่ 2.1 ความสัมพันธ์ระหว่างวิธีรีแฟคทอริงและค่ามาตรวัด 14
ตารางที่ 5.1 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 1 50
ตารางที่ 5.2 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 1 52
ตารางที่ 5.3 ค่ามาตรวัดสำหรับคุณลักษณะของโปรแกรมที่ 1 53
ตารางที่ 5.4 ประเภทร่องรอยที่ไม่ดีที่พบ และวิธีรีแฟคทอริงที่เสนอของโปรแกรมที่ 1 54
ตารางที่ 5.5 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 58
ตารางที่ 5.6 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 59
ตารางที่ 5.7 ประเภทร่องรอยที่ไม่ดีที่พบ และวิธีรีแฟคทอริงที่เสนอของโปรแกรมที่ 2 61
ตารางที่ 5.8 ประเภทร่องรอยที่ไม่ดีที่พบ และวิธีรีแฟคทอริงที่เสนอของโปรแกรมที่ 3 65
ตารางที่ 5.9 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 1หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 1 .. 65
ตารางที่ 5.10 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 1หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 1 66
ตารางที่ 5.11 ค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคุณลักษณะของโปรแกรมที่ 1 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 1 67
ตารางที่ 5.12 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 1หลังประยุกต์ใช้รีแฟคทอริงกรณีที่ 2 68
ตารางที่ 5.13 ค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคุณลักษณะของโปรแกรมที่ 1 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 2 68
ตารางที่ 5.14 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 1หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 2 69
ตารางที่ 5.15 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 1หลังประยุกต์ใช้รีแฟคทอริงกรณีที่ 3 70
ตารางที่ 5.16 ค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคุณลักษณะของโปรแกรมที่ 1 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 3 70
ตารางที่ 5.17 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 1หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 3 71
ตารางที่ 5.18 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 1หลังประยุกต์ใช้รีแฟคทอริงกรณีที่ 4 72
ตารางที่ 5.19 ค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคุณลักษณะของโปรแกรมที่ 1 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 4 72

หน้า

ตารางที่ 5.20 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 1หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 4 73
ตารางที่ 5.21 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 1 หลังประยุกต์ใช้รีแฟคทอริงทั้ง 4 กรณีตามลำดับ 74
ตารางที่ 5.22 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 1หลังประยุกต์ใช้รีแฟคทอริงทั้ง 4 กรณีตามลำดับ 75
ตารางที่ 5.23 ค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคุณลักษณะของโปรแกรมที่ 1 หลังประยุกต์ใช้รีแฟคทอริงทั้ง 4 กรณีตามลำดับ 76
ตารางที่ 5.24 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 1 และ 2 77
ตารางที่ 5.25 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 1 และ 2 79
ตารางที่ 5.26 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 3 และ 4 79
ตารางที่ 5.27 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 3 และ 4 80
ตารางที่ 5.28 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงกรณีที่ 5... 82
ตารางที่ 5.29 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 5 83
ตารางที่ 5.30 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงกรณีที่ 6... 85
ตารางที่ 5.31 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 6 86
ตารางที่ 5.32 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงกรณีที่ 7... 88
ตารางที่ 5.33 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 7 89
ตารางที่ 5.34 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงกรณีที่ 8... 91
ตารางที่ 5.35 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 8 92
ตารางที่ 5.36 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงกรณีที่ 9... 94

หน้า

ตารางที่ 5.37 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 9 95
ตารางที่ 5.38 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงกรณี 10 ... 97
ตารางที่ 5.39 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 10 98
ตารางที่ 5.40 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงกรณี 11 . 100
ตารางที่ 5.41 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 11 101
ตารางที่ 5.42 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงทั้ง 11 กรณี 103
ตารางที่ 5.43 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงทั้ง 11 กรณี 104
ตารางที่ 5.44 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 3 หลังประยุกต์ใช้รีแฟคทอริงกรณีที่ 1. 106
ตารางที่ 5.45 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 3 หลังประยุกต์ใช้รีแฟคทอริงกรณีที่ 2. 107
ตารางที่ 5.46 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 3 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 1 และ 2 109
ตารางที่ ง.1 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 3 164
ตารางที่ ง.2 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 3 165
ตารางที่ ง.3 ค่ามาตรวัดสำหรับคุณลักษณะของโปรแกรมที่ 3 182
ตารางที่ ง.4 ประเภทร่องรอยที่ไม่ดีที่พบในโปรแกรมที่ 3 และวิธีรีแฟคทอริงที่เสนอ 184

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ขั้นตอนการบำรุงรักษาซอฟต์แวร์เป็นกิจกรรมที่เกิดขึ้นหลังจากการพัฒนาซอฟต์แวร์เสร็จ ซึ่งประกอบด้วย การแก้ไขข้อผิดพลาดที่เกิดขึ้น (Corrective) การปรับเปลี่ยนซอฟต์แวร์ให้เข้ากับสภาพแวดล้อมใหม่ (Adaptive) การปรับปรุงฟังก์ชันการทำงานให้สมบูรณ์ขึ้น (Perfective) และการป้องกันและรักษาความน่าเชื่อถือของซอฟต์แวร์ (Preventive) หรือวิศวกรรมทำซ้ำซอฟต์แวร์ (Software Reengineering)

เนื่องจากซอฟต์แวร์ที่พัฒนาเสร็จแล้ว ต้องมีการปรับปรุงแก้ไข เพื่อรองรับการเปลี่ยนแปลงหรือเพิ่มเติมความต้องการของผู้ใช้งานที่มีอยู่ตลอดเวลา ดังนั้นจึงต้องมีการปรับปรุงประสิทธิภาพซอฟต์แวร์ หรือวิศวกรรมทำซ้ำซอฟต์แวร์ ซึ่งเป็นการออกแบบและพัฒนาซอฟต์แวร์ใหม่ ตามความต้องการของระบบเดิม โดยไม่ได้เพิ่มฟังก์ชัน (Function) ภายในระบบ โดยสามารถทำได้หลายวิธี วิธีหนึ่งที่สามารถทำในขั้นตอนนี้ได้เรียกว่า การรีแฟคทอริง (Refactoring) [1]

รีแฟคทอริงเป็นการเปลี่ยนแปลงโครงสร้างภายในของซอฟต์แวร์ ที่ไม่ส่งผลกระทบทำให้พฤติกรรมของซอฟต์แวร์เปลี่ยนแปลงไป การรีแฟคทอริงอย่างเหมาะสม จะช่วยให้การทำความเข้าใจและการบำรุงรักษาซอฟต์แวร์ทำได้ง่ายขึ้น การออกแบบซอฟต์แวร์ดีขึ้น หาข้อผิดพลาดของซอฟต์แวร์ ช่วยให้ซอฟต์แวร์ทำงานได้เร็วขึ้น [2]

บางครั้งในการพัฒนาซอฟต์แวร์เชิงวัตถุในเวอร์ชันแรกของซอฟต์แวร์ จะมีข้อจำกัดทางด้านเวลา และทรัพยากร ผู้พัฒนาจึงมักทำได้เพียงพัฒนาโปรแกรมให้มีฟังก์ชันครบตามความต้องการของผู้ใช้ เพื่อส่งมอบให้ลูกค้าตามกำหนด ดังนั้นในการพัฒนาซอฟต์แวร์เวอร์ชันแรก ผู้พัฒนาอาจจะไม่คำนึงถึงวิธีการที่ทำให้ซอฟต์แวร์บำรุงรักษาได้ง่าย วิธีการรีแฟคทอริงจึงถูกประยุกต์ใช้จริงในโปรแกรมเวอร์ชันถัดไป แต่เนื่องจากการทำรีแฟคทอริงมีกระบวนการที่ซับซ้อน ซึ่งขั้นตอนที่สำคัญคือ การตรวจจับร่องรอยที่ไม่ดี เพื่อที่จะทำให้ทราบว่า ร่องรอยการออกแบบที่ไม่ดีนั้นเป็นอย่างไร และอยู่ที่ไหน ร่องรอยที่ไม่ดี (Bad-smell) หมายถึง ลักษณะของการออกแบบซอฟต์แวร์หรือการโปรแกรมที่ไม่ดี ทำให้การทำความเข้าใจซอฟต์แวร์ และการแก้ไขซอฟต์แวร์ทำได้ยาก

ดังนั้นถ้าไม่มีวิธีการและเครื่องมือมาช่วยทำในขั้นตอนการตรวจจับร่องรอยที่ไม่ดีนี้ จะพบปัญหาที่สำคัญ [3] คือ

1. ทำให้เสียเวลา และกำลังคนในการทำขั้นตอนนี้มาก เพราะผู้พัฒนาขาดความรู้ความสามารถในการตรวจจับร่องรอยที่ไม่ดี
2. ในแต่ละโปรแกรมจะมีร่องรอยที่ไม่ดีที่แตกต่างกัน ดังนั้นจึงต้องทำการตรวจหาร่องรอยที่ไม่ดีทุกครั้ง สำหรับโปรแกรมใดๆ ที่ต้องการทำรีแฟคทอริง
3. ในระบบขนาดใหญ่ที่มีจำนวนความยาวของโปรแกรมหลายพันบรรทัด ถ้าไม่มีเครื่องมือมาช่วยการตรวจจับร่องรอยที่ไม่ดีนั้นต้องใช้เวลา และกำลังคนในการทำขั้นตอนนี้มาก ดังนั้นจึงมีความต้องการเครื่องมือมาช่วยในการตรวจจับร่องรอยที่ไม่ดีนั้นมากกว่า

ดังนั้นเพื่อช่วยให้ขั้นตอนการตรวจจับร่องรอยที่ไม่ดีนั้นสะดวกขึ้น ในงานวิจัยนี้จึงนำเสนอวิธีการตรวจจับร่องรอยที่ไม่ดี 6 ประเภท คือ Feature Envy, Large Class, Lazy Class, Long Method, Long Parameter List และ Switch Statement โดยทำการออกแบบมาตรวัดเชิงวัตถุสำหรับร่องรอยที่ไม่ดีแต่ละประเภท รวมทั้งพัฒนาเครื่องมือตรวจจับร่องรอยที่ไม่ดีสำหรับโปรแกรมภาษาจาวา (Java) ที่สนับสนุนกระบวนการการตรวจจับร่องรอยที่ไม่ดี โดยการคำนวณจากมาตรวัดที่ได้ออกแบบไว้

1.2 วัตถุประสงค์

1. เพื่อออกแบบมาตรวัดซอฟต์แวร์เชิงวัตถุ ที่สนับสนุนกระบวนการตรวจจับร่องรอยที่ไม่ดีสำหรับรีแฟคทอริง
2. เพื่อพัฒนาเครื่องมือสำหรับตรวจจับร่องรอยที่ไม่ดีสำหรับโปรแกรมภาษาจาวา

1.3 ขอบเขตงานวิจัย

1. เลือกวิธีการรีแฟคทอริงอย่างน้อย 2 วิธี ที่นำมาออกแบบมาตรวัดซอฟต์แวร์เชิงวัตถุ
2. โปรแกรมต้นฉบับที่จะถูกนำมาตรวจจับร่องรอยที่ไม่ดีนั้น ต้องเป็นโปรแกรมภาษาจาวา และผ่านการคอมไพล์แล้ว
3. จำนวนโปรแกรมที่ใช้ในการทดสอบกับเครื่องมือวัดอย่างน้อย 3 โปรแกรม
4. ขนาดของซอฟต์แวร์ที่ใช้ในการทดสอบต้องประกอบด้วยจำนวนคลาสอย่างน้อย 5 คลาส

5. พัฒนาเครื่องมือและใช้เครื่องมือบนระบบปฏิบัติการวินโดว์ (Windows) ตั้งแต่รุ่น 98 ขึ้นไป

1.4 ขั้นตอนและวิธีดำเนินงานวิจัย

1. ศึกษาวิธีการรีแฟคทอริง และร่องรอยที่ไม่ดีที่บ่งชี้ความเหมาะสมของการประยุกต์ใช้รีแฟคทอริงแต่ละวิธี
2. ศึกษามาตรวัดซอฟต์แวร์เชิงวัตถุที่มีอยู่ในปัจจุบัน
3. ศึกษาเครื่องมือที่ช่วยในการสร้างตัวแปลภาษา ที่มีการสร้างซอร์สโค้ด (Source Code) ของตัวแปลภาษาเป็นภาษาจาวา ได้แก่ โปรแกรมจาวาซีซี (JavaCC) ของบริษัท Sun Microsystems, Inc.
4. เลือกร่องรอยที่ไม่ดีที่จะใช้ในการหามาตรวัด
5. ออกแบบมาตรวัดซอฟต์แวร์เชิงวัตถุสำหรับร่องรอยที่ไม่ดีแต่ละวิธีที่เลือกใช้
6. ออกแบบและพัฒนาเครื่องมือที่ใช้ตรวจจับร่องรอยที่ไม่ดีสำหรับโปรแกรมภาษาจาวา
7. ทดสอบความน่าเชื่อถือของวิธีการรีแฟคทอริง ด้วยเครื่องมือที่พัฒนา
8. สรุปผลการวิจัย และข้อเสนอแนะ
9. จัดทำรายงานวิทยานิพนธ์

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. มาตรวัดซอฟต์แวร์เชิงวัตถุสำหรับร่องรอยที่ไม่ดี ซึ่งมาตรวัดนี้สามารถนำไปใช้ตรวจจับร่องรอยที่ไม่ดีแต่ละแบบได้
2. เครื่องมือที่จัดทำขึ้นเพื่อตรวจจับร่องรอยที่ไม่ดีสำหรับโปรแกรมภาษาจาวา
3. สามารถนำมาตรวัดที่ได้ไปสนับสนุนการทำรีแฟคทอริงแบบอัตโนมัติต่อไป

บทที่ 2

ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ทฤษฎีที่เกี่ยวข้อง

ทฤษฎีที่เกี่ยวข้องในงานวิจัยการตรวจจับร่องรอยที่ไม่ดีสำหรับรีแฟคทอริงโดยใช้มาตรวัดซอฟต์แวร์เชิงวัตถุ ได้แก่ การวัดซอฟต์แวร์เชิงวัตถุ (Object-oriented software measurement) และรีแฟคทอริง ซึ่งมีรายละเอียดดังต่อไปนี้

2.1.1 การวัดซอฟต์แวร์เชิงวัตถุ [4]

การวัดเป็นพื้นฐานสำคัญสำหรับทุกศาสตร์ทางวิศวกรรม ซึ่งเป็นกระบวนการกำหนดค่าตัวเลข หรือสัญลักษณ์ ให้กับคุณลักษณะ (Attributes) ของเอนทิตี (Entities) หรือสิ่งที่อยู่ในโลกของความเป็นจริงที่เราสนใจ เพื่อบรรยายเอนทิตีนั้น ให้อยู่ในรูปของกฎที่นิยามไว้ชัดเจน สำหรับการวัดในทางวิศวกรรมซอฟต์แวร์ (Software engineering) นั้น การวัดสามารถแบ่งได้เป็น 2 วิธี คือ

1. การวัดทางตรง (Direct measurement) เป็นการวัดเฉพาะคุณลักษณะภายใน (Internal attribute) ของสิ่งที่เราสนใจโดยไม่นำคุณลักษณะหรือเอนทิตีอื่นมาเกี่ยวข้อง ผลจากการวัดทางตรงทำให้ทราบลักษณะในด้านโครงสร้างของซอฟต์แวร์ ลักษณะของมาตรวัด (Metric) ซึ่งเป็นค่าพื้นฐาน เช่น การวัดความยาวของซอร์สโค้ดสามารถวัดได้จากการนับจำนวนบรรทัดทั้งหมดของโปรแกรม เป็นต้น
2. การวัดทางอ้อม (Indirect measurement) เป็นการวัดโดยการนำการวัดทางตรงมาประกอบการวัด เนื่องจากคุณลักษณะบางอย่าง ผลของการวัดทางอ้อมคือคุณภาพของซอฟต์แวร์ (Software quality) ในด้านต่างๆ เช่น การวัดความสามารถในการใช้งาน (Usability) ความสามารถในการทดสอบ (Testability) ความสามารถในการบำรุงรักษา (Maintainability) ความสามารถในการทำความเข้าใจระบบ (Understandability) เป็นต้น การที่จะวัดเพื่อทราบถึงคุณภาพของซอฟต์แวร์ในด้านใดด้านหนึ่งนั้น จะต้องกำหนดเกณฑ์ (Criteria) ในการวัด หรือปัจจัย (Factor) และแต่ละเกณฑ์ จะมีการกำหนดมาตรวัดที่เกี่ยวข้องที่วัดได้จากการวัดทางตรงหรือจากมาตรวัดทางอ้อม

การวัดทั้ง 2 วิธีจำเป็นต้องใช้มาตรวัด เพื่อวัดคุณลักษณะของสิ่งที่เราสนใจ ในปัจจุบันได้มีนักวิจัยหลายท่านได้นำเสนอมาตรวัดใหม่ๆ ที่นิยมใช้กันอย่างแพร่หลาย โดยมาตรวัดเหล่านี้สามารถแบ่งได้เป็น 2 ประเภท ได้แก่ มาตรวัดแบบดั้งเดิม (Traditional metrics) เช่น จำนวนบรรทัดของโปรแกรม (Lines Of Code – LOC) ค่าวัดไซโครเมติกคอมเพลกซิตีของแมคเคบ (Cyclomatric Complexity – CC) ใช้สำหรับวัดค่าความซับซ้อนของโปรแกรม และมาตรวัดเชิงวัตถุ ได้แก่ ระดับความลึกของการสืบทอดคุณสมบัติของคลาส (Depth of Inheritance Tree – DIT) จำนวนคลาสลูก (Number Of Children – NOC) และขาดความสัมพันธ์ระหว่างวัตถุ (Coupling Between Object – CBO) หลังจากหาค่ามาตรวัดต่างๆ ได้แล้ว จะต้องนำมาตรวัดเหล่านี้ มาทำการตรวจสอบ (Validation) ว่าค่ามาตรวัดที่ได้มีความน่าเชื่อถือ (Reliability) สามารถนำไปใช้วัดคุณภาพซอฟต์แวร์ต่อไปได้

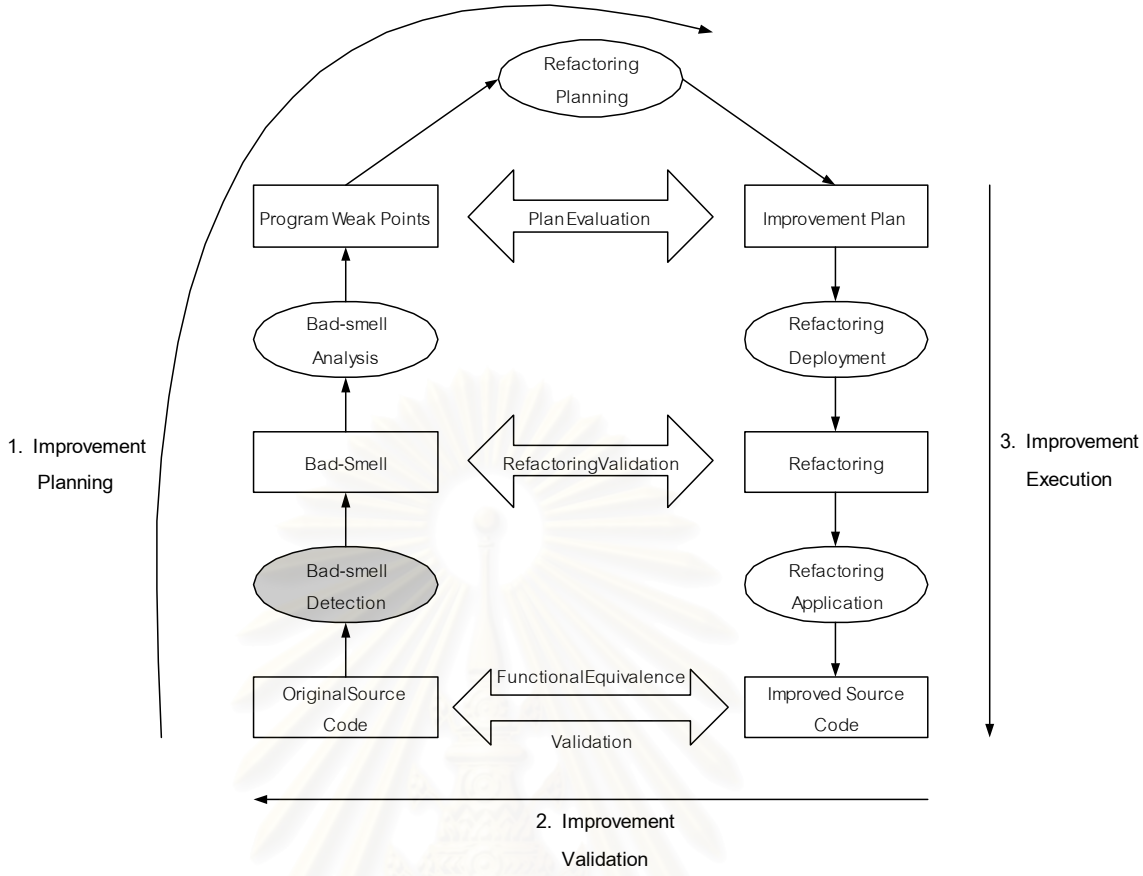
2.1.2 **รีแฟคทอริง** [5]

รีแฟคทอริง หมายถึง การเปลี่ยนแปลงโครงสร้างภายในของซอฟต์แวร์ โดยไม่ทำให้ฟังก์ชันการทำงานของซอฟต์แวร์นั้นเปลี่ยนแปลง เพื่อทำให้ซอฟต์แวร์นั้นง่ายต่อการทำความเข้าใจ เมื่อทำการแก้ไขซอฟต์แวร์ และทำให้เพิ่มความสามารถในการบำรุงรักษาซอฟต์แวร์ การรีแฟคทอริงอย่างเหมาะสม จะสามารถช่วยให้การออกแบบซอฟต์แวร์ ทำความเข้าใจได้ง่ายขึ้น ค้นพบข้อผิดพลาดที่ซ่อนอยู่ภายในโปรแกรม (Bugs) และช่วยให้โปรแกรมทำงานได้เร็วขึ้น วิธีการในการรีแฟคทอริงประกอบด้วย 72 วิธีการ แบ่งตามประเภท ซึ่งแสดงรายละเอียดในภาคผนวก ก

การทำรีแฟคทอริงมีกระบวนการทำงานตามลำดับ [6] ดังนี้

1. การวางแผนการปรับปรุงซอฟต์แวร์ (Improvement Planning) แบ่งเป็น 3 ขั้นตอน คือ การตรวจจับร่องรอยที่ไม่ดี (Bad-smell Detection) การวิเคราะห์ร่องรอยที่ไม่ดี (Bad-smell Analysis) และการวางแผนการทำรีแฟคทอริง (Refactoring Planning)
2. การตรวจสอบความเหมาะสมของการปรับปรุงซอฟต์แวร์ (Improvement Validation) แบ่งเป็น 3 ขั้นตอน คือ การประเมินแผนการทำรีแฟคทอริง (Plan Evaluation) การตรวจสอบความเหมาะสมของการทำรีแฟคทอริง (Refactoring Validation) และการตรวจสอบความเท่าเทียมกันของฟังก์ชัน (Functional Evaluation Validation)
3. การปรับปรุงซอฟต์แวร์ (Improvement Execution) แบ่งเป็น 2 ขั้นตอน คือ การเตรียมการรีแฟคทอริง (Refactoring Deployment) และการประยุกต์ใช้รีแฟคทอริง (Refactoring Application)

กระบวนการดังกล่าว สามารถแสดงรายละเอียดดังรูปที่ 2.1



รูปที่ 2.1 กระบวนการรีแฟคทอริง

การเลือกประเภทวิธีการ และระบุโครงสร้างภายในที่ควรทำรีแฟคทอริงสามารถทำได้จากการพิจารณาโดยใช้วิจารณญาณส่วนตัว (Subjective perception) และพิจารณาจากร่องรอยที่ไม่ดี ซึ่งร่องรอยที่ไม่ดี หมายถึง ลักษณะของการออกแบบซอฟต์แวร์ที่ไม่ดีในขั้นตอนการออกแบบหรือการเขียนโปรแกรมที่ไม่ดีในซอร์สโค้ด จึงทำให้ซอฟต์แวร์ทำงานได้ไม่มีประสิทธิภาพเป็นผลให้ต้องมีการรีแฟคทอริงเพื่อให้ซอฟต์แวร์ทำงานได้มีประสิทธิภาพดีขึ้น ซึ่งร่องรอยที่ไม่ดีมีทั้งหมด 22 ประเภท ดังนี้

1. Duplicate Code คือ โครงสร้างโค้ดที่เหมือนกันมากกว่า 1 ที่
2. Long Method คือ เมทธอดที่มีขนาดใหญ่
3. Large Class คือ คลาสที่มีฟังก์ชัน และจำนวนอินสแตนท์มาก
4. Long Parameter List คือ มีการส่งผ่านพารามิเตอร์มาก ทำให้การทำความเข้าใจยากขึ้น
5. Divergent Change คือ คลาสหนึ่งรองรับการเปลี่ยนแปลงหลายๆ อย่างที่เกิดขึ้น

6. Shotgun Surgery ตรงข้ามกับ Divergent Change คือ เมื่อมีการเพิ่มฟังก์ชันหนึ่ง ต้องทำการแก้ไขในหลายๆ คลาส ดังนั้นความสัมพันธ์ระหว่างคลาสและการเปลี่ยนแปลงใดๆ ควรเป็นหนึ่งต่อหนึ่ง
7. Feature Envy คือ เมื่อมีเมทธอดที่มีการเรียกใช้คุณสมบัติในคลาสอื่น มากกว่าคลาสเจ้าของเมทธอดเอง
8. Data Clumps พบว่ามีการใช้กลุ่มข้อมูลชุดเดียวกันในหลายๆ ที่ภายในโค้ด ซึ่งกลุ่มของข้อมูลประกอบด้วย คุณลักษณะภายในคลาส หรือพารามิเตอร์ในเมทธอด
9. Primitive Obsession คือ การใช้ข้อมูลชนิดตัวอักษร (String) และวันที่ (Date) ในภาษาจาวามีการสร้างเป็นคลาส เมื่อมีการใช้ชนิดข้อมูลเหล่านี้ จึงอาจทำให้คลาสมีขนาดใหญ่
10. Switch Statement คือ การปรากฏของสวิทซ์สเตทเมนท์ (Switch statement) เหมือนกันในโค้ดหลายๆ ที่ ทำให้เกิดโค้ดที่ซ้ำกันได้
11. Parallel Inheritance Hierarchy คือ เมื่อมีการสร้างคลาสลูกของคลาสหนึ่ง จะทำให้เกิดเป็นคลาสลูกของอีกคลาสหนึ่งด้วย
12. Lazy Class คือ คลาสที่ไม่ได้มีฟังก์ชันที่ทำงานหลักๆ มาก
13. Speculative Generality คือ มีการสร้างคลาสหรือเมทธอดทำหน้าที่เฉพาะกรณีพิเศษ ทำให้การทำเข้าความใจและการ บำรุงรักษาซอฟต์แวร์ยากขึ้น เช่น มีการสร้างคลาสหรือเมทธอดขึ้นมาเป็นกรณีทดสอบ
14. Temporary Field คือ การที่พบอ็อบเจกต์ ที่มีตัวแปรที่ใช้สำหรับเก็บค่าผลลัพธ์ชั่วคราวในโค้ด
15. Message Chains คือ การที่มีการเรียกใช้ข้อมูลของอ็อบเจกต์ หนึ่งจากอีกอ็อบเจกต์อื่น ซึ่งไม่ได้เรียกใช้จากอ็อบเจกต์ นั้นโดยตรง
16. Middle Man คือ คลาสที่ทำหน้าที่เป็นตัวกลางการติดต่อระหว่างออบเจ็ค
17. Inappropriate Intimacy คือ การที่พบว่ามีการพยายามเรียกใช้ส่วนไพรเวท (Private) ระหว่างคลาส
18. Alternative Classes with Difference Interfaces คือ การที่มีเมทธอดที่ทำหน้าที่เดียวกันแต่มีส่วนซิกเนเจอร์ต่างกัน (Signature)

19. Incomplete Library Class คือ การที่มีคลาสไลบรารี่ (Library Class) ที่ไม่สมบูรณ์ ทำให้การทำความเข้าใจคลาส เพื่อนำกลับมาใช้ใหม่ทำได้ยากขึ้น
20. Data Class คือ คลาสที่มีฟังก์ชันหลักเป็นการกำหนดค่าให้กับข้อมูล และภายในคลาสมีเพียงเอคเซสเซอร์เมทธอด (Accessor method) เท่านั้น
21. Refused Bequest คือ คุณสมบัติที่คลาสลูกไม่ต้องการใช้ แต่ได้รับจากการสืบทอดคุณสมบัติจากคลาสแม่มาทั้งหมด
22. Comment การเขียนคำอธิบายนั้น ไม่ได้เป็นร่องรอยที่ไม่ดี แต่เมื่อมีการแก้ไขร่องรอยที่ไม่ดีข้างต้นด้วยวิธีการ รีแฟคทอริงแล้ว ควรเขียนคำอธิบายการทำงานในแต่ละส่วน เพื่อทำให้การทำความเข้าใจง่ายขึ้น

ตัวอย่างของร่องรอยที่ไม่ดี และวิธีรีแฟคทอริง เช่น Feature Envy คือ การที่มีเมทธอดหรือคุณลักษณะไปเรียกใช้หรือถูกใช้โดยคลาสอื่นมากกว่าคลาสที่เป็นเจ้าของเอง ซึ่งสามารถแก้ไขได้โดยวิธี Move Method โดยย้ายเมทธอดนั้นไปยังคลาสที่มีจำนวนการเรียกใช้คุณสมบัติ (Feature) มากกว่า แล้วลบเมทธอดในคลาสเดิมทิ้ง หรือให้เมทธอดในคลาสเดิมทำหน้าที่ง่ายๆ อย่างอื่น เช่น ไปเรียกใช้เมทธอดที่สร้างขึ้นใหม่ แต่ถ้าเมทธอดนั้นเรียกใช้คุณสมบัติจากคลาสอื่นมากกว่า 1 คลาส ควรปรับปรุงด้วยวิธี Extract Method โดยแยกส่วนของเมทธอดนั้น ที่ถูกเรียกใช้โดยคลาสอื่นมาสร้างเป็นเมทธอดใหม่ และย้ายเมทธอดเหล่านั้นไปยังคลาสที่มีการเรียกใช้ต่างกัน หรือวิธี Move Attribute โดยย้ายคุณลักษณะนั้นไปยังคลาสที่มีจำนวนการเรียกใช้คุณลักษณะมากที่สุด

2.2 งานวิจัยที่เกี่ยวข้อง

2.2.1 งานวิจัย “Detecting Design Flaws via Metrics in Object-Oriented System” [7]

งานวิจัยนี้นำเสนอมาตรวัดสำหรับตรวจจับร่องรอยที่ไม่ดีในระบบเชิงวัตถุ และนำเสนอขั้นตอนการออกแบบวิธีการสำหรับการตรวจจับร่องรอยไม่ดีบนซอร์สโค้ด 5 ขั้นตอน ซึ่งแบ่งเป็น 3 ขั้นตอนแรก คือ กำหนดแรงจูงใจ (Motivation) กำหนดวิธีการวัด (Strategy) การออกแบบมาตรวัด (Metrics) จะอธิบายถึงนิยามของเทคนิคการตรวจจับนี้ ส่วน 2 ขั้นตอนสุดท้าย คือ การคำนวณค่ามาตรวัด (Measurement) และการตรวจสอบผลที่พบ (Finding) จะอธิบายถึงการนำเทคนิคนี้ไปประยุกต์ใช้กับกรณีศึกษา โดยผู้วิจัยได้นำวิธีการนี้มาประยุกต์ใช้กับร่องรอยที่ไม่ดี 2 วิธี คือ Data Class และ God Class ซึ่งมีรายละเอียดดังนี้

Data Class [5]

คือ คลาสที่มีทำหน้าที่หลักคือ การกำหนดค่าให้กับข้อมูล และภายในคลาสมีเพียงเอค-เซสเซอร์เมทธอด (Accessors method) คือเมทธอดที่ใช้สำหรับอ่านค่าสมาชิกข้อมูลในคลาสเท่านั้น

แรงจูงใจ : เนื่องจาก Data Class นั้น ขาดความมีหน้าที่ของเมทธอด (Lack of functional method) นั่นคือความสัมพันธ์ระหว่างข้อมูลและพฤติกรรมนั้นไม่ได้อยู่ร่วมกัน ซึ่งไม่ใช่ลักษณะสำคัญของการออกแบบเชิงวัตถุ ทำให้มีผลกระทบต่อคุณลักษณะทางคุณภาพ ดังนี้

1. ในการบำรุงรักษาซอฟต์แวร์ ถ้ามีการเปลี่ยนแปลงใน Data Class จะทำให้กระทบกับคลาสอื่นที่เรียกใช้ข้อมูลใน Data Class นั้น
2. ในการทดสอบซอฟต์แวร์ ถ้ามีคลาสอื่นๆ มาเรียกใช้ข้อมูลใน Data Class เหมือนกัน จะเกิดโค้ดที่ซ้ำซ้อนขึ้น ทำให้เพิ่มภาระในการทดสอบระบบมากขึ้น
3. ในการทำความเข้าใจคลาสที่เรียกใช้ Data Class เพราะฟังก์ชันการทำงานไม่ได้อยู่ในคลาสนั้น แต่มาอยู่ใน Data Class แทน

วิธีการวัด : สามารถตรวจจับ Data Class ได้โดยการหาคลาสที่เรียกว่า ไลท์เวทคลาส (Lightweight classes) เช่น คลาสที่ในอินเตอร์เฟส (Interface) ส่วนมากไม่มีฟังก์ชันอยู่ จากนั้นหาคลาสที่มีการกำหนดเอคเซสเซอร์เมทธอดมาก และมีการประกาศตัวแปรไว้ในอินเตอร์เฟสนั้น

มาตรวัด :

1. มาตรวัด Weight of a class (*WOC*)

   นิยาม: จำนวนของเมทธอดที่ไม่ใช่เอคเซสเซอร์เมทธอดในคลาสหารด้วยจำนวนสมาชิกของอินเตอร์เฟสทั้งหมด โดยไม่รวมสมาชิกที่สืบทอดมา

   คำอธิบาย: ถ้า *WOC* มีค่าน้อยๆ หมายความว่าอาจเป็น Data Class ได้ เพราะคลาสที่มีการออกแบบ ที่ดีจะมี *WOC* = 1 หรือใกล้เคียง แสดงว่าเมทธอดมีฟังก์ชันทำหน้าที่อื่นภายในคลาส

   ข้อกำหนดของค่าที่พิจารณา: คลาสที่มี *WOC* อยู่ระหว่าง 0 – 0.33

2. มาตรวัด Number of public attribute (*NOPA*)

   นิยาม: จำนวนคุณลักษณะที่ไม่ได้ถูกสืบทอดมา แต่มีการประกาศในอินเตอร์เฟสของคลาส

คำอธิบาย: คลาสที่มีสมาชิกของข้อมูลเป็นแบบพับบลิค (Public) ที่ละเมิดกฎการห่อหุ้ม (Encapsulation) ของการออกแบบเชิงวัตถุ

ข้อกำหนดของค่าที่พิจารณา: คลาสที่อยู่ใน 10 ลำดับแรก ที่ $NOPA \geq 5$

3. มาตรวัด Number of accessor method (*NOAM*)

นิยาม: จำนวนเอคเซสเซอร์เมทธอดที่ไม่ได้ถูกสืบทอดมา แต่มีการประกาศอยู่ในอินเตอร์เฟสของคลาส

คำอธิบาย: ถ้าคลาสมี *NOAM* สูง หมายความว่า การทำงานบางส่วนของคลาสนั้นเป็นไปได้มากที่จะไปปรากฏอยู่ในคลาสอื่นๆ ได้

ข้อกำหนดของค่าที่พิจารณา: คลาสที่อยู่ใน 10 ลำดับแรก ที่ $NOAM \geq 3$

God Class [8]

การออกแบบเชิงวัตถุที่ดี จะต้องมีการกระจายการทำงานระหว่างคลาสให้ชัดเจน ในระบบอัจฉริยะ (Intelligence system) เรียก คลาสที่ทำหน้าที่ควบคุมการทำงานหลักทั้งหมดในระบบอัจฉริยะว่า God Class

แรงจูงใจ : เนื่องจาก God Class มีการทำงานหลักหลายหน้าที่ มอบหมายหน้าที่รองให้กับคลาสอื่น และเรียกใช้ข้อมูลจากคลาสอื่นนั้น ทำให้มีผลกระทบต่อคุณลักษณะทางคุณภาพดังนี้

1. การนำกลับมาใช้ใหม่ เพราะ God Class นั้น มีหลายฟังก์ชัน ทำให้ความเป็นไปได้ที่จะนำคลาสนั้นกลับมาใช้ใหม่ได้อีกลดลง
2. การทำความเข้าใจคลาส เพราะความรับผิดชอบของ God Class มีหลากหลาย มีผลให้การทำความเข้าใจคลาสนั้น เข้าใจได้ยากขึ้น

วิธีการวัด : God Class พิจารณาได้จากการเข้าถึงข้อมูลมาจากไลท์เวทคลาสมีมาก ทั้งเข้าถึงโดยตรงและผ่านเอคเซสเซอร์เมทธอด และมีพฤติกรรมที่ไม่มีการสื่อสารมาก แสดงว่าไม่มีการกระจายการทำงานระหว่างคลาส สามารถตรวจจับ God Class โดยเริ่มจากการหาคลาสที่ขึ้นกับข้อมูลในไลท์เวทคลาสมาก จากนั้นกำจัดคลาสที่มีขนาดเล็ก และคลาสที่มีการเกาะกันเป็นก้อนน้อยออก

มาตรวัด :

1. มาตรวัด Access of foreign data (*AOFD*)

นิยาม: จำนวนคลาสภายนอกจากที่ซึ่งยอมให้คลาสนั้นเข้าถึงข้อมูล ทั้งเข้าถึงโดยตรงและผ่านเอคเซสเซอร์เมทธอด โดยไม่รวมถึงอินเนอร์คลาส (Inner class) และคลาสที่สืบทอดคุณสมบัติ (Superclass)

คำอธิบาย: ถ้าคลาสมีค่า *AOFD* สูง หมายความว่า คลาสนี้มีความน่าจะเป็นที่เป็น God Class สูง

ข้อกำหนดของค่าที่พิจารณา: คลาสที่อยู่ใน 10 ลำดับแรก ที่ $AOFD \geq 3$

2. มาตรวัด Weighted method count (*WMC*) [9]

นิยาม: คำนวณได้จากผลรวมของค่าความซับซ้อนของมาตรวัดไซโคลเมติกของแมค-เคบทุกเมทธอดภายในแต่ละคลาส ถ้าไม่คำนึงถึงค่าความซับซ้อนแล้ว *WMC* จะคำนวณได้จากค่าจำนวนของเมทธอดในคลาส

คำอธิบาย: ถ้าคลาสมีค่า *WMC* สูง หมายความว่าคลาสมีขนาดใหญ่และมีความซับซ้อนมาก เพราะฉะนั้นอาจเป็น God Class ได้

ข้อกำหนดของค่าที่พิจารณา: คลาสที่อยู่ใน 10 ลำดับแรกของระบบ

3. มาตรวัด Tight class cohesion (*TCC*) [10]

นิยาม: จำนวนความสัมพันธ์ของเมทธอดที่สื่อสารกันโดยตรง

คำอธิบาย: ถ้าคลาสมีค่า *TCC* ต่ำ คือ มีพฤติกรรมที่ไม่มีการสื่อสารมากในคลาส

ข้อกำหนดของค่าที่พิจารณา: คลาสที่มีค่า *TCC* อยู่ระหว่าง 0 – 0.33

จากนั้นได้สร้างเครื่องมือสำหรับตรวจจับร่องรอยที่ไม่ดีขึ้น ประกอบด้วย 2 ส่วนคือ TableGen สำหรับเก็บข้อมูลการออกแบบที่ต้องการจากโปรแกรมภาษาซีพลัสพลัส (C++) เช่น คลาส เมทธอด ความสัมพันธ์ระหว่างคลาส การเรียกใช้เมทธอด เป็นต้น นำข้อมูลเหล่านั้นมาเก็บในรูปแบบของตาราง และส่วนการนำ มาตรวัดมาคำนวณโดยใช้เครื่องมือฐานข้อมูลเชิงสัมพันธ์ (Relational database engine) ซึ่งในงานวิจัยนี้เลือกใช้ ฐานข้อมูลออราเคิล (Oracle) เพื่อสร้างชุดคำสั่งเอสคิวแอล (SQL) สำหรับคำนวณมาตรวัดที่เลือกใช้ โดยดึงข้อมูลจากตารางเหล่านั้น

ผลลัพธ์ที่ได้จากงานวิจัยนี้ คือ มาตรวัดสำหรับการตรวจจับร่องรอยที่ไม่ดี 2 วิธี คือ Data Class และGod Class ดังรายละเอียดข้างต้น ซึ่งสามารถนำมาตรวัดนี้ไปใช้กับกรณีศึกษาอื่นๆ เพื่อตรวจสอบความน่าเชื่อถือของมาตรวัดนี้ และสามารถนำขั้นตอนการออกแบบวิธีการสำหรับการตรวจจับร่องรอยที่ไม่ดีนี้ไปประยุกต์ใช้ เพื่อออกแบบวิธีการสำหรับการตรวจจับร่องรอยที่ไม่ดีอื่นๆ แต่ขั้นตอนการออกแบบนี้ไม่สามารถประยุกต์ใช้กับร่องรอยที่ไม่ดี 2 ประเภท คือ ร่องรอยที่ไม่

ดีที่ไม่สามารถวัดได้ เช่น Duplicate Code และร่องรอยที่ไม่ดีที่ไม่สามารถพบได้ที่ซอร์สโค้ด เช่น จำนวนการตรวจพบข้อผิดพลาดต่อคลาส ซึ่งสามารถหาได้จากรายงานข้อผิดพลาดที่พบ

2.2.2 งานวิจัย "A Quantitative Evaluation of Maintainability Enhancement by Refactoring" [6]

งานวิจัยนี้นำเสนอมาตรวัดที่สนับสนุนกระบวนการการตรวจสอบความเหมาะสมของการทำรีแฟคทอริง โดยประเมินจากความสามารถในการบำรุงรักษาของเมทธอด ผู้วิจัยนี้ทำการวัดความสามารถในการบำรุงรักษาจากการขึ้นต่อกัน (Coupling) โดยนิยามมาตรวัดเพื่อวัดการขึ้นต่อกันของเมทธอดเป็น 3 ประเภทคือ การขึ้นต่อกันแบบส่งค่าคืน (Return value coupling) การขึ้นต่อกันแบบพารามิเตอร์ (Parameter coupling) และการขึ้นต่อกันแบบใช้ตัวแปรร่วมกัน (Share variable coupling) ซึ่งมาตรวัดทั้งสามมาตรวัดจะถูกรวมเข้าด้วยกันเป็นมาตรวัดเดียว โดยให้น้ำหนัก (Weight) ตามความเหมาะสม เพื่อใช้ในการวัดการเปลี่ยนแปลงของความสามารถในการบำรุงรักษา ในงานวิจัยนี้ได้พิจารณาเฉพาะวิธีการรีแฟคทอริงที่ช่วยเพิ่มความสามารถในการบำรุงรักษา และช่วยลดการขึ้นต่อกันระหว่างเมทธอด เช่น วิธี Extract Method วิธี Extract Class และวิธี Move Method ผู้วิจัยจึงได้ทำการทดลองกับโปรเจคที่พัฒนาโดยภาษาซีพลัสพลัส โดยวัดค่ามาตรวัดการขึ้นต่อกัน เพื่อเปรียบเทียบค่ามาตรวัดระหว่างก่อน และหลังการรีแฟคทอริง โดยใช้เครื่องมือ Refactoring Assistant สำหรับคำนวณค่ามาตรวัดการขึ้นต่อกันในโปรเจคที่เลือกมาใช้ทดลอง และเครื่องมือนี้สามารถทำรีแฟคทอริงอัตโนมัติสำหรับวิธีรีแฟคทอริงพื้นฐาน ดังเช่นวิธี Extract Method วิธี Move Method และ วิธี Inline Method ได้

ผลลัพธ์ที่ได้จากงานวิจัยนี้ แสดงให้เห็นว่าค่ามาตรวัดการขึ้นต่อกันทั้ง 3 มาตรวัด หลังจากถูกรวมเป็นมาตรวัดเดียวแล้ว จะลดลงหลังจากทำวิธีรีแฟคทอริงแล้ว หมายความว่าสามารถเพิ่มความสามารถในการบำรุงรักษาของเมทธอด จากการทำรีแฟคทอริงได้ แต่งานวิจัยนี้พิจารณาเพียงการขึ้นต่อกันของเมทธอดเท่านั้น ดังนั้นในวิทยานิพนธ์ฉบับนี้ จึงนำวิธีการตรวจสอบความเหมาะสมของการทำรีแฟคทอริง ซึ่งประเมินจากความสามารถในการบำรุงรักษาซอฟต์แวร์มาใช้ โดยเปรียบเทียบมาตรวัดทางด้านการเกาะกันเป็นก้อน (Cohesion) และมาตรวัดทางด้านความซับซ้อน (Complexity) ก่อนและหลังการประยุกต์ใช้รีแฟคทอริง

2.2.3 งานวิจัย "A Systematic Class Refactoring Approach based on Metrics" [11]

งานวิจัยนี้ นำเสนอวิธีการประยุกต์ใช้รีแฟคทอริง โดยการใช้มาตรวัดเพื่อที่จะช่วยเหลือในการตัดสินใจว่า ควรจะทำรีแฟคทอริงที่ใด และเลือกวิธีรีแฟคทอริงไหนมาประยุกต์ โดยใช้มาตรวัดการเกาะกันเป็นก้อน คือ มาตรวัด *CBMC* และมาตรวัดการขึ้นต่อกัน คือ มาตรวัด *CBO* สำหรับ

เลือกว่าคลาสใดควรจะทำการรีแฟคทอริง โดยเลือกคลาสที่มีค่ามาตรวัด *CBMC* ต่ำ และมาตรวัด *CBO* สูง และกำหนดมาตรวัดอื่นๆ เพื่อเลือกวิธีรีแฟคทอริงที่เหมาะสมมาประยุกต์ใช้ แบ่งมาตรวัดเป็น 3 กลุ่ม คือ มาตรวัดสำหรับรีแฟคทอริงคลาส เมทธอด และคุณลักษณะ ซึ่งมีรายละเอียดดังนี้

1. มาตรวัดสำหรับรีแฟคทอริงคลาส มี 3 มาตรวัด คือ

   1) มาตรวัด *NMC* คือ จำนวนเมทธอดปกติ (Normal methods) และจำนวนเมทธอดที่สืบทอดคุณสมบัติมาจากคลาสแม่

   โดยที่ เมทธอดพิเศษ (Special methods) คือเมทธอดประเภทเอคเซสเซอร์เมทธอด ดีลีเกชันเมทธอด (Delegation method) คอนสตรัคเตอร์ (Constructor) และดีสตรัคเตอร์ (Destructor) และเมทธอดปกติ คือเมทธอดที่ไม่อยู่ในกลุ่มเมทธอดพิเศษ

   2) มาตรวัด *NAC* คือ จำนวนคุณลักษณะภายในคลาส และคุณลักษณะที่สืบทอดคุณสมบัติจากคลาสแม่

   3) มาตรวัด *NGC* คือ จำนวนกลูเมทธอด (Glue methods)

   โดยที่ กลูเมทธอด ($M_g(Gr)$) คือ เซตของเมทธอดที่ทำให้กราฟที่อ้างอิง (Reference graph - Gr) สามารถแบ่งเป็นกราฟย่อยได้ และกราฟที่อ้างอิงคือกราฟแสดงการติดต่อกันระหว่างสมาชิกของคลาส

2. มาตรวัดสำหรับรีแฟคทอริงเมทธอด มี 3 มาตรวัด คือ

   1) มาตรวัด *NUM* คือ จำนวนเมทธอดปกติที่เรียกใช้เมทธอด *m* ภายในคลาสที่กำหนดเมทธอด *m*

   2) มาตรวัด *NIM* คือ จำนวนเมทธอดปกติที่เรียกใช้เมทธอด *m* ภายนอกคลาสที่กำหนดเมทธอด *m*

   3) มาตรวัด *RNM* คือ อัตราส่วนระหว่างสมาชิกที่ถูกใช้ในเมทธอด *m* และไม่ได้อยู่ในคลาสที่กำหนดเมทธอด *m* กับสมาชิกทั้งหมดที่ถูกใช้ในเมทธอด *m*

3. มาตรวัดสำหรับรีแฟคทอริงคุณลักษณะ มี 2 มาตรวัด คือ

   1) มาตรวัด *NUA* คือ จำนวนเมทธอดปกติที่เรียกใช้คุณลักษณะ *ma* และอยู่ในคลาสที่กำหนดคุณลักษณะ *ma*

2) มาตรวัด *RNA* คือ อัตราส่วนระหว่างจำนวนเมทธอดปกติที่เรียกใช้คุณลักษณะ *ma* และไม่ได้อยู่ในคลาสที่กำหนดคุณลักษณะ *ma* กับจำนวนเมทธอดปกติทั้งหมดที่เรียกใช้คุณลักษณะ *ma*

ตารางที่ 2.1 ความสัมพันธ์ระหว่างวิธีรีแฟคทอริงและค่ามาตรวัด

	มาตรวัด			วิธีรีแฟคทอริง
คลาส	*NMC*	*NAC*	*NGC*	
	↑	↑	↓	Extract Class
	↓	↓		Inline Class
เมทธอด	*NIM*	*NUM*	*RNM*	
	↑	↓		Move Method
		↓	↑	
	↓	↓	↓	Push Down Method
คุณลักษณะ	*NUA*	*RNA*		
	↓	↑		Move Attribute
	↓	↓		Push Down Attribute

(สัญลักษณ์ ↑ หมายถึงสูง และ ↓ หมายถึงต่ำ)

วิธีการเลือกวิธีรีแฟคทอริงเพื่อประยุกใช้ แสดงรายละเอียดดังตารางที่ 2.1 ยกตัวอย่างเช่น ถ้าคลาสใดมีค่ามาตรวัด *NMC* และ *NAC* สูง ส่วนค่ามาตรวัด *NGC* ต่ำ ควรเลือกใช้วิธี Extract Class แต่ถ้าค่ามาตรวัด *NMC* และ *NAC* ต่ำ ควรเลือกใช้วิธี Inline Class เป็นต้น

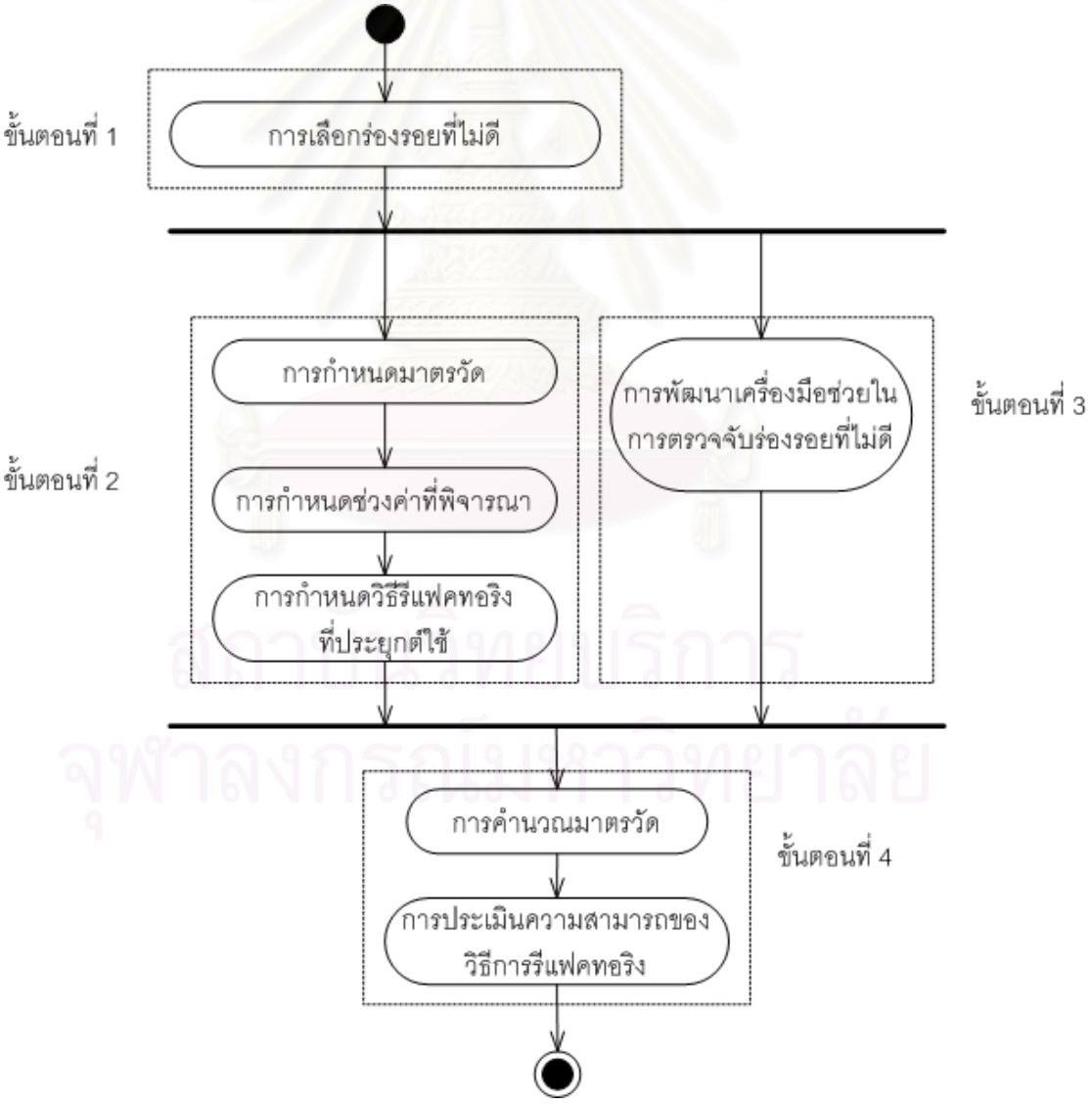
ผลลัพธ์ที่ได้จากงานวิจัยนี้ คือ วิธีการช่วยในการตัดสินใจว่าควรจะรีแฟคทอริงที่ใด และควรใช้วิธีรีแฟคทอริงไหน ซึ่งรายละเอียดแสดงดังตารางความสัมพันธ์ระหว่างวิธีรีแฟคทอริงและค่ามาตรวัด แต่วิธีการที่ใช้ในงานวิจัยนี้ เป็นการกำหนดเพียงค่ามาตรวัดสูง หรือต่ำ เพื่อช่วยในการตัดสินใจ ซึ่งไม่ได้กำหนดว่าค่ามาตรวัดสูง หรือต่ำมีค่าเท่าใด อาจจะทำให้เกิดความไม่แน่นอนในการเลือกใช้วิธีรีแฟคทอริงต่างๆ ดังนั้นควรจะกำหนดช่วงของค่ามาตรวัดที่แน่นอนว่ามีค่าเท่าใด

หมายเหตุ : งานวิจัยนี้ทำที่ประเทศเกาหลี ในช่วงเวลาเดียวกับการทำวิทยานิพนธ์นี้ จึงมีความคล้ายกัน

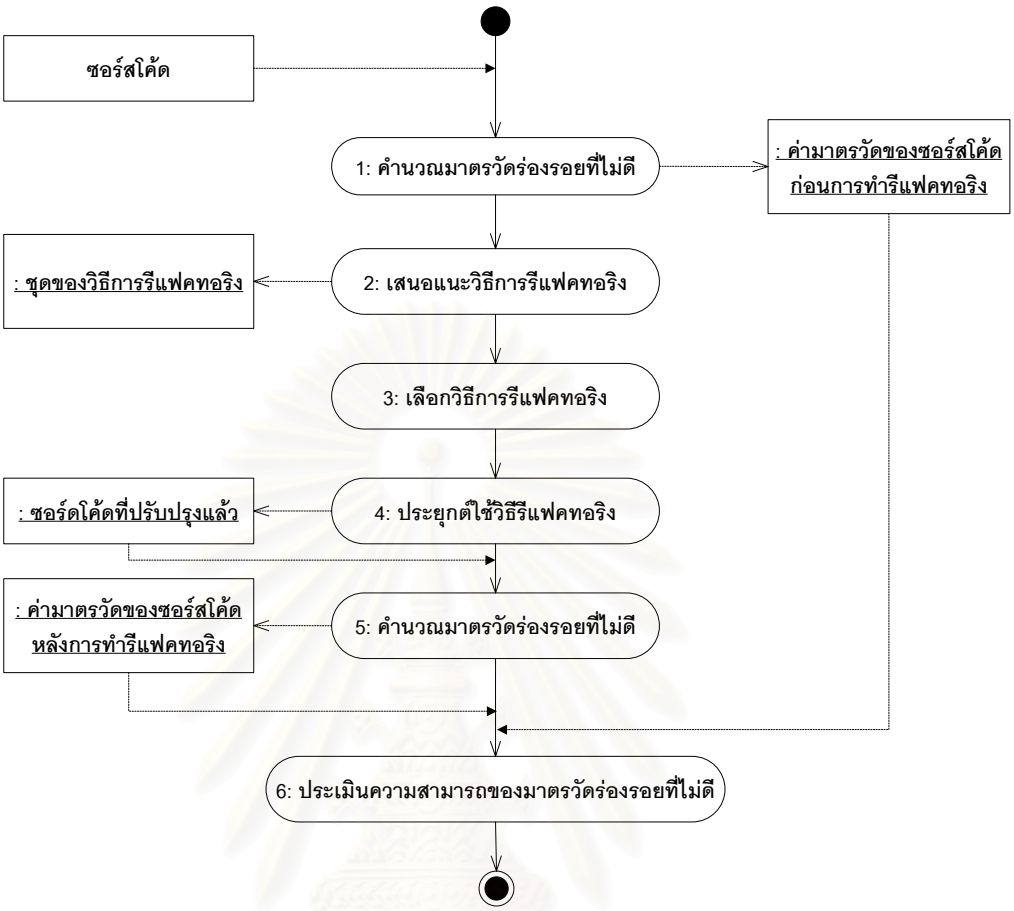
บทที่ 3

วิธีการตรวจจับร่องรอยที่ไม่ดี

งานวิจัยนี้สามารถแบ่งออกเป็น 4 ขั้นตอน ดังแสดงด้วยแผนภาพแอคทิวิตี (Activity diagram) ในรูปที่ 3.1 ได้แก่ การเลือกร่องรอยที่ไม่ดีสำหรับซอฟต์แวร์เชิงวัตถุ การออกแบบมาตรวัดสำหรับการตรวจจับร่องรอยที่ไม่ดี การออกแบบและพัฒนาเครื่องมือช่วยในการตรวจจับร่องรอยที่ไม่ดี และการทดสอบความน่าเชื่อถือของมาตรวัดร่องรอยที่ไม่ดี ในบทนี้จะอธิบายถึงขั้นตอนที่ 1 และ 2 เป็นรายละเอียดเกี่ยวกับวิธีการตรวจจับร่องรอยที่ไม่ดี ส่วนขั้นตอนที่ 3 และ 4 รายละเอียดอยู่ในบทที่ 4 และ 5 ตามลำดับ



รูปที่ 3.1 แผนภาพขั้นตอนการวิจัย



รูปที่ 3.2 แผนภาพขั้นตอนตรวจจับร่องรอยที่ไม่ดี

สำหรับวิธีการตรวจจับร่องรอยที่ไม่ดีนั้น แบ่งเป็น 2 ส่วนคือ การออกแบบวิธีการตรวจจับร่องรอยที่ไม่ดี และการออกแบบมาตรวัดร่องรอยที่ไม่ดี (Bad-smell metrics) ซึ่งมีรายละเอียดดังนี้

3.1 การออกแบบวิธีการตรวจจับร่องรอยที่ไม่ดี

งานวิจัยนี้ ได้นำเสนอวิธีการตรวจจับร่องรอยที่ไม่ดีสำหรับซอร์สโค้ด แสดงโดยแผนภาพแอคทิวิตีในรูปที่ 3.2 รายละเอียดของแต่ละขั้นตอน มีดังนี้

1. นำซอร์สโค้ดที่ต้องการตรวจจับร่องรอยที่ไม่ดี มาคำนวณค่ามาตรวัดร่องรอยที่ไม่ดี ผลที่ได้คือ ค่าของมาตรวัดในซอฟต์แวร์ที่พบว่าอยู่ในข้อกำหนดของค่าที่พิจารณา ซึ่งแสดงว่าอาจจะเป็นร่องรอยที่ไม่ดี

2. เสนอแนะวิธีการรีแฟคทอริง ซึ่งวิธีการรีแฟคทอริงนี้ อาจจะมีได้มากกว่าหนึ่งวิธี เพื่อแก้ไขร่องรอยที่ไม่ดีแต่ละประเภท
3. จากนั้นเลือกวิธีการรีแฟคทอริงเพื่อนำไปแก้ไขร่องรอยที่ไม่ดีที่พบ
4. นำวิธีการรีแฟคทอริงที่เลือกไปประยุกต์ใช้กับซอร์สโค้ดที่นำมาตรวจจับร่องรอยที่ไม่ดี หลังจากนั้นจะได้ซอร์สโค้ดที่ปรับปรุงแล้ว
5. นำซอร์สโค้ดที่ปรับปรุงแล้วมาคำนวณค่ามาตรวัดร่องรอยที่ไม่ดีอีกครั้ง เพื่อหาค่ามาตรวัดในซอร์สโค้ดหลังการทำรีแฟคทอริง
6. ในขั้นตอนสุดท้าย นำค่ามาตรวัดทางร่องรอยที่ไม่ดีก่อนและหลังการทำรีแฟคทอริงมาเปรียบเทียบ เพื่อประเมินความสามารถของมาตรวัดร่องรอยที่ไม่ดี

3.2 การออกแบบมาตรวัดร่องรอยที่ไม่ดี

การอธิบายและกำหนดนิยามของมาตรวัดสำหรับตรวจจับและหาตำแหน่งของร่องรอยที่ไม่ดี จำแนกเป็น 6 ส่วนคือ นิยาม แรงจูงใจ วิธีการวัด มาตรวัดร่องรอยที่ไม่ดี ข้อกำหนดของค่าที่พิจารณา (Specification of outliers) และการประยุกต์ใช้รีแฟคทอริง (Application of refactoring) มีรายละเอียดดังนี้

1. นิยาม คือ การอธิบายลักษณะของร่องรอยที่ไม่ดีแต่ละประเภท
2. แรงจูงใจ คือ รายละเอียดของผลกระทบของร่องรอยที่ไม่ดีต่อซอฟต์แวร์ และอธิบายถึงการพบร่องรอยที่ไม่ดีนั้นได้อย่างไร
3. วิธีการวัด คือ วิธีการที่ใช้เพื่อตรวจหาร่องรอยที่ไม่ดี
4. มาตรวัดร่องรอยที่ไม่ดี คือ การกำหนดนิยามของมาตรวัดสำหรับตรวจหาร่องรอยที่ไม่ดี ซึ่งมาตรวัดร่องรอยที่ไม่ดีที่นำมาใช้ในงานวิจัยนี้ มาจากการออกแบบมาตรวัดใหม่ 7 มาตรวัด คือ มาตรวัด *NCDM*, *NCDA*, *NCRA*, *SMIV*, *SFIV*, *NOT* และ *NOSS* และการใช้มาตรวัดเดิมที่มีอยู่แล้ว เช่น มาตรวัด *NIM*, *NIV* เป็นต้น ให้ตรงตามลักษณะของร่องรอยที่ไม่ดีแต่ละวิธีที่เลือกใช้
5. ข้อกำหนดของค่าที่พิจารณา คือ ช่วงของค่ามาตรวัดที่เป็นร่องรอยที่ไม่ดี ถ้าค่าของมาตรวัดในซอฟต์แวร์อยู่ในช่วงที่กำหนดนี้ แสดงว่าอาจจะเป็นร่องรอยที่ไม่ดีได้ ซึ่งนำช่วงของค่ามาตรวัดมาจากการออกแบบใหม่สำหรับมาตรวัดใหม่ และมาจากหนังสือ Object-Oriented Software Metrics [12] สำหรับมาตรวัดเดิมที่มีอยู่แล้ว

6. การประยุกต์ใช้รีแฟคทอริง คือ ข้อเสนอแนะวิธีการรีแฟคทอริงสำหรับการปรับปรุงซอฟต์แวร์ เพื่อกำจัดร่องรอยที่ไม่ดี ซึ่งวิธีการรีแฟคทอริงที่เลือกใช้ในงานวิจัยนี้ นำมาจากหนังสือ Refactoring: Improving the design of existing code [5]

จากขั้นตอนการออกแบบดังกล่าว ได้นำมาออกแบบมาตรวัดร่องรอยที่ไม่ดี 6 ประเภท คือ Feature Envy, Large Class, Lazy Class, Long Method, Long Parameter Lists และ Switch Statement ซึ่งมีรายละเอียดดังนี้

Feature Envy

1. นิยาม

หมายถึง การที่มีเมทธอด หรือคุณลักษณะไปเรียกใช้หรือถูกใช้โดยคลาสอื่นมากกว่าคลาสที่เป็นเจ้าของเมทธอด หรือคุณลักษณะนั้น

2. แรงจูงใจ

จุดประสงค์หลักของเทคโนโลยีเชิงวัตถุ คือ การห่อหุ้มข้อมูลกับฟังก์ชันที่กระทำกับข้อมูลนั้นเข้าด้วยกัน แต่ลักษณะของ Feature Envy ขาดคุณสมบัติของการห่อหุ้ม เกิดขึ้นเมื่อมีเมทธอดไปเรียกใช้หรือถูกใช้โดยคลาสอื่น มากกว่าคลาสที่กำหนดเมทธอดนั้น หรือถ้าคลาสอื่น มาเรียกใช้คุณลักษณะมากกว่าคลาสที่กำหนดคุณลักษณะนั้นเอง

3. วิธีการวัด

คุณสมบัติ หมายถึง คุณลักษณะ และเมทธอดที่กำหนดภายในคลาส รวมถึงคุณสมบัติของคลาสที่สืบทอดคุณสมบัติมาด้วย ดังนั้นจึงแบ่งการตรวจหา Feature Envy ซึ่งพิจารณาตามการเรียกใช้คุณลักษณะ และเมทธอดเป็น 2 วิธี ดังนี้

1) การตรวจหา Feature Envy สำหรับเมทธอด

พิจารณาภายในเมทธอด โดยการนับจำนวนการเรียกใช้คุณสมบัติของคลาสที่กำหนดเมทธอดนั้น เปรียบเทียบกับจำนวนการเรียกใช้คุณสมบัติของคลาสอื่น ถ้าพบว่าเมทธอดใด มีจำนวนการเรียกใช้คุณสมบัติของคลาสที่กำหนดเมทธอดนั้นน้อยกว่าคลาสอื่น ให้สงสัยว่าเมทธอดนั้น เป็นร่องรอยที่ไม่ดี ซึ่งสามารถปรับปรุงด้วยวิธี Move Method โดยย้ายเมทธอดนั้นไปยังคลาสที่มีจำนวนการเรียกใช้คุณสมบัติมากกว่า แต่ถ้าเมทธอดนั้นเรียกใช้คุณสมบัติจากคลาสอื่นมากกว่า 1 คลาส ควรปรับปรุงด้วยวิธี Extract Method โดยแยกส่วนของเมทธอดนั้น ที่ถูกเรียกใช้โดยบางคลาส มาสร้างเป็นเมทธอดใหม่ และย้ายเมทธอดเหล่านั้นไปยังคลาสที่มีการเรียกใช้ต่างกัน

2) การตรวจหา Feature Envy สำหรับคุณลักษณะ

พิจารณาคุณลักษณะที่มีขอบเขตเป็นพับบลิค ทั้งหมดของทุกคลาส โดยการนับจำนวนการเรียกใช้คุณลักษณะภายในคลาสที่กำหนดคุณลักษณะนั้น เปรียบเทียบกับจำนวนการเรียกใช้คุณลักษณะภายในคลาสอื่น ถ้าพบว่าคุณลักษณะใด มีจำนวนการเรียกใช้ภายในคลาสที่กำหนดคุณลักษณะนั้นน้อยกว่าจำนวนการเรียกใช้ภายในคลาสอื่น ให้สงสัยว่าคุณลักษณะนั้นเป็นร่องรอยที่ไม่ดี ซึ่งสามารถปรับปรุงด้วย วิธี Move Attribute โดยย้ายคุณลักษณะนั้นไปยังคลาสที่มีจำนวนการเรียกใช้คุณลักษณะมากที่สุด

4. มาตรวัดร่องรอยที่ไม่ดี

วิธีการตรวจหา Feature Envy โดยใช้มาตรวัดร่องรอยที่ไม่ดี ดังนี้

1) มาตรวัด Number of called methods ($NCDM_m(C)$)

NCDM คือ จำนวนการเรียกใช้เมทธอดของคลาส *C* รวมถึงคลาสที่ *C* สืบทอดคุณสมบัติมาทั้งหมดภายในเมทธอด *m*

2) มาตรวัด Number of called attributes ($NCDA_m(C)$)

NCDA คือ จำนวนการเรียกใช้คุณลักษณะของคลาส C รวมถึงคลาสที่ *C* สืบทอดคุณสมบัติมาทั้งหมดภายในเมทธอด *m*

3) มาตรวัด Number of caller attributes ($NCRA_x(C)$)

NCRA คือ จำนวนการเรียกใช้คุณลักษณะ *x* ภายในคลาส *C* และคลาสที่สืบทอดคุณสมบัติมาทั้งหมด

5. ข้อกำหนดของค่าที่พิจารณา

จำแนกเป็น 2 ส่วน คือ ข้อกำหนดของค่าที่พิจารณาสำหรับเมทธอด และข้อกำหนดของค่าที่พิจารณาสำหรับคุณลักษณะ มีรายละเอียดดังนี้

กำหนดให้ *m* คือ เมทธอดที่สงสัยว่าเป็นร่องรอยที่ไม่ดี

x คือ คุณลักษณะที่สงสัยว่าเป็นร่องรอยที่ไม่ดี

C คือ คลาสทั้งหมดภายในซอฟต์แวร์

C_i คือ คลาสที่กำหนดเมทธอดหรือคุณลักษณะที่สงสัยว่าเป็นร่องรอยที่ไม่ดี

C_j คือ คลาสอื่นๆ ภายในซอฟต์แวร์ที่ไม่ใช่คลาส C_i

i, j เท่ากับ 1 ถึง *n* โดย *n* คือจำนวนคลาสทั้งหมดภายในซอฟต์แวร์

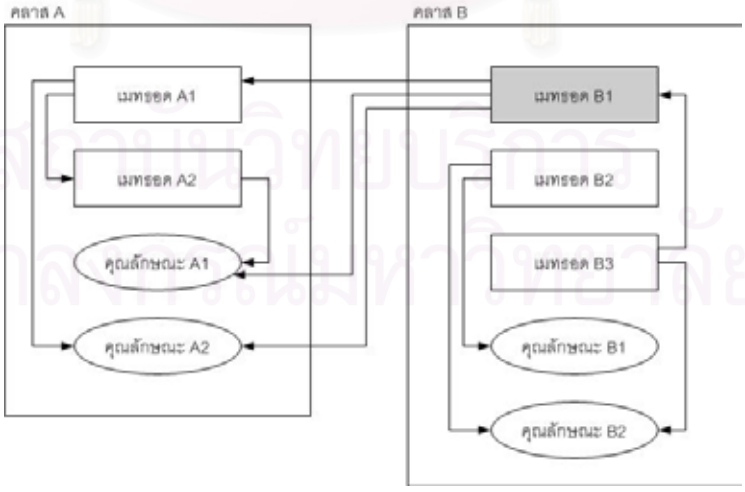
1) ข้อกำหนดของค่าที่พิจารณาสำหรับเมทธอด คือ เมทธอดที่มีค่ามาตรวัดอยู่ในช่วง $NCDM_m(C_i)+NCDA_m(C_i)<NCDM_m(C_j)+NCDA_m(C_j)$ ซึ่ง $C_i, C_j \in C$ และ $C_i \neq C_j$ นั่นคือ ภายในเมทธอด *m* มีการเรียกใช้คุณสมบัติของคลาส C_j มากกว่าคลาส C_i ดังนั้นควรจะย้ายเมทธอด *m* จากคลาสเจ้าของเมทธอดคือ C_i ไปยังคลาส C_j ที่มีการเรียกใช้คุณสมบัติมากกว่า

2) ข้อกำหนดของค่าที่พิจารณาสำหรับคุณลักษณะ คือ คุณลักษณะที่มีค่ามาตรวัดอยู่ในช่วง $NCRA_x(C_i)<NCRA_x(C_j)$ ซึ่ง $C_i, C_j \in C$ และ $C_i \neq C_j$ นั่นคือ คุณลักษณะ *x* ถูกเรียกใช้จากคลาส C_j มากกว่าคลาส C_i ดังนั้นควรจะย้ายคุณลักษณะ *x* จากคลาสเจ้าของคุณลักษณะนั้นคือ C_i ไปยังคลาส C_j ที่มีการเรียกใช้คุณสมบัติมากกว่า

6. การประยุกต์ใช้รีแฟคทอริง

วิธีการรีแฟคทอริงที่ใช้ปรับปรุง Feature Envy มี 3 วิธี คือ Move Method, Move Attribute และ Extract Method ซึ่งมีรายละเอียดดังนี้

1) วิธี Move Method ประยุกต์ใช้ เมื่อพบว่าค่ามาตรวัดอยู่ในช่วงข้อกำหนดของค่าที่พิจารณาสำหรับเมทธอด แสดงให้เห็นว่า ภายในเมทธอด *m* เรียกใช้คุณสมบัติของคลาส C_i น้อยกว่าคลาส C_j ดังนั้นควรจะย้ายเมทธอด *m* จากคลาส C_i ไปยังคลาส C_j ซึ่งตัวอย่างการคำนวณค่ามาตรวัด *NCDM* กับ *NCDA* และการประยุกต์ใช้วิธี Move Method แสดงในรูปที่ 3.3



รูปที่ 3.3 การเรียกใช้เมทธอดหรือคุณลักษณะภายในเมทธอดระหว่างคลาส *A* และ *B* ก่อนการประยุกต์ใช้วิธี Move Method

จากรูปที่ 3.3 พิจารณาเมทธอด *B1* นำมาตรวัดที่ได้ออกแบบไว้มาคำนวณได้ผลดังนี้

$NCDM_{B1}(A) = 1$, $NCDA_{B1}(A) = 2$, $NCDM_{B1}(B) = 0$ และ $NCDA_{B1}(B) = 0$

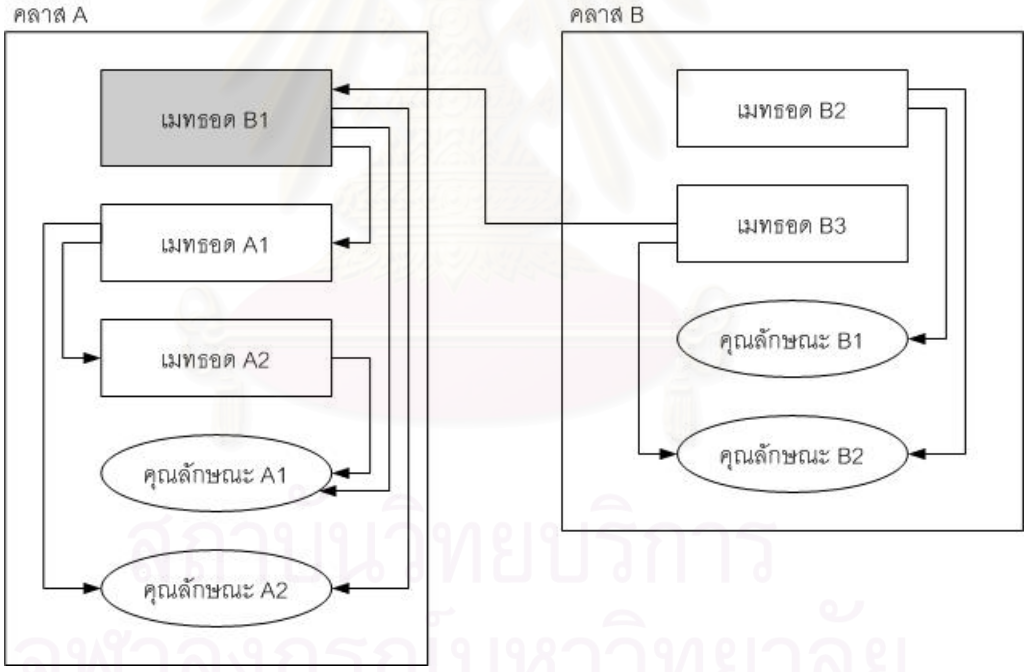
$NCDM_{B1}(A) + NCDA_{B1}(A) = 3$

$NCDM_{B1}(B) + NCDA_{B1}(B) = 0$

จากผลของค่ามาตรวัด พบว่าเมทธอด *B1* มีค่ามาตรวัดอยู่ในช่วงข้อกำหนดของค่าที่พิจารณาสำหรับเมทธอดตามสมการดังนี้

$NCDM_{B1}(B) + NCDA_{B1}(B) < NCDM_{B1}(A) + NCDA_{B1}(A)$

แสดงให้เห็นว่า ภายในเมทธอด *B1* เรียกใช้คุณสมบัติของคลาส *B* น้อยกว่าคุณสมบัติของคลาส *A* ดังนั้นควรจะย้ายเมทธอด *B1* จากคลาส *B* ไปยังคลาส *A* ซึ่งแสดงดังรูปที่ 3.4



รูปที่ 3.4 การเรียกใช้เมทธอดหรือคุณลักษณะภายในเมทธอดระหว่างคลาส *A* และ *B* หลังการประยุกต์ใช้วิธี Move Method

2) วิธี Move Attribute ประยุกต์ใช้เมื่อพบคุณลักษณะใดๆ มีค่ามาตรวัดอยู่ในช่วงข้อกำหนดของค่าที่พิจารณาสำหรับคุณลักษณะ แสดงให้เห็นว่าคุณลักษณะ *x* ถูกเรียกใช้จากคลาส C_j มากกว่าคลาส C_i ดังนั้นควรจะย้ายคุณลักษณะ *x* จากคลาส C_i ไปยังคลาส C_j

3) วิธี Extract Method ประยุกต์ใช้เมื่อพบเมทธอดใดๆ มีค่ามาตรวัดอยู่ในช่วงข้อกำหนดของค่าที่พิจารณาสำหรับเมทธอด มากกว่าหนึ่งเงื่อนไข คือมีคลาส C_j มากกว่าหนึ่งคลาส ที่มีค่ามาตรวัดอยู่ในช่วงข้อกำหนดของค่าที่พิจารณาสำหรับเมทธอด แสดงให้เห็นว่า ภายในเมทธอด *m* มีการเรียกใช้คุณสมบัติของคลาสอื่นมากกว่าหนึ่งคลาส ดังนั้นควรจะแยกส่วนของเมทธอด *m* ตามการเรียกใช้คุณสมบัติของคลาสนั้นๆ มาสร้างเป็นเมทธอดใหม่ และย้ายเมทธอดเหล่านั้น ไปยังคลาสต่างๆ ตามที่มีการเรียกใช้งาน

Large Class

1. นิยาม

คือ คลาสที่มีหน้าที่การทำงาน และตัวแปรอินสแตนท์ (Instance variable) จำนวนมาก

2. แรงจูงใจ

ในการพัฒนาซอฟต์แวร์เชิงวัตถุนั้น การสร้างคลาสที่มีขนาดใหญ่เป็นสาเหตุทำให้เกิดโค้ดที่ซ้ำกัน (Duplicate code) ได้ เป็นผลให้โค้ดมีความซับซ้อน และทำให้การบำรุงรักษายากมากขึ้น ซึ่งการสร้างคลาสที่มีขนาดเล็กสามารถนำกลับมาใช้ใหม่ได้ง่ายกว่าคลาสที่มีขนาดใหญ่ ดังนั้นควรจะกำจัดความซ้ำซ้อนของโค้ดภายในคลาสให้หมดไป เพื่อลดขนาดของคลาสให้เล็กลง

3. วิธีการวัด

การตรวจหา Large Class สามารถทำโดย แบ่งการพิจารณาเป็น 2 ส่วนคือ ขนาด และการเกาะกันเป็นก้อน มีรายละเอียดดังนี้

1) พิจารณาคลาสทั้งหมดภายในซอฟต์แวร์ หาขนาดของคลาส โดยคำนวณจากจำนวนเมทธอด และคุณลักษณะที่กำหนดภายในคลาสทั้งหมด เลือกคลาสที่มีขนาดมากกว่าค่าที่กำหนดไว้ในข้อกำหนดของค่าที่พิจารณา

2) จากนั้นเลือกคลาสที่มีการเกาะกันเป็นก้อนต่ำที่สุด

เนื่องจากงานวิจัยของ Lorenz และ Kidd [12] ได้เสนอว่ารูปแบบการเรียกใช้ตัวแปรอินสแตนท์สามารถนำมาช่วยในการแบ่งคลาสได้ ดังนั้นสามารถประยุกต์ใช้วิธี Extract Class โดยเลือกเมทธอดที่เรียกใช้ตัวแปรอินสแตนท์ตัวเดียวกันมาสร้างเป็นคลาสใหม่ แต่ถ้าบางส่วนของเมทธอดหรือคุณลักษณะถูกเรียกใช้ โดยบางตัวแปรอินสแตนท์ของคลาสนั้น แล้วเมทธอดหรือคุณลักษณะเหล่านั้น ควรจะนำมาสร้างเป็นคลาสลูก (Subclass)

โดยประยุกต์ใช้วิธี Extract Subclass และสามารถประยุกต์ใช้วิธี Extract Interface เมื่อหลายๆ ไคลเอนท์ (Clients) ใช้เฉพาะบางส่วนของอินเตอร์เฟสของ Large Class นั้นเหมือนกัน หรือมีมากกว่าหนึ่งคลาสที่มีบางส่วนของอินเตอร์เฟสเหมือนกัน กรณีสุดท้ายคือถ้า Large Class นั้นเป็นคลาสที่ทำหน้าที่เป็นส่วนที่ติดต่อกับผู้ใช้งาน (Graphic User Interface – GUI) ควรจะประยุกต์ใช้วิธี Duplicate Observed Data

4. มาตรวัดร่องรอยที่ไม่ดี

วิธีการตรวจหา Large Class โดยใช้มาตรวัดร่องรอยที่ไม่ดี ดังนี้

1) มาตรวัด Number of instance method in a class (*NIM*) [12]

NIM คือ จำนวนเมทธอดทั้งหมดที่กำหนดภายในคลาส ทั้งพับบลิค ไพรเวท และโพรเทค (Protected)

2) มาตรวัด Number of instance variables in a class (*NIV*) [12]

NIV คือ จำนวนตัวแปรทั้งหมดที่ประกาศภายในคลาส ทั้งพับบลิค ไพรเวท และโพรเทค

3) มาตรวัด Tight class cohesion (*TCC*) [10]

TCC คือ จำนวนความสัมพันธ์ของเมทธอดที่สัมพันธ์กันทางตรง ซึ่งคำนวณได้จาก $TCC = NDC(C) / NP(C)$

กำหนดให้

Number of direct connection (*NDC*(*C*)) คือ จำนวนของความสัมพันธ์ทางตรงระหว่างเมทธอดภายในคลาส *C* นั่นคือ เมทธอด 2 เมทธอดที่จะมีความสัมพันธ์ทางตรงเมื่อทั้ง 2 เมทธอดมีการใช้งานคุณลักษณะตัวเดียวกันของคลาส แต่ไม่รวมถึงความสัมพันธ์ระหว่างเมทธอดกับคอนสตรัคเตอร์

Number of pairs of abstracted methods (*NP*) คือ ค่าที่เป็นไปได้มากที่สุดของความสัมพันธ์ทั้งทางตรง และทางอ้อมในคลาส คำนวณได้จาก $N * (N-1)/2$ โดยที่ *N* คือจำนวนเมทธอดทั้งหมดภายในคลาส ยกเว้นคอนสตรัคเตอร์

4) มาตรวัด Set of method used by instance variable in a class ($SMIV(V_i)$)

SMIV คือ เซตของเมทธอดที่มีการเรียกใช้ที่ตัวแปรอินสแตนท์ V_i เมื่อ V_i คือตัวแปรทุกตัวที่ประกาศไว้ในคลาส

5) มาตรวัด Set of feature used by instance variable of a class ($SFIV(V_i)$)

SFIV คือ เซตของเมทธอดและคุณลักษณะที่ถูกเรียกใช้โดยตัวแปรอินสแตนท์ V_i ของคลาส เมื่อ V_i คือตัวแปรที่มีชนิดเป็นคลาสที่พิจารณาที่ประกาศในไคลเอนท์

5. ข้อกำหนดของค่าที่พิจารณา [12]

พิจารณาคลาสทั้งหมดในซอฟต์แวร์ เลือกคลาสที่มีค่า *NIM* >20 สำหรับคลาสทั่วไป แต่ถ้าคลาสทำหน้าที่เป็นส่วนติดต่อกับผู้ใช้งานมีค่า *NIM* >40 และค่า *NIV* >3 สำหรับคลาสทั่วไป แต่ถ้าคลาสทำหน้าที่เป็นส่วนติดต่อกับผู้ใช้งานมีค่า *NIV* >9 จากนั้นเลือกคลาสที่มี *TCC* ต่ำที่สุด ($\min(TCC)$)

6. การประยุกต์ใช้รีแฟคทอริง

วิธีการรีแฟคทอริงที่ใช้ปรับปรุง Large Class มี 4 วิธี คือ Extract Class, Extract Subclass, Extract Interface และ Duplicate Observed Data ซึ่งมีรายละเอียดดังนี้

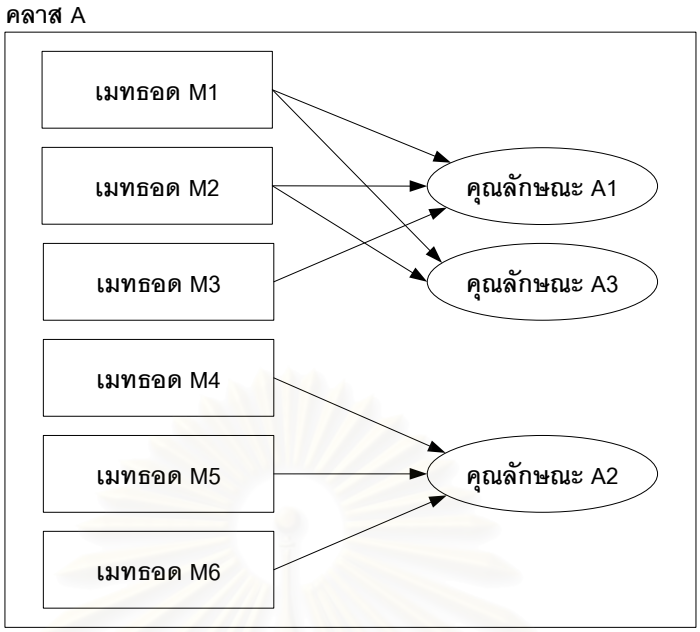
1) วิธี Extract Class ประยุกต์ใช้ เมื่อพบว่า $SMIV(V_i) \cap SMIV(V_j) = \varnothing$ ซึ่ง $V_{i,}\ V_j \in V$ และ $V_i \neq V_j$

กำหนดให้

$V_{i,}\ V_j$ คือ ตัวแปรอินสแตนท์ทั้งหมดภายในคลาสที่สงสัยว่าเป็นร่องรอยที่ไม่ดี

i, j เท่ากับ 1 ถึง *n* โดย *n* คือจำนวนตัวแปรอินสแตนท์ทั้งหมดภายในคลาสที่สงสัยว่าเป็นร่องรอยที่ไม่ดี

ถ้าเงื่อนไขดังกล่าวข้างต้นเป็นจริง แสดงให้เห็นว่า มีคู่ของตัวแปรอินสแตนท์ที่ไม่ได้ถูกเรียกใช้ร่วมกันในเมทธอด ดังนั้นสามารถแยกตัวแปรอินสแตนท์คู่นั้นและย้ายเมทธอดที่เรียกใช้ตัวแปรอินสแตนท์ทั้งสองนั้นออกจากกันได้ ซึ่งตัวอย่างการประยุกต์ใช้วิธี Extract Class แสดงในรูปที่ 3.5



รูปที่ 3.5 การเรียกใช้ตัวแปรอินสแตนท์จากเมทธอดภายในคลาส *A*

จากรูปที่ 3.5 คลาส *A* ประกอบด้วย 6 เมทธอด (*M1, M2, M3, M4, M5* และ *M6*) และ 3 ตัวแปรอินสแตนท์ (*A1, A2* และ *A3*) กำหนดให้คลาส *A* เป็นคลาสที่สงสัยว่าเป็นร่องรอยที่ไม่ดีคือมีค่ามาตรวัด *NIM*, *NIV* และ *TCC* อยู่ในช่วงของข้อกำหนดที่พิจารณาของ Large Class จากนั้นคำนวณค่ามาตรวัด *SMIV* และได้ผลการคำนวณค่ามาตรวัดของคลาส *A* ดังนี้

$$SMIV(A1) = \{M1,\ M2,\ M3\}$$
$$SMIV(A2) = \{M4,\ M5,\ M6\}$$
$$SMIV(A3) = \{M1,\ M2\}$$

จากนั้นคำนวณเงื่อนไขของการประยุกต์ใช้วิธี Extract Class ได้ผลลัพธ์แสดงดังนี้

$$SMIV(A1) \cap SMIV(A2) = \varnothing$$
$$SMIV(A1) \cap SMIV(A3) = \{M1,\ M2\}$$
$$SMIV(A2) \cap SMIV(A3) = \varnothing$$

จากผลลัพธ์ข้างต้น จะเห็นว่ามีคู่ของตัวแปรอินสแตนท์ *A1*, *A2* และ *A2*, *A3* ไม่ได้ถูกใช้ร่วมกันในเมทธอดใดๆ ดังนั้นสามารถประยุกต์ใช้วิธี Extract Class แสดงในรูปที่ 3.6

รูปที่ 3.6 การประยุกต์ใช้วิธี Extract Class

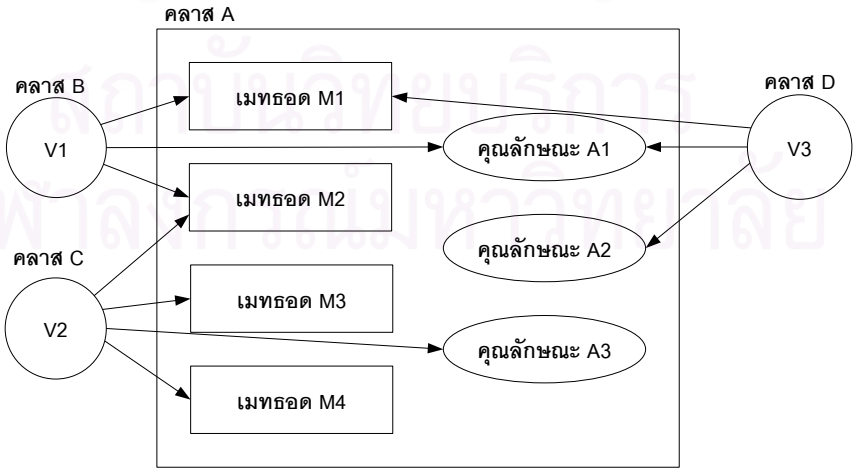
2) วิธี Extract Subclass ประยุกต์ใช้ เมื่อพบว่า $SFIV(V_i) \cap SFIV(V_j) = \varnothing$ ซึ่ง $V_i, V_j \in V$ และ $V_i \neq V_j$

กำหนดให้

V_i, V_j คือ ตัวแปรอินสแตนท์ของคลาสที่สงสัยว่าเป็นร่องรอยที่ไม่ดีทั้งหมด ที่ถูกประกาศใช้ในคลาสอื่น

i, *j* เท่ากับ 1 ถึง *n* โดย *n* คือจำนวนตัวแปรอินสแตนท์ของคลาสที่สงสัยว่าเป็นร่องรอยที่ไม่ดีทั้งหมด

ถ้าเงื่อนไขดังกล่าวข้างต้นเป็นจริง แสดงให้เห็นว่า มีตัวแปรอินสแตนท์ที่เรียกใช้คุณสมบัติเพียงบางส่วนของคลาส ดังนั้นสามารถแยกคุณสมบัตินั้นมา สร้างเป็นคลาสลูก เพื่อใช้ในการประกาศตัวแปรอินสแตนท์ของคลาสลูกนั้นมาใช้งานแทน รูปที่ 3.7 แสดงตัวอย่างของคุณสมบัติของคลาสที่ควรจะย้ายจากคลาสเดิม มาสร้างเป็นคลาสลูกใหม่ โดยการประยุกต์ใช้วิธี Extract Subclass



รูปที่ 3.7 การเรียกใช้เมทธอดและคุณลักษณะที่ตัวแปรอินสแตนท์ของคลาส *A* เรียกใช้

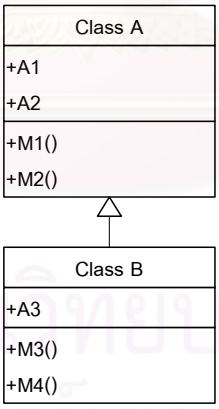
จากรูปที่ 3.7 คลาส *A* ประกอบด้วย 4 เมทธอด (*M1*, *M2*, *M3*, และ *M4*) 3 คุณลักษณะ (*A1*, *A2* และ *A3*) และตัวแปรอินสแตนท์ของคลาส *A* ที่ถูกเรียกใช้ในคลาส *B*, *C* และ *D* (*V1*, *V2* และ *V3*) กำหนดให้คลาส *A* เป็นคลาสที่สงสัยว่าเป็นร่องรอยที่ไม่ดีคือมีค่ามาตรวัด *NIM*, *NIV* และ *TCC* อยู่ในช่วงของข้อกำหนดที่พิจารณาของ Large Class จากนั้นคำนวณค่ามาตรวัด *SFIV* และได้ผลการคำนวณมาตรวัดของคลาส *A* ดังนี้

$$SFIV(V1) = \{A1,\ M1,\ M2\}$$
$$SFIV(V2) = \{A3,\ M2,\ M3,\ M4\}$$
$$SFIV(V3) = \{A1,\ A2,\ M1\}$$

จากนั้นคำนวณเงื่อนไขของการประยุกต์ใช้วิธี Extract Subclass ได้ผลลัพธ์แสดงดังนี้

$$SFIV(V1) \cap SFIV(V2) = \{M2\}$$
$$SFIV(V1) \cap SFIV(V3) = \{A1,\ M1\}$$
$$SFIV(V2) \cap SFIV(V3) = \varnothing$$

จากผลลัพธ์ข้างต้น จะเห็นว่ามีคู่ตัวแปรอินสแตนท์ *V2* และ *V3* ไม่ได้เรียกใช้เมทธอดหรือคุณลักษณะของคลาส *A* ร่วมกัน ดังนั้นสามารถประยุกต์ใช้วิธี Extract Subclass ได้ แสดงในรูปที่ 3.8



รูปที่ 3.8 แผนภาพคลาสหลังการประยุกต์ใช้วิธี Extract Subclass

3) วิธี Extract Interface ประยุกต์ใช้ เมื่อไคลเอนท์ เรียกใช้งานเพียงบางส่วนของอินเตอร์เฟสของ Large Class หรือมีมากกว่าหนึ่งคลาสเรียกใช้งานเพียงบางส่วนของอินเตอร์เฟสร่วมกัน ดังนั้นส่วนของอินเตอร์เฟสของ Large Class ที่ถูกใช้ร่วมกัน ควรจะนำมาสร้างเป็นอินเตอร์เฟสใหม่

4) วิธี Duplicate Observed Data ประยุกต์ใช้ เมื่อคลาสที่สงสัยว่าเป็น Large Class นั้นเป็นคลาสที่ทำงานเป็นส่วนที่ติดต่อกับผู้ใช้งาน คือ javax.swing.* และ java.awt.* ดังนั้นควรแบ่งคลาสเป็น 2 โดเมน (Domain) คือ โดเมนของข้อมูลสำหรับการควบคุมส่วนที่ติดต่อกับผู้ใช้งาน และโดเมนของเมทธอดที่ให้คลาสอื่นเรียกใช้ จากนั้นสร้างอ็อบเจกต์ (Object) ใหม่เหมือนกับโดเมนข้อมูล และสร้างออบเซิร์บเวอร์ (Observer) เพื่อซิงโครไนส์ (Synchronize) โดเมนข้อมูลทั้งสองโดเมน

Lazy Class

1. นิยาม

   คือ คลาสที่ไม่มีหน้าที่การทำงานที่สำคัญมาก

2. แรงจูงใจ

   การสร้างคลาสขึ้นมาทำงานภายในซอฟต์แวร์ ผู้พัฒนาต้องเสียเวลา และค่าใช้จ่ายในการบำรุงรักษาซอฟต์แวร์ ดังนั้น Lazy Class ซึ่งไม่ได้มีความรับผิดชอบที่สำคัญมาก ควรจะถูกกำจัดออก เพื่อลดเวลา และค่าใช้จ่ายลง ซึ่ง Lazy Class มีความหมายตรงข้ามกับ Large Class จะเกิดเมื่อคลาสมีฟังก์ชันที่ประกาศในอินเตอร์เฟสน้อย หรือมีความแตกต่างของฟังก์ชันที่ประกาศในอินเตอร์เฟสระหว่างคลาสแม่และคลาสลูกน้อย

3. วิธีการวัด

   การตรวจหา Lazy Class สามารถทำได้โดย แบ่งการพิจารณาเป็น 2 ส่วนคือ ขนาดและการสืบทอดคุณสมบัติ (Inherit) มีรายละเอียดดังนี้ พิจารณาคลาสทั้งหมดภายในซอฟต์แวร์ หาคลาสที่มีจำนวนเมทธอด และคุณลักษณะที่กำหนดภายในคลาสทั้งหมดจำนวนน้อย จากนั้นพิจารณาที่การสืบทอดคุณสมบัติ ถ้าคลาสนั้นเป็นคลาสที่สืบทอดคุณสมบัติมาจากคลาสแม่ ควรประยุกต์ใช้วิธี Collapse Hierarchy โดยรวมคุณสมบัติของคลาสแม่และคลาสลูกเข้าด้วยกัน แต่ถ้าไม่มีการสืบทอดคุณสมบัติแล้ว ควรประยุกต์ใช้วิธี Inline Class เพื่อรวมคลาสที่สงสัยว่าเป็น Lazy Class เข้ากับคลาสที่มีการเรียกใช้คลาสนี้

4. มาตรวัดร่องรอยที่ไม่ดี

   วิธีการตรวจหา Lazy Class โดยใช้มาตรวัดร่องรอยที่ไม่ดี ดังนี้

   1) มาตรวัด Number of instance method in a class (*NIM*) [12]

      รายละเอียดเหมือนกับมาตรวัดร่องรอยที่ไม่ดีของ Large Class

   2) มาตรวัด Number of instance variables in a class (*NIV*) [12]

รายละเอียดเหมือนกับมาตรวัดร่องรอยที่ไม่ดีของ Large Class

3) มาตรวัด Depth of inheritance hierarchy (*DIT*) [9]

DIT คือ ความลึกของการสืบทอดคุณสมบัติของคลาส ในกรณีที่มีการสืบทอดคุณสมบัติจากหลายคลาส *DIT* คือความลึกของการสืบทอดคุณสมบัติของคลาสที่มากที่สุดจากรูธโหนด (Root node)

5. ข้อกำหนดของค่าที่พิจารณา

คลาสที่มีค่ามาตรวัดอยู่ในช่วงผลรวมระหว่าง *NIM* และ *NIV* มีค่าน้อยที่สุดภายในซอฟต์แวร์ ($\min(NIM + NIV)$)

6. การประยุกต์ใช้รีแฟคทอริง

วิธีการรีแฟคทอริงที่ใช้ปรับปรุง Lazy Class มี 2 วิธี คือ Inline Class และ Collapse Hierarchy ซึ่งมีรายละเอียดดังนี้

1) วิธี Inline Class ประยุกต์ใช้ เมื่อคลาสที่สงสัยว่าเป็นร่องรอยที่ไม่ดี มีค่ามาตรวัด *DIT* = 0 แสดงว่าคลาสนั้นไม่มีการสืบทอดคุณสมบัติจากคลาสอื่น จึงควรจะย้ายคุณสมบัติทั้งหมดของคลาสนั้น ไปยังคลาสที่มีการเรียกใช้แทน และลบคลาสเดิมทิ้ง

2) วิธี Collapse Hierarchy ประยุกต์ใช้ เมื่อคลาสที่สงสัยว่าเป็นร่องรอยที่ไม่ดี มีค่ามาตรวัด *DIT* > 0 แสดงว่าคลาสนั้นมีการสืบทอดคุณสมบัติจากคลาสอื่น จึงควรจะย้ายคุณสมบัติทั้งหมดของคลาสนั้น ไปยังคลาสแม่ที่สืบทอดคุณสมบัติมา และลบคลาสเดิมทิ้ง

ตัวอย่างการประยุกต์ใช้วิธี Inline Class และ Collapse Hierarchy แสดงรายละเอียดในภาคผนวก ข

Long Method

1. นิยาม

คือ เมทธอดที่มีขนาดใหญ่ ซึ่งมีจำนวนพารามิเตอร์ (Parameter) และจำนวนตัวแปรชั่วคราว (Temporary variable) มาก

2. แรงจูงใจ

ร่องรอยที่ไม่ดีนี้ เกิดขึ้นเมื่อมีการสร้างส่วนโปรแกรม (Component) ที่ใหญ่ คือเมทธอดที่มีความซับซ้อน มีจำนวนพารามิเตอร์ และตัวแปรชั่วคราวมาก ซึ่งคุณสมบัติของการเขียน

โปรแกรมเชิงวัตถุ (Object-Oriented Programming) ที่สำคัญคือ เพื่อออกแบบส่วนโปรแกรมให้มีขนาดเล็ก เพื่อง่ายต่อการทำความเข้าใจ และบำรุงรักษา แต่ร่องรอยที่ไม่ดีนี้ทำให้ขาดคุณสมบัติในข้อนี้ และทำให้การทำความเข้าใจเมทธอดยากขึ้น

3. วิธีการวัด

การตรวจหา Long Method สามารถทำได้โดย แบ่งการพิจารณาเมทธอดทั้งหมด ภายในซอฟต์แวร์ หาเมทธอดที่มีขนาดใหญ่ คือมีจำนวนสเตทเมนต์ (Statement) มาก

4. มาตรวัดร่องรอยที่ไม่ดี

วิธีการตรวจหา Long Method โดยใช้มาตรวัดร่องรอยที่ไม่ดี ดังนี้

1) มาตรวัด Number of statement in a method (*NOS*) [12]

NOS คือ จำนวนสเตทเมนต์ของเมทธอด

2) มาตรวัด Number of parameters in a method (*NOP*) [13]

NOP คือ จำนวนพารามิเตอร์ที่ปรากฏอยู่ในซิกเนเจอร์ของเมทธอด

3) มาตรวัด Number of temporary variables in a method (*NOT*)

NOT คือ จำนวนตัวแปรชั่วคราวที่ประกาศใช้ในเมทธอด

5. ข้อกำหนดของค่าที่พิจารณา [12]

พิจารณาเมทธอดทั้งหมดภายในซอฟต์แวร์ เลือกเมทธอดที่มีค่ามาตรวัด $NOS > 7$

6. การประยุกต์ใช้รีแฟคทอริง

วิธีการรีแฟคทอริงที่ใช้ปรับปรุง Long Method มี 6 วิธี คือ Extract Method, Replace Method with Method Object, Replace Temp with Query, Introduce Parameter Object, Preserve whole Object และ Decompose Conditional ซึ่งมีรายละเอียดดังนี้

1) วิธี Extract Method และ Replace Method with Method Object ประยุกต์ใช้เมื่อเมทธอดที่สงสัยว่าเป็นร่องรอยที่ไม่ดี มีค่าผลรวมของมาตรวัด *NOP* และ *NOT* มากที่สุดในซอฟต์แวร์ ($\max(NOP + NOT)$)

2) วิธี Replace Temp with Query ประยุกต์ใช้เมื่อเมทธอดที่สงสัยว่าเป็นร่องรอยที่ไม่ดี มีค่ามาตรวัด *NOT* มากที่สุดในซอฟต์แวร์ ($\max(NOT)$)

3) วิธี Introduce Parameter Object และ Preserve Whole Object ประยุกต์ใช้เมื่อเมทธอดที่สงสัยว่าเป็นร่องรอยที่ไม่ดี มีค่ามาตรวัด *NOP* มากที่สุดในซอฟต์แวร์ ($\max(NOP)$)

4) วิธี Decompose Conditional ประยุกต์ใช้เมื่อบางส่วนของเมทธอดที่สงสัยว่าเป็นร่องรอยที่ไม่ดีเป็นเงื่อนไข (Conditional) หรือวนซ้ำ (Loop)

ตัวอย่างการประยุกต์ใช้วิธี Extract Method, Replace Method with Method Object, Replace Temp with Query, Introduce Parameter Object, Preserve whole Object และ Decompose Conditional แสดงรายละเอียดในภาคผนวก ข

Long Parameter Lists

1. นิยาม

   คือ การส่งผ่านทุกๆ ค่าที่เมทธอดต้องการใช้ไปยังเมทธอดนั้น โดยผ่านค่าพารามิเตอร์

2. แรงจูงใจ

   ในการเขียนโปรแกรมเชิงวัตถุ ถ้าเมทธอดต้องการใช้ค่าใดๆ สามารถส่งข้อความ (Message send) ไปยังอ็อบเจกต์ เพื่อร้องขอค่าที่ต้องการใช้ ดังนั้นผู้เขียนโปรแกรมไม่จำเป็นต้องส่งทุกๆ ค่าที่เมทธอดต้องการใช้ผ่านพารามิเตอร์ไปยังเมทธอดนั้น เพราะอาจทำให้การทำความเข้าใจโค้ดนั้นยากขึ้น

3. วิธีการวัด

   การตรวจหา Long Parameter Lists สามารถทำโดยพิจารณาเมทธอดที่มีจำนวนพารามิเตอร์ที่ประกาศในซิกเนเจอร์มาก

4. มาตรวัดร่องรอยที่ไม่ดี

   วิธีการตรวจหา Long Parameter Lists โดยใช้มาตรวัดร่องรอยที่ไม่ดี คือ มาตรวัด Number of parameters in a method (*NOP*) [13] ซึ่งรายละเอียดเหมือนกับมาตรวัดร่องรอยที่ไม่ดีใน Long Method

5. ข้อกำหนดของค่าที่พิจารณา

   พิจารณาเมทธอดทั้งหมดภายในซอฟต์แวร์ เลือกเมทธอดที่มีค่ามาตรวัด *NOP* มากที่สุดภายในซอฟต์แวร์ ($\max(NOP)$)

6. การประยุกต์ใช้รีแฟคทอริง

วิธีการรีแฟคทอริงที่ใช้ปรับปรุง Long Parameter Lists มี 3 วิธี คือ Replace Parameters with Method, Preserve Whole Object และ Introduce Parameter Object ซึ่งมีรายละเอียดดังนี้

1) วิธี Replace Parameters with Method ประยุกต์ใช้เมื่อค่าพารามิเตอร์นั้น ขึ้นกับการเรียกใช้เมทธอด ดังนั้นสามารถลบค่าพารามิเตอร์นั้น แล้วเปลี่ยนมาเป็นการเรียกใช้เมทธอดแทน

2) วิธี Preserve Whole Object ประยุกต์ใช้เมื่อค่าพารามิเตอร์ที่ส่งมาทั้งหมดนั้น มาจากอ็อบเจกต์ตัวเดียวกัน ดังนั้นสามารถลบค่าพารามิเตอร์เหล่านั้น และส่งอ็อบ-เจกต์ นั้นมาแทนได้

3) วิธี Introduce Parameter Object ประยุกต์ใช้เมื่อพบว่ามีกลุ่มของพารามิเตอร์ชุดเดียวกัน ถูกใช้ในหลายๆ เมทธอด ดังนั้นสามารถแทนที่กลุ่มของพารามิเตอร์นั้น ด้วยอ็อบเจกต์ ได้

ตัวอย่างการประยุกต์ใช้วิธี Replace Parameters with Method, Preserve Whole Object และ Introduce Parameter Object แสดงรายละเอียดในภาคผนวก ข

Switch Statements

1. นิยาม

คือ การปรากฏของสวิทช์สเตทเมนท์เหมือนกันในโค้ดหลายๆ ที่

2. แรงจูงใจ

ปัญหาของสวิทช์สเตทเมนท์คือ ทำให้เกิดโค้ดซ้ำกัน ดังนั้นโปรแกรมเชิงวัตถุจึงใช้แนวคิดโพลีมอร์ฟิซึม (Polymorphism) เพื่อช่วยแก้ปัญหานี้

3. วิธีการวัด

การตรวจหา Switch Statement สามารถทำโดยหาตำแหน่งของสวิทช์สเตทเมนท์ภายในโค้ด

4. มาตรวัดร่องรอยที่ไม่ดี

วิธีการตรวจหา Switch Statement โดยใช้มาตรวัดร่องรอยที่ไม่ดี คือ มาตรวัด Number of switch statement in a method (*NOSS*) คือ จำนวนสวิทช์สเตทเมนท์ภายในเมทธอด

5. ข้อกำหนดของค่าที่พิจารณา

พิจารณาเมทธอดทั้งหมดภายในซอฟต์แวร์ เลือกเมทธอดที่มีค่ามาตรวัด *NOSS* > 0

6. การประยุกต์ใช้รีแฟคทอริง

วิธีการรีแฟคทอริงที่ใช้ปรับปรุง Switch Statement มี 7 วิธี คือ Extract Method, Move Method, Replace Type Code with Subclasses, Replace Type Code with State/Strategy, Replace Conditional with Polymorphism, Replace Parameter with Explicit Methods และ Introduce Null Object ซึ่งมีรายละเอียดดังนี้

1) วิธี Extract Method ประยุกต์ใช้เพื่อย้ายโค้ดส่วนที่เป็นสวิทซ์สเตทเมนท์มา สร้างเป็นเมทธอดใหม่ และใช้วิธี Move Method เพื่อย้ายเมทธอดนั้นไปยังคลาสที่มีโพลีมอร์ฟิ-ซึม หลังจากนั้นประยุกต์ใช้วิธี Replace Type Code with Subclasses หรือ Replace Type Code with State/Strategy เพื่อใช้แทนที่โค้ดที่ถูกย้ายไป

2) วิธี Replace Conditional with Polymorphism ประยุกต์ใช้เมื่อมีการกำหนดโครงสร้างการสืบทอดคุณสมบัติไว้

3) วิธี Replace Parameter with Explicit Methods ประยุกต์ใช้เมื่อบางเคสสเตทเมนต์ (Case statement) มีผลกระทบกับเมทธอดเพียงเมทธอดเดียว

4) วิธี Introduce null object ประยุกต์ใช้เมื่อพบว่าเงื่อนไขหนึ่งในเคสสเตทเมนต์เป็นค่าว่าง (Null)

ตัวอย่างการประยุกต์ใช้วิธี Extract Method, Move Method, Replace Type Code with Subclasses, Replace Type Code with State/Strategy, Replace Conditional with Polymorphism, Replace Parameter with Explicit Methods และ Introduce Null Object อยู่ในภาคผนวก ข

บทที่ 4

การออกแบบและพัฒนาเครื่องมือช่วยในการตรวจจับร่องรอยที่ไม่ดี

ในบทนี้จะอธิบายรายละเอียดเกี่ยวกับขั้นตอนการออกแบบและพัฒนาเครื่องมือ ช่วยในการตรวจจับร่องรอยที่ไม่ดี จะใช้แผนภาพยูเอ็มแอล (UML Diagram) ประกอบด้วย แผนภาพยูสเคส (Use case diagram) แผนภาพคลาส (Class diagram) แผนภาพซีเควนซ์ (Sequence diagram) และการตรวจสอบค่ามาตรวัดที่ได้จากเครื่องมือ ซึ่งมีรายละเอียดดังนี้

รูปที่ 4.1 แผนภาพยูสเคสของเครื่องมือช่วยในการตรวจจับร่องรอยที่ไม่ดี

4.1 แผนภาพยูสเคส

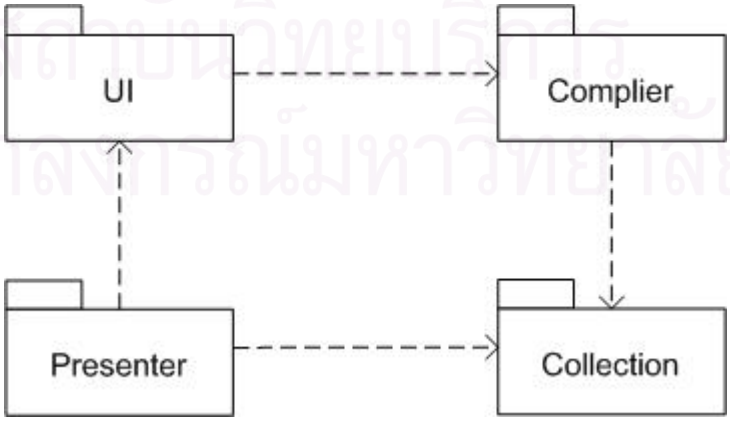
แผนภาพยูสเคสเป็นแผนภาพแสดงเรื่องราวทั้งหมดของขอบเขตของปัญหาว่าต้องประกอบด้วยฟังก์ชันใด และแต่ละฟังก์ชันมีความสัมพันธ์กันอย่างไรในมุมมองของผู้ใช้ ซึ่งแผนภาพยูสเคสของเครื่องมือช่วยในการตรวจจับร่องรอยที่ไม่ดี แสดงดังรูปที่ 4.1

ผู้ใช้สามารถใช้งานและติดต่อกับเครื่องมือนี้ได้ 8 กรณี คือ ผู้ใช้สามารถกำหนดโปรแกรมต้นฉบับภาษาจาวาที่ต้องการหาค่ามาตรวัด ผู้ใช้สามารถดูค่ามาตรวัดร่องรอยที่ไม่ดีและวิธีการรีแฟคทอริงที่เสนอเพื่อแก้ไขร่องรอยที่ไม่ดี 6 ประเภท และผู้ใช้สามารถเรียกดูค่ามาตรวัดร่องรอยที่ไม่ดีทั้งหมดสำหรับคลาส เมทธอด และคุณลักษณะของโปรแกรม

การที่ผู้ใช้จะสามารถดูค่ามาตรวัดต่างๆ เหล่านี้ได้จะต้องมีการคำนวณค่ามาตรวัดก่อน การคำนวณค่ามาตรวัดนี้จะต้องอาศัยการแปลงโปรแกรมต้นฉบับเป็นแผนภูมิต้นไม้ โดยใช้พาร์-เซอร์ของภาษาจาวา แล้วเก็บข้อมูลคุณสมบัติต่างๆ เช่น ชื่อคลาส (Class name) ชนิดของคลาส (Type of class) ตัวแปรอินสแตนท์ ชื่อเมทธอด (Method name) ชนิดของเมทธอด (Type of method) ชนิดของการส่งค่ากลับของเมทธอด (Return type of method) ตัวแปรที่ประกาศในเมท-ธอด (Local variables) เป็นต้น

4.2 แผนภาพคลาส

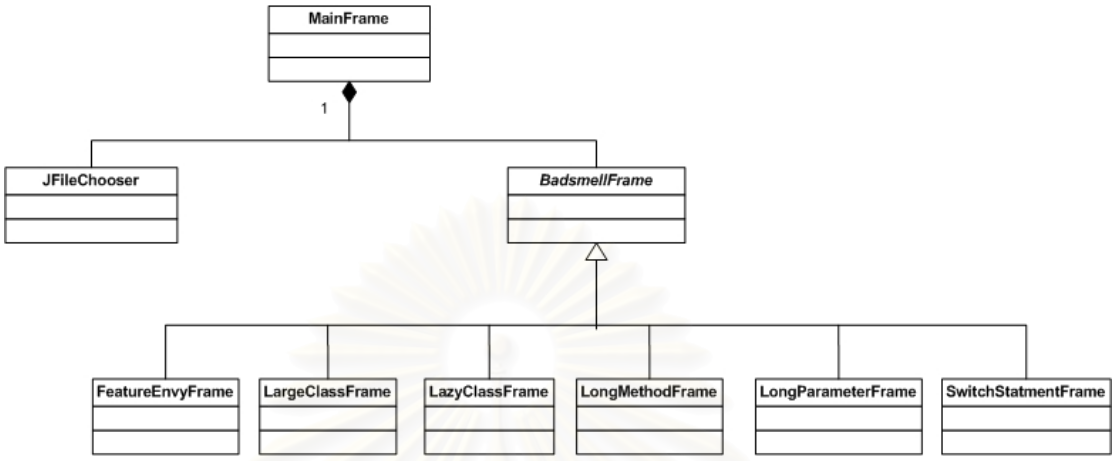
แผนภาพคลาสเป็นแผนภาพที่ใช้แสดงคลาส และความสัมพันธ์ระหว่างคลาส แผนภาพของระบบประกอบด้วยแผนภาพคลาส ซึ่งสามารถแบ่งออกเป็น 4 แพ็กเกจหลักคือแพ็กเกจส่วนติดต่อผู้ใช้ (UI) แพ็กเกจคอมไพเลอร์ (Compiler) แพ็กเกจคำนวณค่าตัววัด (Collection) และแพ็กเกจจัดรูปแบบการแสดงผลค่าตัววัด (Presenter) ซึ่งสามารถแสดงความสัมพันธ์ระหว่างแพ็กเกจต่าง ๆ ของระบบดังรูปที่ 4.2



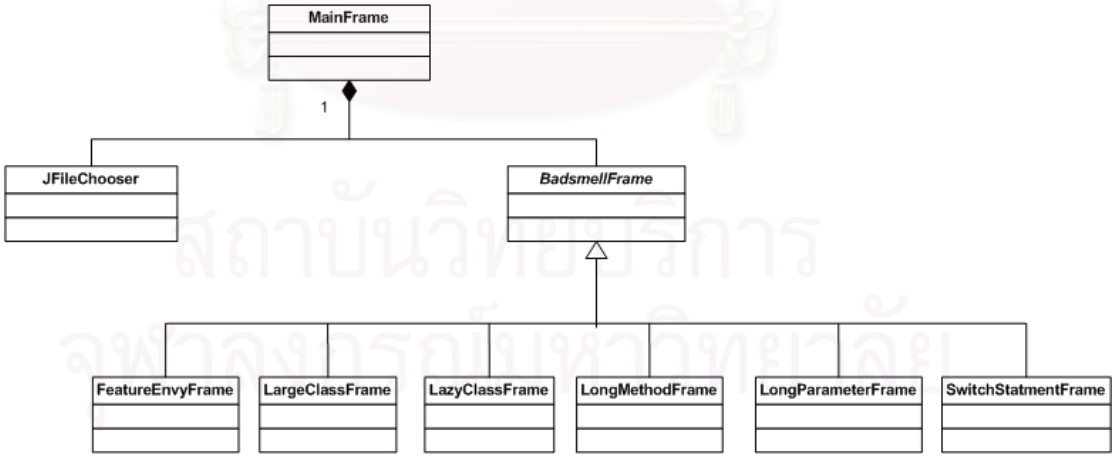
รูปที่ 4.2 แผนภาพคลาสแสดงความสัมพันธ์ระหว่างแพ็กเกจต่าง ๆ ของระบบ

4.2.1 แพ็กเกจส่วนติดต่อกับผู้ใช้

แพ็กเกจส่วนติดต่อผู้ใช้ เป็นชุดของคลาสที่ทำหน้าที่ติดต่อกับผู้ใช้งานระบบดังแสดงใน



รูปที่ 4.3 แพ็กเกจส่วนติดต่อผู้ใช้ ได้แก่ คลาส MainFrame เป็นคลาสที่ทำการสร้างเมนูให้ผู้ใช้เลือกและเป็นเฟรมหลักที่ให้เฟรมอื่นๆ แสดงผลในเฟรมหลักนี้ ได้แก่ คลาส JFileChooser เป็นคลาสที่ทำการเลือกโปรแกรมต้นฉบับภาษาจาวาเข้าสู่โปรเจค คลาส BadsmellFrame สำหรับดูค่ามาตรวัดร่องรอยที่ไม่ดีแยกตามประเภทต่างๆ 6 ประเภทคือ คลาส FeatureEnvyFrame คลาส LargeClassFrame คลาส LazyClassFrame คลาส LongMethodFrame คลาส LongParameterListFrame และคลาส SwitchStatementFrame ซึ่งคลาสต่างๆ ในแพ็กเกจนี้สืบทอดคุณสมบัติมาจากแพ็กเกจสวิง (Swing)



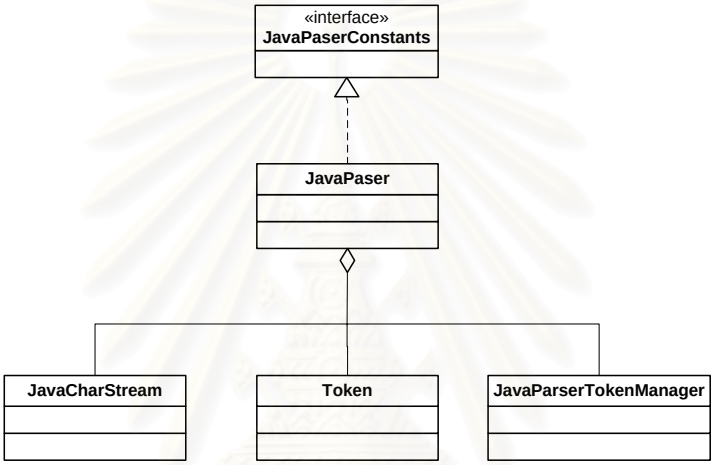
รูปที่ 4.3 แผนภาพคลาสในภาพรวมของแพ็กเกจส่วนติดต่อผู้ใช้

4.2.2 แพ็กเกจคอมไพเลอร์

แพ็กเกจคอมไพเลอร์ เป็นชุดของคลาสที่ทำหน้าที่นำโปรแกรมต้นฉบับมาผ่านขบวนการสร้างตัวแปลภาษาเพื่อสร้างซินแท็กซ์ทรีดังแสดงในรูปที่ 4.4 แสดงแผนภาพคลาสของแพ็กเกจ

คอมไพเลอร์ ผู้วิจัยได้เลือกเครื่องมือช่วยสร้างตัวแปลภาษาที่ชื่อว่า จาวา คอมไพเลอร์ คอมไพ-เลอร์ (Java Complier Complier - JavaCC) โดยผลที่ได้จากการประมวลผลด้วยเครื่องมือนี้จะได้ชุดของคลาสต่างๆ ดังนี้

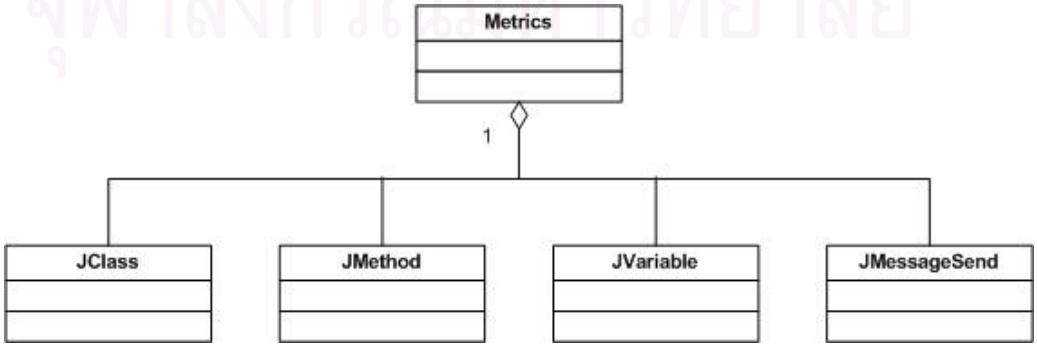
คลาส JavaParser จะเป็นคลาสหลักที่จะนำโปรแกรมต้นฉบับมาสร้างซินแท็กซ์ทรี โดยจะมีอินเตอร์เฟสคลาส JavaParserConstants ที่มีการประกาศค่าคงที่ต่าง ๆ และโทเคน (Token) ต่างๆ (โทเคน คือ กลุ่มอักขระตามไวยากรณ์ของภาษาจาวา) และมีคลาส Token คลาส JavaCharStream และคลาส JavaParserTokenManager ที่มีหน้าที่อ่านอักขระของโปรแกรมต้นฉบับและแบ่งให้เป็นโทเคน คลาสจาวาพาร์เซอร์ภายในจะประกอบไปด้วยตัวสร้างโหนดต่างๆ



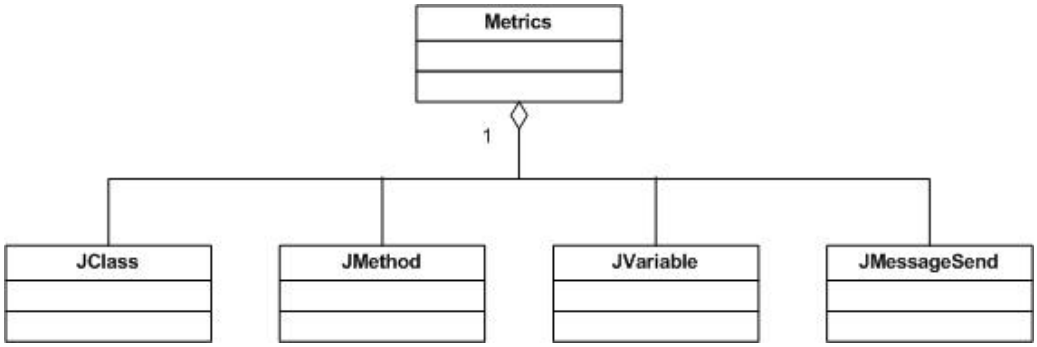
รูปที่ 4.4 แผนภาพคลาสของแพ็กเกจคอมไพเลอร์

4.2.3 แพ็กเกจคำนวณค่าตัววัด

แพ็กเกจคำนวณค่าตัววัด เป็นชุดของคลาสหน่วยรู้จำ โดยมีคลาสตัววัด (Metrics) ทำหน้าที่ท่องไปบนซินแท็กซ์ทรี เพื่อเก็บข้อมูลจากโหนดต่างๆ และคำนวณค่าตัววัดต่างๆ ซึ่งแพ็กเกจนี้ได้แบ่งออกเป็นหน่วยย่อยๆ คือ คลาส JClass คลาส JMethod คลาส JVariable และคลาส JMessage ซึ่งจะถูกเก็บอยู่ในรูปแบบของตัวแปรแบบเวคเตอร์ โดยมีคลาสที่สืบทอดคุณสมบัติมาจากคลาสเวคเตอร์ (java.util.Vector)



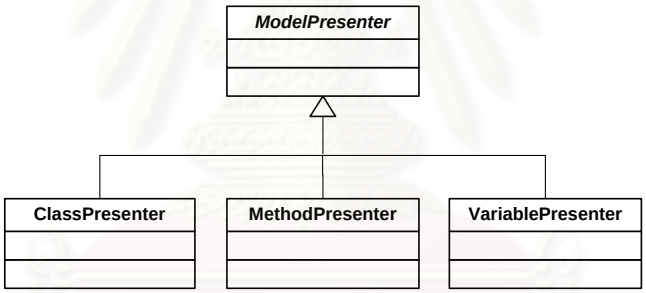
รูปที่ 4.5 แสดงให้เห็นถึงความสัมพันธ์ของคลาสต่างๆ ในชุดแพ็กเกจนี้



รูปที่ 4.5 แผนภาพคลาสของแพ็กเกจคำนวณค่าตัววัด

4.2.4 แพ็กเกจจัดรูปแบบการแสดงผลค่าตัววัด

แพ็กเกจจัดรูปแบบการแสดงผลค่าตัววัด เป็นชุดของคลาสที่ทำหน้าที่นำค่าตัววัดจากแพ็กเกจคำนวณค่าตัววัด มาจัดรูปแบบตามค่าตัววัดที่ต้องการ เพื่อส่งต่อให้แพ็กเกจส่วนติดต่อผู้ใช้ นำไปแสดงผลค่าวัดตามที่ได้จัดรูปแบบไว้ โดยได้แยกรูปแบบการแสดงผลตัววัดตามประเภทของหน่วยรู้จำ ได้แก่ คลาส ClassPresenter คลาส MethodPresenter และคลาส VariablePresenter รูปที่ 4.6 แสดงให้เห็นถึงความสัมพันธ์ของคลาสต่างๆ ในชุดแพ็กเกจนี้



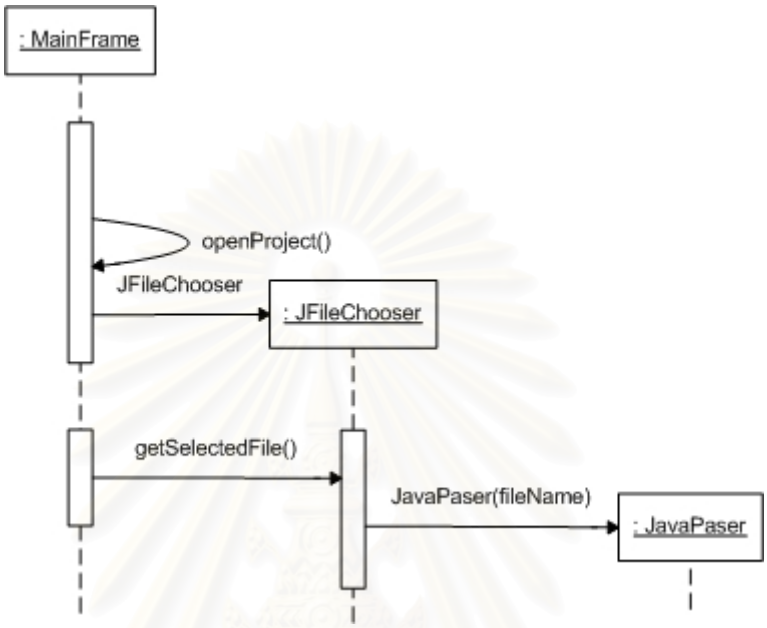
รูปที่ 4.6 แผนภาพคลาสของแพ็กเกจจัดรูปแบบการแสดงผลค่าตัววัด

4.3 แผนภาพซีเควนซ์

แผนภาพซีเควนซ์ของเครื่องมือช่วยในการตรวจจับร่องรอยที่ไม่ดีมี 10 แผนภาพ คือ แผนภาพซีเควนซ์ของกำหนดโปรแกรมต้นฉบับ แผนภาพซีเควนซ์ของการสร้างพาร์เซอร์ แผนภาพซีเควนซ์ของการคำนวณค่ามาตรวัด แผนภาพซีเควนซ์ของการเรียกดูค่ามาตรวัดร่องรอยที่ไม่ดีและวิธีการรีแฟคทอริงของ Feature Envy, Large Class, Lazy Class, Long Method, Long Parameter Lists และ Switch Statement และแผนภาพซีเควนซ์ของการเรียกดูค่ามาตรวัดร่องรอยที่ไม่ดีทั้งหมด ซึ่งมีรายละเอียดดังนี้

4.3.1 แผนภาพซีเควนซ์ของกำหนดโปรแกรมต้นฉบับ

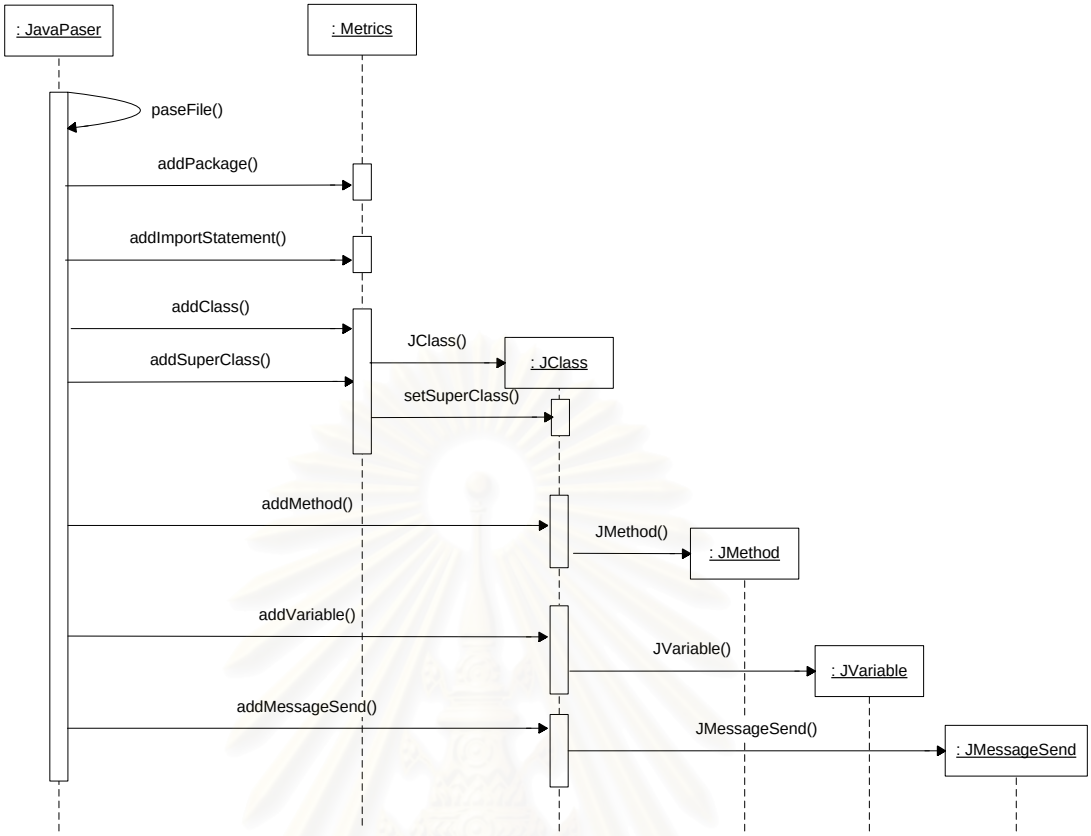
แผนภาพซีเควนซ์ของการกำหนดโปรแกรมต้นฉบับ ดังรูปที่ 4.7 เริ่มจากเครื่องมือสร้างเฟรม เพื่อให้ผู้ใช้ทำการเลือกโปรแกรมต้นฉบับ แล้วนำโปรแกรมต้นฉบับที่ผู้ใช้เลือกมาทำการสร้างพาร์เซอร์ของภาษาจาวา เพื่อเก็บข้อมูลในโปรแกรม



รูปที่ 4.7 แผนภาพซีเควนซ์ของการกำหนดโปรแกรมต้นฉบับ

4.3.2 แผนภาพซีเควนซ์ของการสร้างพาร์เซอร์

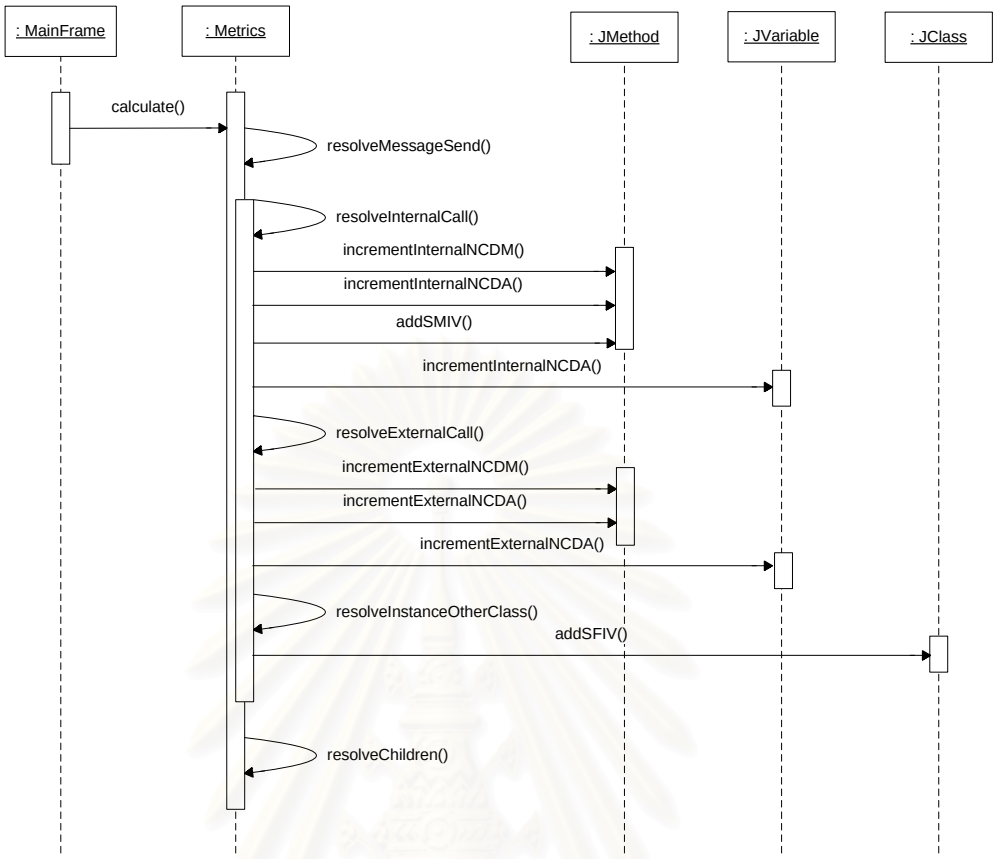
แผนภาพซีเควนซ์ของการสร้างพาร์เซอร์ของภาษาจาวา ดังรูปที่ 4.8 เริ่มต้นจากอ่านโปรแกรมต้นฉบับมาสร้างซินแท็กซ์ทรี และเก็บข้อมูลที่สำคัญคือ แพกเกจ คลาส เมทธอด คุณลักษณะ และข้อความที่ส่งภายในเมทธอด เพื่อนำไปคำนวณค่ามาตรวัดต่อไป



รูปที่ 4.8 แผนภาพซีเควนซ์ของการสร้างพาร์เซอร์

4.3.3 แผนภาพซีเควนซ์ของการคำนวณค่ามาตรวัด

แผนภาพซีเควนซ์ของการคำนวณค่ามาตรวัด ดังรูปที่ 4.9 ซึ่งคำนวณเฉพาะมาตรวัดร่องรอยที่ไม่ดีที่ต้องคำนวณจากการส่งข้อความภายในเมทธอด คือ มาตรวัด *NDCM*, *NCDA*, *NCRA*, *SMIV* และ *SFIV* เริ่มจากพิจารณาข้อความที่ส่งแต่ละข้อความ และตรวจสอบว่าเป็นการเรียกใช้คุณสมบัติภายในคลาส หรือการเรียกใช้ภายนอกคลาส ถ้าเป็นการเรียกใช้ภายในให้เพิ่มค่ามาตรวัด *NDCM*, *NCDA*, *NCRA* และ *SMIV* สำหรับคลาสเจ้าของเมทธอดหรือคุณลักษณะนั้น แต่ถ้าเป็นการเรียกใช้ภายนอกให้เพิ่มค่า มาตรวัด *NDCM*, *NCDA*, *NCRA* และ *SFIV* สำหรับคลาสอื่นๆ ที่มีการเรียกใช้ ส่วนมาตรวัดอื่นๆ จะเป็นค่ามาตรวัดที่สามารถหาได้จากคุณสมบัติของคลาส เมทธอด และคุณลักษณะที่ได้มีการเก็บข้อมูลไว้แล้ว ตั้งแต่การสร้างพาร์เซอร์ เช่น จำนวนเมทธอดของคลาส เป็นต้น



รูปที่ 4.9 แผนภาพซีเควนซ์ของการคำนวณค่ามาตรวัด

4.3.4 แผนภาพซีเควนซ์ของการเรียกดูมาตรวัดร่องรอยที่ไม่ดีและวิธีการรีแฟคทอริงของ Feature Envy

แผนภาพซีเควนซ์ของการเรียกดูมาตรวัดร่องรอยที่ไม่ดี และวิธีการรีแฟคทอริงของ Feature Envy ดังรูปที่ 4.10 เริ่มจากสร้างเฟรมเพื่อแสดงผลค่ามาตรวัด และดึงค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับ Feature Envy คือ มาตรวัด *NCDA*, *NCDA* และ *NCRA* หลังจากนั้นตรวจสอบว่าเมทธอดหรือคุณลักษณะใดบ้างที่มีค่ามาตรวัดอยู่ในข้อกำหนดที่พิจารณา และแสดงผลวิธีรีแฟคทอริงที่ใช้แก้ไขเมทธอดหรือคุณลักษณะที่พบว่าเป็น Feature Envy

4.3.5 แผนภาพซีเควนซ์ของการเรียกดูมาตรวัดร่องรอยที่ไม่ดีและวิธีการรีแฟคทอริงของ Large Class

แผนภาพซีเควนซ์ของการเรียกดูมาตรวัดร่องรอยที่ไม่ดี และวิธีการรีแฟคทอริงของ Large Class ดังรูปที่ 4.11 เริ่มจากสร้างเฟรมเพื่อแสดงผลค่ามาตรวัด และดึงค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับ Large Class คือ มาตรวัด *NIM*, *NIV* และ *TCC* หลังจากนั้นตรวจสอบว่าคลาสใดบ้างที่มี

ค่ามาตรวัดอยู่ในข้อกำหนดที่พิจารณา และตรวจสอบเงื่อนไขเพื่อแสดงผลวิธีรีแฟคทอริงที่ใช้แก้ไขคลาสที่พบว่าเป็น Large Class

4.3.6 แผนภาพซีเควนซ์ของการเรียกดูมาตรวัดร่องรอยที่ไม่ดีและวิธีการรีแฟคทอริงของ Lazy Class

แผนภาพซีเควนซ์ของการเรียกดูมาตรวัดร่องรอยที่ไม่ดี และวิธีการรีแฟคทอริงของ Lazy Class ดังรูปที่ 4.12 เริ่มจากสร้างเฟรมเพื่อแสดงผลค่ามาตรวัด และดึงค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับ Lazy Class คือ มาตรวัด *NIM*, *NIV* และ *DIT* หลังจากนั้นตรวจสอบว่าคลาสใดบ้างที่มีค่ามาตรวัดอยู่ในข้อกำหนดที่พิจารณา และตรวจสอบเงื่อนไขเพื่อแสดงผลวิธีรีแฟคทอริงที่ใช้แก้ไขคลาสที่พบว่าเป็น Lazy Class

4.3.7 แผนภาพซีเควนซ์ของการเรียกดูมาตรวัดร่องรอยที่ไม่ดีและวิธีการรีแฟคทอริงของ Long Method

แผนภาพซีเควนซ์ของการเรียกดูมาตรวัดร่องรอยที่ไม่ดี และวิธีการรีแฟคทอริงของ Long Method ดังรูปที่ 4.13 เริ่มจากสร้างเฟรมเพื่อแสดงผลค่ามาตรวัด และดึงค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับ Long Method คือ มาตรวัด *NOS*, *NOP* และ *NOT* หลังจากนั้นตรวจสอบว่าคลาสใดบ้างที่มีค่ามาตรวัดอยู่ในข้อกำหนดที่พิจารณา และตรวจสอบเงื่อนไขเพื่อแสดงผลวิธีรีแฟคทอริงที่ใช้แก้ไขเมทธอดที่พบว่าเป็น Long Method

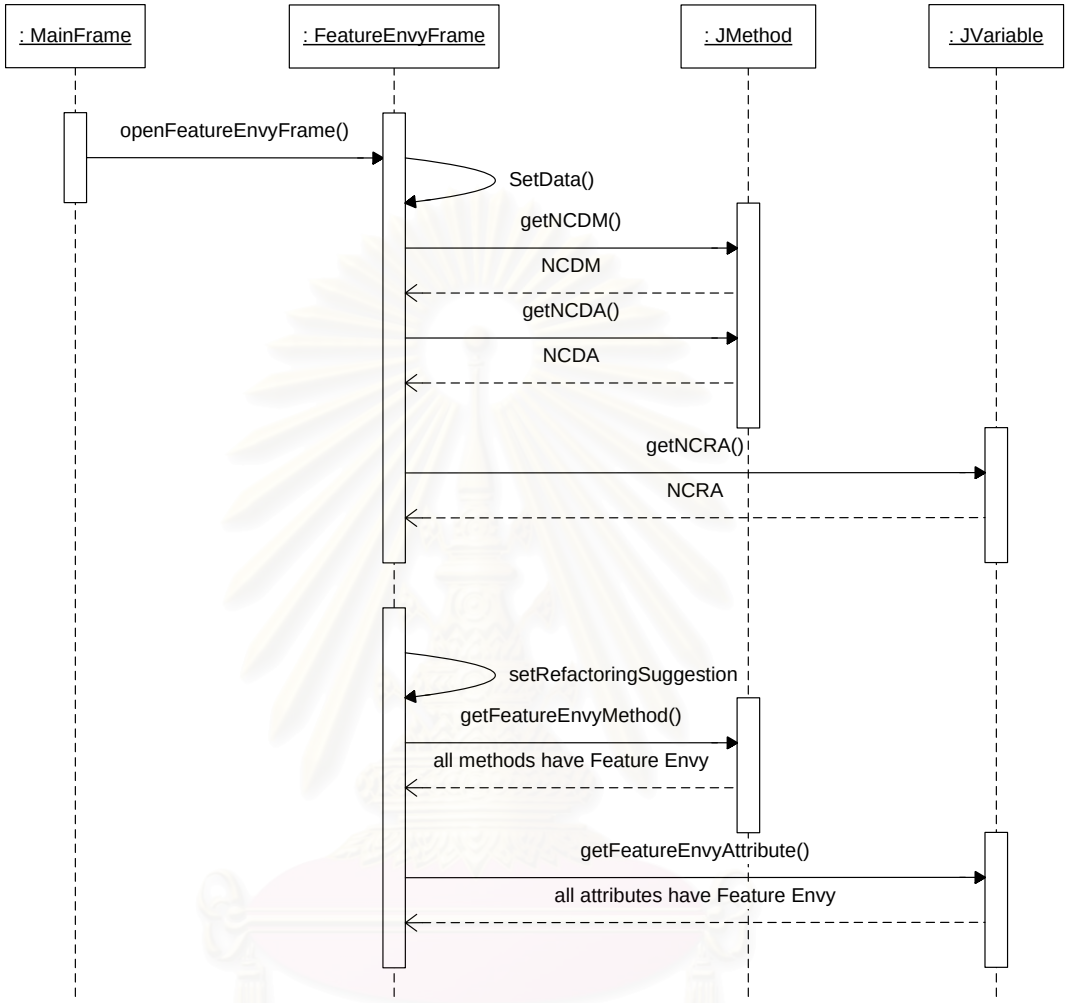
4.3.8 แผนภาพซีเควนซ์ของการเรียกดูมาตรวัดร่องรอยที่ไม่ดีและวิธีการรีแฟคทอริงของ Long Parameter Lists

แผนภาพซีเควนซ์ของการเรียกดูมาตรวัดร่องรอยที่ไม่ดี และวิธีการรีแฟคทอริงของ Long Parameter Lists ดังรูปที่ 4.14 เริ่มจากสร้างเฟรมเพื่อแสดงผลค่ามาตรวัด และดึงค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับ Long Parameter Lists คือ มาตรวัด *NOP* หลังจากนั้นตรวจสอบว่าคลาสใดบ้างที่มีค่ามาตรวัดอยู่ในข้อกำหนดที่พิจารณา และตรวจสอบเงื่อนไขเพื่อแสดงผลวิธีรีแฟคทอริงที่ใช้แก้ไขเมทธอดที่พบว่าเป็น Long Parameter Lists

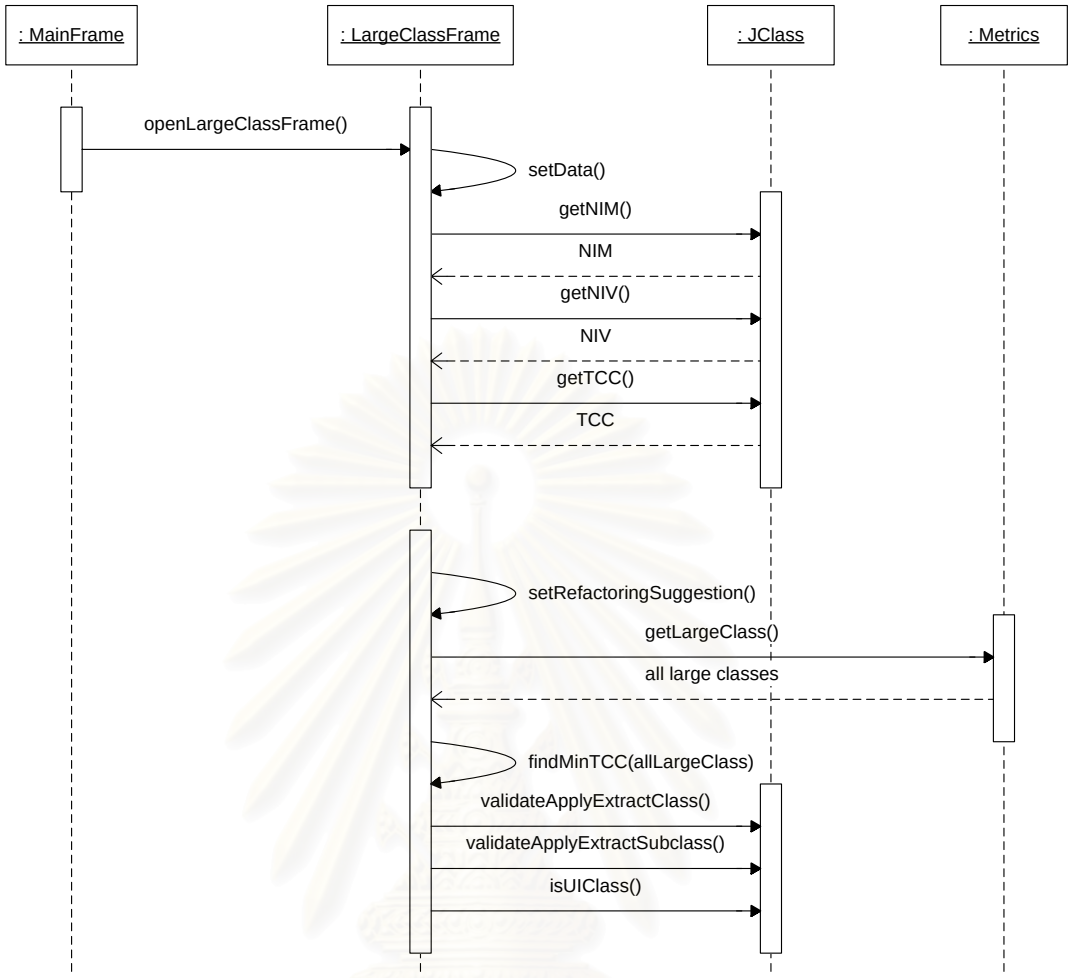
4.3.9 แผนภาพซีเควนซ์ของการเรียกดูมาตรวัดร่องรอยที่ไม่ดีและวิธีการรีแฟคทอริงของ Switch Statement

แผนภาพซีเควนซ์ของการเรียกดูมาตรวัดร่องรอยที่ไม่ดี และวิธีการรีแฟคทอริงของ Switch Statement ดังรูปที่ 4.15 เริ่มจากสร้างเฟรมเพื่อแสดงผลค่ามาตรวัด และดึงค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับ Switch Statement คือ มาตรวัด *NOS* หลังจากนั้นตรวจสอบว่าคลาสใดบ้างที่มีค่า

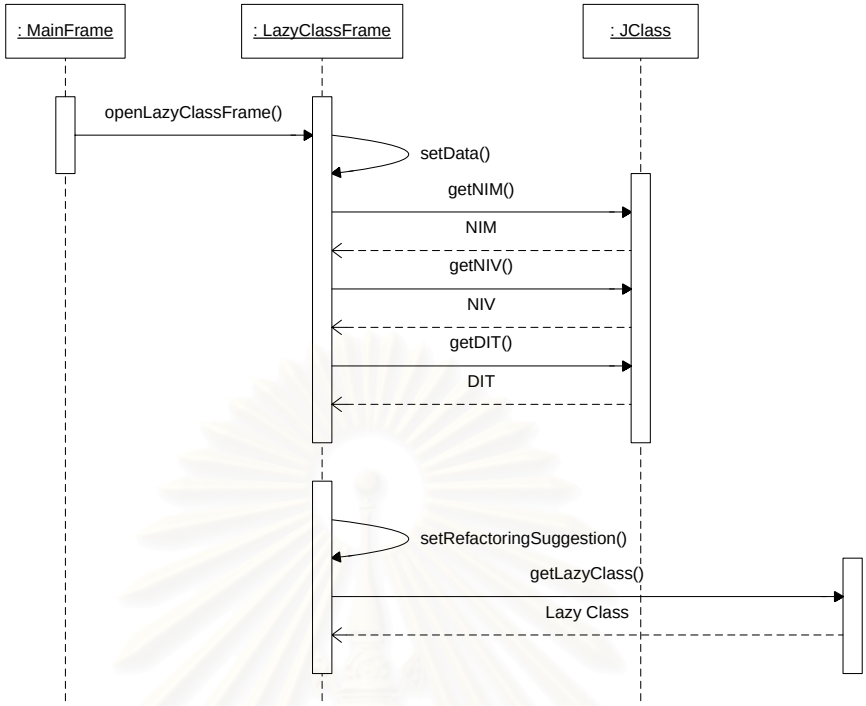
มาตรวัดอยู่ในข้อกำหนดที่พิจารณา และตรวจสอบเงื่อนไขเพื่อแสดงผลวิธีรีแฟคทอริงที่ใช้แก้ไขเมทธอดที่พบว่าเป็น Switch Statement



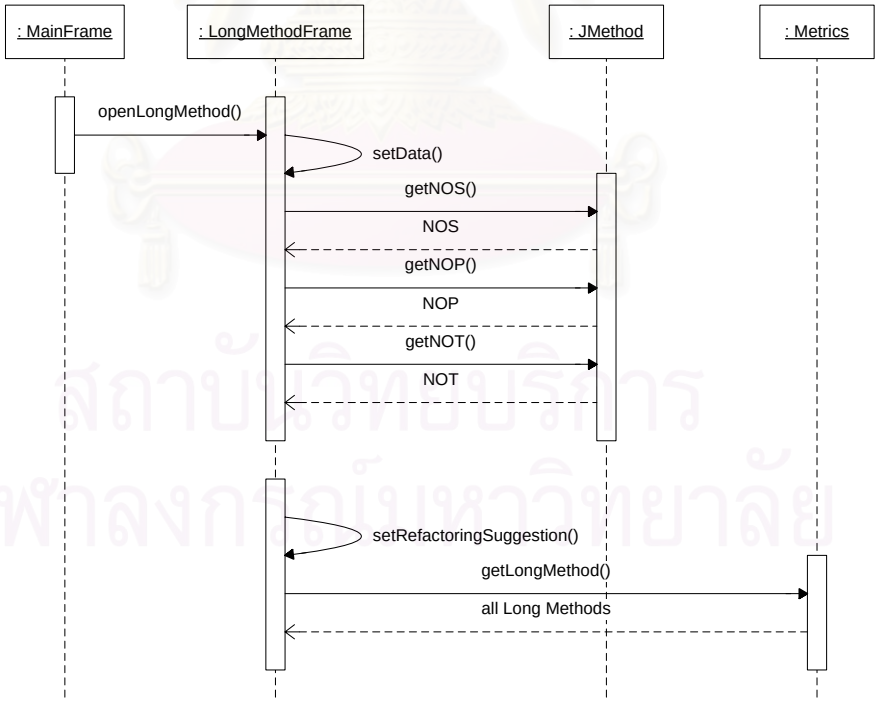
รูปที่ 4.10 แผนภาพซีเควนซ์ของการเรียกดูค่ามาตรวัด และวิธีรีแฟคทอริงของ Feature Envy



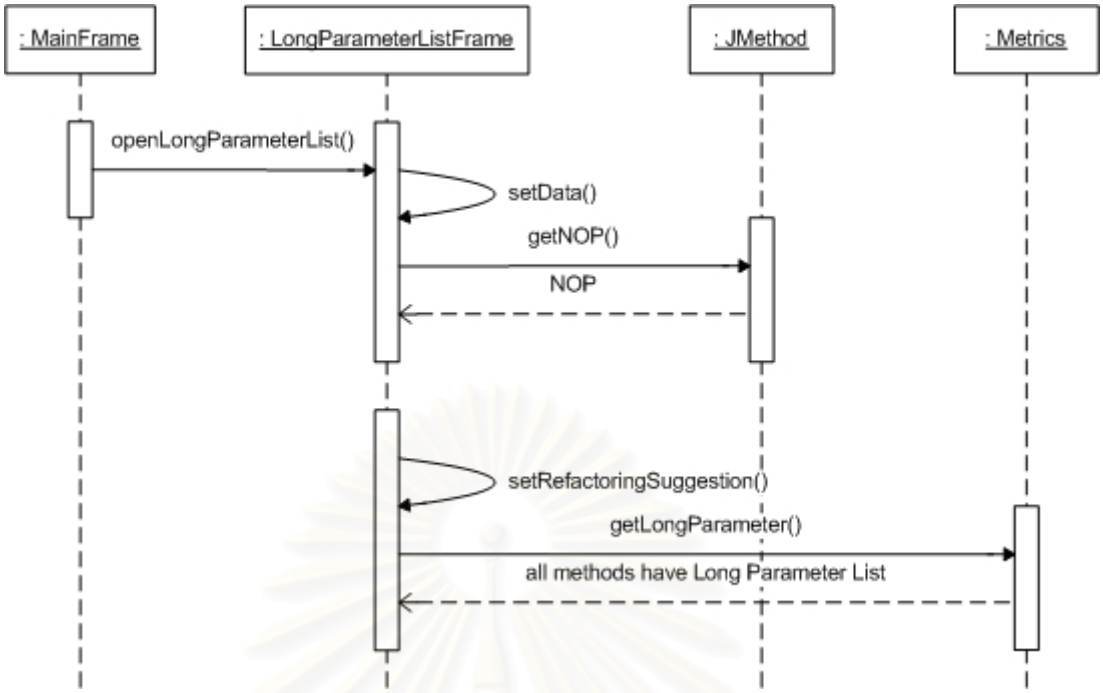
รูปที่ 4.11 แผนภาพซีเควนซ์ของการเรียกดูค่ามาตรวัด และวิธีรีแฟคทอริงของ Large Class



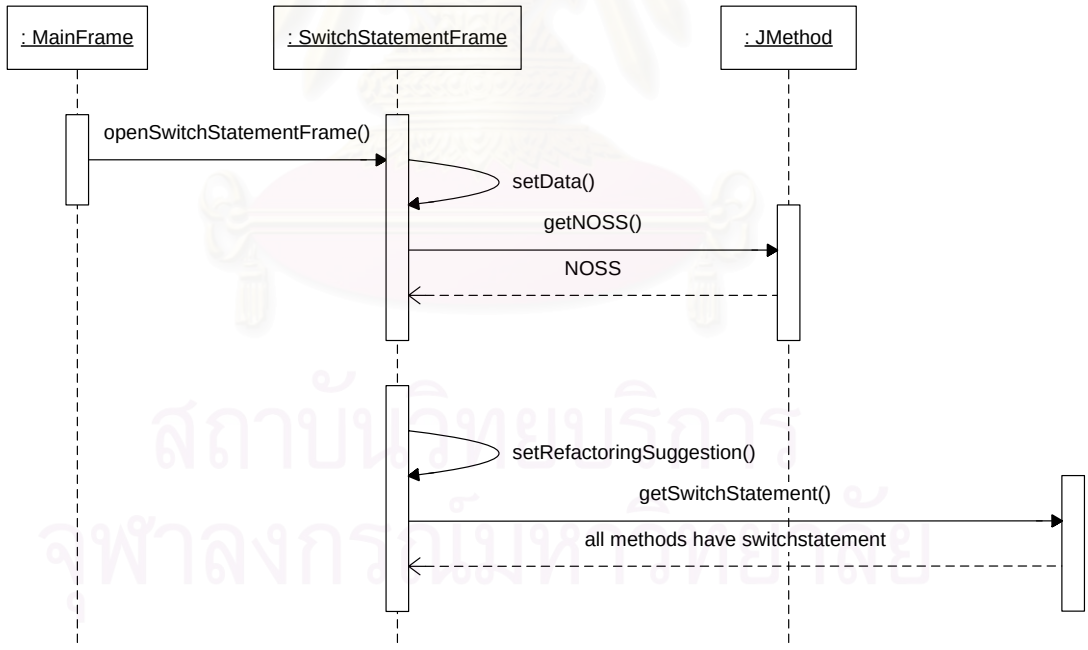
รูปที่ 4.12 แผนภาพซีเควนซ์ของการเรียกดูค่ามาตรวัด และวิธีรีแฟคทอริงของ Lazy Class



รูปที่ 4.13 แผนภาพซีเควนซ์ของการเรียกดูค่ามาตรวัด และวิธีรีแฟคทอริงของ Long Method



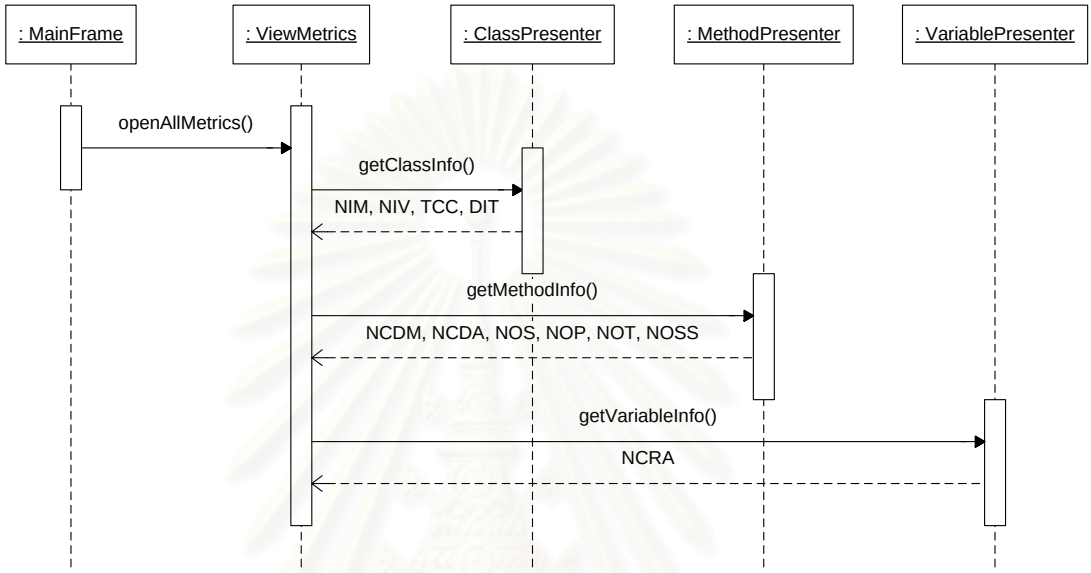
รูปที่ 4.14 แผนภาพซีเควนซ์ของการเรียกดูค่ามาตรวัด
และวิธีรีแฟคทอริงของ Long Parameter Lists



รูปที่ 4.15 แผนภาพซีเควนซ์ของการเรียกดูค่ามาตรวัด และวิธีรีแฟคทอริงของ Switch Statement

4.3.10 **แผนภาพซีเควนซ์ของการเรียกดูค่ามาตรวัดร่องรอยที่ไม่ดีทั้งหมด**

แผนภาพซีเควนซ์ของการเรียกดูค่ามาตรวัดร่องรอยที่ไม่ดี ดังรูปที่ 4.16 เริ่มจากสร้างเฟรมเพื่อแสดงผลค่ามาตรวัด จากนั้นดึงค่ามาตรวัดสำหรับคลาส เมทธอด และคุณลักษณะทั้งหมดมาแสดงผล



รูปที่ 4.16 แผนภาพซีเควนซ์ของการเรียกดูค่ามาตรวัดสำหรับคลาส เมทธอด และคุณลักษณะของโปรแกรม

บทที่ 5

การทดสอบความน่าเชื่อถือของมาตรวัดร่องรอยที่ไม่ดี

ผู้วิจัยได้ทำการทดสอบความน่าเชื่อถือของมาตรวัดร่องรอยที่ไม่ดี โดยนำโปรแกรมต้นฉบับมาทดสอบ มาทำการตรวจจับร่องรอยที่ไม่ดีโดยใช้เครื่องมือที่ช่วยในการตรวจจับร่องรอยที่ไม่ดีที่พัฒนาขึ้น จากนั้นจะเลือกวิธีการรีแฟคทอริงที่เหมาะสมตามวิธีที่เครื่องมือนำเสนอ เพื่อแก้ไขร่องรอยที่ไม่ดีที่พบให้ดีขึ้น แบ่งเป็น 2 ประเภท คือ การประยุกต์ใช้วิธีรีแฟคทอริงเพียง 1 วิธี เพื่อแก้ไขร่องรอยที่ไม่ดีแต่ละประเภท และการประยุกต์ใช้วิธีรีแฟคทอริงหลายๆ วิธี เพื่อแก้ไขร่องรอยที่ไม่ดีทุกประเภท หลังจากได้โปรแกรมต้นฉบับที่ทำรีแฟคทอริงทั้ง 2 ประเภทแล้ว นำมาคำนวณค่ามาตรวัดร่องรอยที่ไม่ดีอีกครั้ง เพื่อตรวจสอบว่าไม่พบร่องรอยที่ไม่ดีที่ได้แก้ไขแล้ว จากนั้นจะเปรียบเทียบผลการแก้ไขร่องรอยที่ไม่ดีแต่ละประเภทกับการแก้ไขร่องรอยที่ไม่ดีทุกประเภท เพื่อเสนอแนะวิธีการเลือกรีแฟคทอริงที่เหมาะสม และตรวจสอบค่ามาตรวัดที่ได้จากเครื่องมือ ซึ่งแบ่งเป็น 4 ขั้นตอน คือ การคำนวณค่ามาตรวัดร่องรอยที่ไม่ดี การประเมินความสามารถของมาตรวัดร่องรอยที่ไม่ดี การเลือกวิธีรีแฟคทอริง และการตรวจสอบค่ามาตรวัดที่ได้จากเครื่องมือ ดังรายละเอียดต่อไปนี้

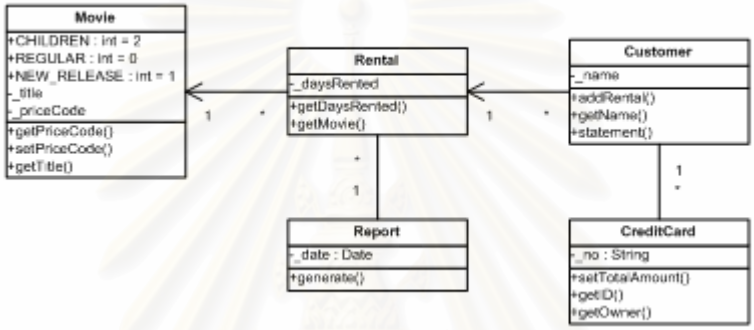
5.1 การคำนวณค่ามาตรวัดร่องรอยที่ไม่ดี

ในขั้นตอนนี้ ใช้เครื่องมือที่พัฒนาขึ้น มาช่วยคำนวณค่ามาตรวัดที่ออกแบบไว้เพื่อตรวจจับร่องรอยที่ไม่ดีจากโปรแกรมต้นฉบับที่มีขนาดต่างๆ กัน จำนวน 3 โปรแกรม ซึ่งเครื่องมือจะตรวจสอบว่า ผลของค่ามาตรวัดร่องรอยที่ไม่ดีอยู่ในข้อกำหนดของร่อยรอยที่ไม่ดีประเภทใด เพื่อจะเสนอแนะวิธีรีแฟคทอริงสำหรับการแก้ไขร่องรอยที่ไม่ดีที่ได้กำหนดไว้แล้วในขั้นตอนที่ 3.2 จากนั้นผู้วิจัยจะเลือกวิธีการรีแฟคทอริงตามความเหมาะสมกับโปรแกรมต้นฉบับที่นำมาทดสอบ และนำโปรแกรมต้นฉบับไปทำรีแฟคทอริงตามวิธีการที่เลือก

5.1.1 ผลการคำนวณของโปรแกรมที่ 1

โปรแกรมที่ใช้ในการทดสอบโปรแกรมที่ 1 ได้มาจากตัวอย่างโปรแกรมในหนังสือ Refactoring: Improving the design of existing code เขียนโดย Martin Fowler [5] เป็นโปรแกรมของระบบเช่าภาพยนตร์ มี 5 คลาส ขนาด 97 LOC ซึ่งแผนภาพคลาสแสดงรายละเอียดในรูปที่ 5.1 และซอร์สโค้ดแสดงรายละเอียดในรูปที่ 5.2 เมื่อนำโปรแกรมนี้ มาทำการตรวจจับร่องรอยที่ไม่ดีด้วยเครื่องมือที่พัฒนาขึ้น จะได้ค่ามาตรวัดแยกตามประเภทมาตรวัด 3 ประเภท คือ

มาตรวัดสำหรับคลาส เมทธอด และคุณลักษณะ ดังตารางที่ 5.1 5.2 และ 5.3 ตามลำดับ ยกตัวอย่างเช่น ในตารางที่ 5.1 คลาส Movie มีค่ามาตรวัด *NIM* = 4, *NIV* =5, *TCC*=0.33 และ *DIT*=0 เป็นต้น ส่วนในตารางมาตรวัดสำหรับเมทธอด และคุณลักษณะ ช่องที่แรเงาสีเทาของค่าผลรวมของมาตรวัด *NCDM* กับ *NCDA* และค่ามาตรวัด *NCRA* แสดงถึงค่ามาตรวัดของการเรียกใช้คุณสมบัติภายในคลาสที่กำหนดเมทธอดหรือคุณลักษณะนั้น เพื่อเปรียบเทียบกับการเรียกใช้ภายในคลาสอื่น ยกตัวอย่างเช่น เมทธอด statement ภายในคลาส Customer มีค่าผลรวมของมาตรวัด *NCDM* กับ *NCDA* ภายในคลาส Customer เท่ากับ 2 ซึ่งเป็นคลาสที่กำหนดเมทธอด statement เอง แต่ภายในคลาส Movie และคลาส Rental เท่ากับ 4 และ 9 ตามลำดับ



รูปที่ 5.1 แผนภาพคลาสของโปรแกรมที่ 1

```
class Movie
{
 public static final int CHILDREN = 2;
 public static final int REGULAR = 0;
 public static final int NEW_RELEASE = 1;
 private String _title;
 private int _priceCode;
public Movie(String title, int priceCode){
  _title = title;
  _priceCode = priceCode;
}
public int getPriceCode(){
  return _priceCode;
}
public void setPriceCode(int arg){
  _priceCode = arg;
}
public String getTitle(){
  return _title;
}
}
class CreditCard
{
  private String _id;
  private String _owner;
  private int _totalAmount = 0;
public CreditCard(String id, String name){
        _id = id;
        _owner = name;
}
public void setTotalAmount(int amount){
        _totalAmount += amount;
}
public String getID(){ return _id;}
public String getOwner(){return _owner;}
}
```

```
class Rental
{
  private Movie _movie;
  private int _daysRented;
  public Rental(Movie movie, int
  daysRented){
    _movie = movie;
    _daysRented = daysRented;
  }
  public int getDaysRented(){
    return _daysRented;
  }
  public Movie getMovie(){
    return _movie;
  }
}
```

```
class Report
{
  private Date _date;
  public Report(Date today){
    _date = today;
  }
public String generate(Customer arg){
  String result = "Report : "+
                  _date.toString();
  result += arg.statement()+"\n";
  return result;
  }
}
```

รูปที่ 5.2 ซอร์สโค้ดของโปรแกรมที่ 1 ก่อนการทำรีแฟคทอริง

```
class Customer
{
    private String _name;
    private Vector _rental = new Vector();
    public Customer(String name){ _name = name; }
    public void addRental(Rental arg){ _rental.addElement(arg);}
    public String getName(){ return _name; }
    public String statement(){
    double totalAmount = 0;
    int frequentRenterPoints = 0;
    Enumeration rentals = _rental.elements();
    String result = "Rental Record for "+ getName() +"\n";
    while(rentals.hasMoreElements()){
        double thisAmount = 0;
        Rental each = (Rental) rentals.nextElement();
        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.CHILDREN:
                    thisAmount += 1.5;
                    if(each.getDaysRented()>3)
                    thisAmount += (each.getDaysRented()-3)*1.5;
                    break;
        }
        frequentRenterPoints++;
        if((each.getMovie().getPriceCode()==Movie.NEW_RELEASE)&&each.getDaysRented()>1)
        frequentRenterPoints++;
        result += "\t"+each.getMovie().getTitle()+"\t"+String.valueOf(thisAmount)+"\n";
        totalAmount += thisAmount;
    }
result += "Amount owed is "+String.valueOf(totalAmount)+"\n";
result +="You earned"+String.valueOf(frequentRenterPoints)+"frequent renter points";
    return result;
    }
}
```

รูปที่ 5.2 (ต่อ) ซอร์สโค้ดของโปรแกรมที่ 1 ก่อนการทำรีแฟคทอริง

ตารางที่ 5.1 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 1

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
Movie	4	2	0.33	0
Rental	3	2	0	0
Customer	5	3	0.17	0
CreditCard	4	3	0	0
Report	**2**	**1**	**0**	**0**

ตารางที่ 5.2 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 1

เมทธอด		มาตรวัด								
		NCDM + NCDA					*NOS*	*NOP*	*NOT*	*NOSS*
		Movie	Rental	Customer	CreditCard	Report				
คลาส Movie	getPriceCode()	1	0	0	0	0	1	0	0	0
	setPriceCode()	1	0	0	0	0	1	1	0	0
	getTitle()	1	0	0	0	0	1	0	0	0
คลาส Rental	getDaysRented()	0	1	0	0	0	1	0	0	0
	getMovie()	0	1	0	0	0	1	0	0	0
คลาส Customer	**addRental()**	0	0	1	0	0	1	1	0	0
	addCreditCard()	0	0	1	0	0	1	1	0	0
	getName()	0	0	1	0	0	1	0	0	0
	statement()	4	9	2	0	0	23	0	6	1
คลาส CreditCard	**setTotalAmount()**	0	0	0	1	0	1	1	0	0
	getID()	0	0	0	1	0	1	0	0	0
	getOwner()	0	0	0	1	0	1	0	0	0
คลาส Report	**generate()**	0	0	0	0	1	2	1	1	0

ตารางที่ 5.3 ค่ามาตรวัดสำหรับคุณลักษณะของโปรแกรมที่ 1

คุณลักษณะ		มาตรวัด *NCRA*				
		Movie	Rental	Customer	CreditCard	Report
คลาส Movie	CHILDREN	0	0	1	0	0
	REGULAR	0	0	1	0	0
	NEW_RELEASE	0	0	2	0	0

(หมายเหตุ : แสดงผลเฉพาะคุณลักษณะที่มีขอบเขตเป็นพับบลิค)

จากผลของค่ามาตรวัดดังตารางที่ 5.1, 5.2 และ 5.3 พบว่าค่ามาตรวัดสำหรับเมทธอดและคุณลักษณะอยู่ในช่วงของข้อกำหนดที่พิจารณา เป็นร่องรอยที่ไม่ดี 6 ประเภท คือ

1. Switch Statement ได้แก่เมทธอด statement ของคลาส Customer เพราะพบว่าค่ามาตรวัด *NOSS* = 1 ซึ่งอยู่ในข้อกำหนดของค่าที่พิจารณาของ Switch Statement
2. Long Method ได้แก่เมทธอด statement ของคลาส Customer เพราะพบว่าค่ามาตรวัด *NOS* = 23 ซึ่งอยู่ในข้อกำหนดของค่าที่พิจารณาของ Long Method
3. Feature Envy ได้แก่เมทธอด statement ของคลาส Customer เพราะพบว่าค่ามาตรวัด *NCDM(Customer) + NCDA(Customer) < NCDM(Movie) + NCDA(Movie)* และ *NCDM(Customer) + NCDA(Customer) < NCDM(Rental) + NCDA(Rental)* ซึ่งอยู่ในข้อกำหนดของค่าที่พิจารณาของ Feature Envy สำหรับเมทธอด
4. Feature Envy ได้แก่คุณลักษณะ CHILDREN, REGULAR และ NEW_RELEASE ของคลาส Movie เพราะพบว่าค่ามาตรวัด *NCRA*(*Movie*) < *NCRA(Customer)* ซึ่งอยู่ในข้อกำหนดของค่าที่พิจารณาของ Feature Envy สำหรับคุณลักษณะ
5. Long Parameter Lists ได้แก่เมทธอด setPriceCode ของคลาส Movie เมทธอด addRental และเมทธอด addCreditCard ของคลาส Customer และเมทธอด generate ของคลาส Report เพราะพบว่าค่ามาตรวัด *NOP* = 1 ซึ่งมีค่ามากที่สุด ภายในโปรแกรม และอยู่ในข้อกำหนดของค่าที่พิจารณาของ Long Parameter Lists
6. Lazy Class ได้แก่ คลาส Report เพราะพบว่าผลรวมของค่ามาตรวัด *NIM* และ *NIV* มีค่าน้อยที่สุด ภายในโปรแกรม และอยู่ในข้อกำหนดของค่าที่พิจารณาของ Lazy Class

สำหรับวิธีการรีแฟคทอริงที่เสนอเพื่อแก้ไขร่องรอยที่ไม่ดีที่พบ แสดงรายละเอียดดังตารางที่ 5.4

ตารางที่ 5.4 ประเภทร่องรอยที่ไม่ดีที่พบ และวิธีรีแฟคทอริงที่เสนอของโปรแกรมที่ 1

ประเภท	ร่องรอยที่ไม่ดี	วิธีรีแฟคทอริง
Switch Statement	เมทธอด statement ของคลาส Customer	Extract Method Replace Conditional with Polymorphism Replace Parameter with Explicit Methods Introduce null object
Long Method	เมทธอด statement ของคลาส Customer	Replace Temp with Query Decompose Conditional
Feature Envy	เมทธอด statement ของคลาส Customer	Extract Method
	CHILDREN, REGULAR และ NEW_RELEASE ของคลาส Movie	Move Attribute
Long Parameter Lists	เมทธอด setPriceCode ของคลาส Movie	Replace Parameters with Method Preserve Whole Object Introduce Parameter Object
	เมทธอด addRental ของคลาส Customer	Replace Parameters with Method Preserve Whole Object Introduce Parameter Object
	เมทธอด addCreditCard ของคลาส Customer	Replace Parameters with Method Preserve Whole Object Introduce Parameter Object
	เมทธอด generate ของคลาส Report	Replace Parameters with Method Preserve Whole Object Introduce Parameter Object
Lazy Class	คลาส Report	Inline Class

ดังนั้นผู้วิจัยจึงได้เลือกวิธีการรีแฟคทอริงให้เลือกประยุกต์ใช้กับโปรแกรมต้นฉบับ เพื่อปรับปรุงร่องรอยที่ไม่ดี 4 ประเภทที่พบ (ส่วนตัวอย่างการแก้ไขร่องรอยที่ไม่ดีประเภท Long Parameter Lists และ Lazy Class อยู่ในการทดสอบโปรแกรมที่ 2 และ 3 ตามลำดับ) โดยกรณีที่ 1 สำหรับปรับปรุงคุณลักษณะ CHILDREN, REGULAR และ NEW_RELEASE และกรณีที่ 2, 3 และ 4 สำหรับปรับปรุงเมทธอด statement ซึ่งมีรายละเอียดดังนี้

กรณีที่ 1 วิธี Move Attribute โดยการย้ายคุณลักษณะ CHILDREN, REGULAR และ NEW_RELEASE จากคลาส Movie ไปยังคลาส Customer ดังแสดงในรูปที่ 5.3

```
class Customer
{
      …
      public static final int CHILDRENS = 2;
      public static final int REGULAR = 0;
      public static final int NEW_RELEASE = 1;
      …
      public String statement(){
              double totalAmount = 0;
              int frequentRenterPoints = 0;
              Enumeration rentals = _rental.elements();
              String result = "Rental Record for "+ getName() +"\n";
              while(rentals.hasMoreElements()){
                      double thisAmount = 0;
                      Rental each = (Rental) rentals.nextElement();
                      thisAmount = amountFor(each);
                      frequentRenterPoints++;

      if((each.getMovie().getPriceCode()==NEW_RELEASE)&&each.getDaysRented()>1)
              frequentRenterPoints++;
      result += "\t"+each.getMovie().getTitle()+"\t"+String.valueOf(thisAmount)+"\n";
      totalAmount += thisAmount;
              }
      result += "Amount owed is "+String.valueOf(totalAmount)+"\n";
      result += "You earned "+String.valueOf(frequentRenterPoints)+" frequent renter
points";
              return result;
      }
      private double amountFor(Rental each){
              double thisAmount = 0;
              switch (each.getMovie().getPriceCode())
              {
                      case REGULAR:
                              thisAmount +=2;
                              if (each.getDaysRented()>2)
                              thisAmount += (each.getDaysRented()-2)*1.5;
                              break;
                      case NEW_RELEASE:
                              thisAmount += each.getDaysRented()*3;
                              break;
                      case CHILDRENS:
                              thisAmount += 1.5;
                              if(each.getDaysRented()>3)
                              thisAmount += (each.getDaysRented()-3)*1.5;
                              break;
              }
              return thisAmount;
      }
}
```

รูปที่ 5.3 ซอร์สโค้ดของโปรแกรมที่ 1 หลังการประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 1

กรณีที่ 2 วิธี Extract Method โดยการย้ายสวิทช์สเตทเมนต์ในเมทธอด statement มาสร้างเป็นเมทธอดใหม่คือ เมทธอด amountFor ดังแสดงในรูปที่ 5.4

กรณีที่ 3 วิธี Extract Method โดยการแยกส่วนของเมทธอด statement ตามการเรียกใช้คุณสมบัติของคลาส Movie และ Rental มาสร้างเป็นเมทธอดใหม่ จากนั้นใช้วิธี Move Method เพื่อย้ายเมทธอดเหล่านั้น ไปยังคลาสต่างๆ ตามที่มีการเรียกใช้งาน จากที่มีการสร้างเมทธอด amountFor ในกรณีที่ 2 แล้ว จึงทำการย้ายเมทธอด amountFor จากคลาส Customer ไปยังคลาส Rental ดังแสดงในรูปที่ 5.5

```
class Customer
{
…
public String statement(){
double totalAmount = 0;
int frequentRenterPoints = 0;
Enumeration rentals = _rental.elements();
String result = "Rental Record for "+ getName() +"\n";
while(rentals.hasMoreElements()){
      double thisAmount = 0;
      Rental each = (Rental) rentals.nextElement();
      thisAmount = amountFor(each);
      frequentRenterPoints++;
      if((each.getMovie().getPriceCode()==NEW_RELEASE)&&each.getDaysRented()>1)
      frequentRenterPoints++;
      result += "\t"+each.getMovie().getTitle()+"\t"+String.valueOf(thisAmount)+"\n";
      totalAmount += thisAmount;
}
result += "Amount owed is "+String.valueOf(totalAmount)+"\n";
result  +=  "You  earned  "+String.valueOf(frequentRenterPoints)+"  frequent  renter
points";
return result;
}
private double amountFor(Rental each){
      double thisAmount = 0;
      switch (each.getMovie().getPriceCode())      {
      case REGULAR:
            thisAmount +=2;
            if (each.getDaysRented()>2)
            thisAmount += (each.getDaysRented()-2)*1.5;
            break;
      case NEW_RELEASE:
            thisAmount += each.getDaysRented()*3;
            break;
      case CHILDRENS:
            thisAmount += 1.5;
            if(each.getDaysRented()>3)
            thisAmount += (each.getDaysRented()-3)*1.5;
            break;
      }
      return thisAmount;
}
}
```

รูปที่ 5.4 ซอร์สโค้ดของโปรแกรมที่ 1 หลังการประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 2

```
class Rental
{
      private Movie _movie;
      private int _daysRented;

      public Rental(Movie movie, int daysRented){
            _movie = movie;
            _daysRented = daysRented;
      }
      public int getDaysRented(){
            return _daysRented;
      }
      public Movie getMovie(){
            return _movie;
      }
```

รูปที่ 5.5 ซอร์สโค้ดของโปรแกรมที่ 1 หลังการประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 3

```
                public double amountFor(){
                double thisAmount = 0;
                      switch (getMovie().getPriceCode()){
                      case Customer.REGULAR:
                      thisAmount +=2;
                      if (getDaysRented()>2)
                      thisAmount += (getDaysRented()-2)*1.5;
                      break;
                      case Customer.NEW_RELEASE:
                           thisAmount += getDaysRented()*3;
                      break;
                      case Customer.CHILDRENS:
                        thisAmount += 1.5;
                      if(getDaysRented()>3)
                      thisAmount += (getDaysRented()-3)*1.5;
                      break;
                }
                return thisAmount;

  }
}
```

รูปที่ 5.5 (ต่อ) ซอร์สโค้ดของโปรแกรมที่ 1 หลังการประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 3

กรณีที่ 4 วิธี Replace Temp with Query ควรจะประยุกต์ใช้กับเมทธอด statement โดยการแทนที่ตัวแปรชั่วคราวด้วยการเรียกใช้เมทธอดแทน เพราะค่ามาตรวัด *NOT* มีค่ามากที่สุด โดยการแทนที่การเรียกใช้ตัวแปร thisAmount ด้วยการเรียกใช้เมทธอด amountFor เพื่อคำนวณค่าตัวแปร ภายในเมทธอด statement ของคลาส Customer ดังแสดงในรูปที่ 5.6

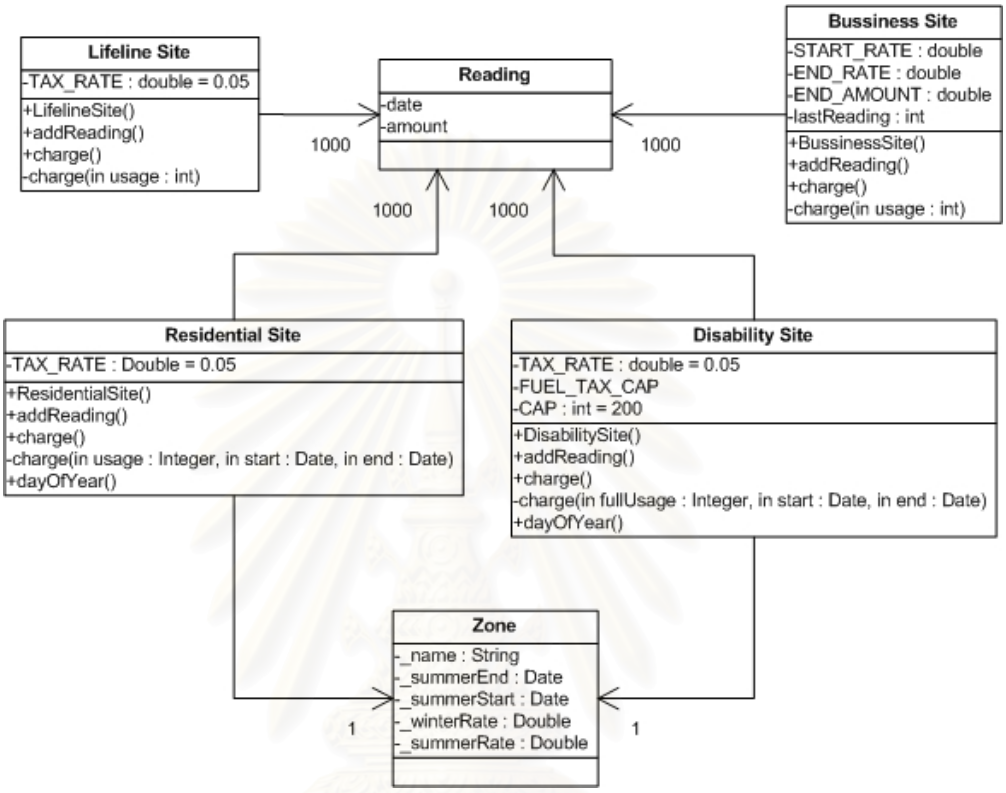
```
   class Customer
   {
        …
        public String statement(){
                double totalAmount = 0;
                int frequentRenterPoints = 0;
                Enumeration rentals = _rental.elements();
                String result = "Rental Record for "+ getName() +"\n";
                while(rentals.hasMoreElements()){
                        Rental each = (Rental) rentals.nextElement();
                        frequentRenterPoints++;
        if((each.getMovie().getPriceCode()==NEW_RELEASE)&&each.getDaysRented()>1)
                        frequentRenterPoints++;
   result+=
   "\t"+each.getMovie().getTitle()+"\t"+String.valueOf(each.amountFor())+"\n";
                        totalAmount += each.amountFor();
                }
   result += "Amount owed is "+String.valueOf(totalAmount)+"\n";
   result  +=  "You  earned  "+String.valueOf(frequentRenterPoints)+"  frequent  renter
   points";
   return result;
   }
   }
```

รูปที่ 5.6 ซอร์สโค้ดของโปรแกรมที่ 1 หลังการประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 4

5.1.2 ผลการคำนวณของโปรแกรมที่ 2

โปรแกรมต้นฉบับที่ใช้ในการทดสอบโปรแกรมที่ 2 ได้มาจากตัวอย่างโปรแกรมในหนังสือ Refactoring: Improving the design of existing code เขียนโดย Martin Fowler [5] มี 6 คลาส ขนาด 257 LOC เป็นโปรแกรมสำหรับระบบคิดค่าไฟฟ้าของผู้ใช้งาน ซึ่งแสดงแผนภาพคลาสในรูปที่ 5.7 และรายละเอียดซอร์สโค้ดในภาคผนวก ค เมื่อนำโปรแกรมต้นฉบับนี้ มาทำการคำนวณค่า

มาตรวัดร่องรอยที่ไม่ดีด้วยเครื่องมือที่พัฒนาขึ้น จะได้ค่ามาตรวัดแยกตามประเภทมาตรวัด 3 ประเภท คือ มาตรวัดสำหรับคลาส และเมทธอด ดังตารางที่ 5.5 และ 5.6 ตามลำดับ ส่วนมาตรวัดสำหรับคุณลักษณะ ไม่มีผลค่ามาตรวัด เนื่องจากโปรแกรมนี้ไม่มีคุณลักษณะที่มีขอบเขตเป็นพับ-บลิค



รูปที่ 5.7 แผนภาพคลาสของโปรแกรมที่ 2

จากตารางที่ 5.5 พบว่าค่ามาตรวัดสำหรับคลาสอยู่ในช่วงของข้อกำหนดที่พิจารณาของ Lazy Class คือ คลาส LifelineSite

ตารางที่ 5.5 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
BusinessSite	4	3	0.334	0
DisabilitySite	5	2	0.167	0
LifelineSite	**3**	**1**	**0.334**	**0**
Reading	3	2	0	0
ResidentialSite	5	2	0.167	0
Zone	6	5	0	0

ตารางที่ 5.6 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2

เมทธอด		มาตรวัด									
		NCDM + NCDA						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
BusinessSite	addReading(Reading newReading)	2	0	0	0	0	0	1	1	0	0
	charge()	5	0	0	0	0	0	1	0	1	0
	charge(int usage)	12	0	0	0	0	0	**10**	1	5	0
DisabilitySite	addReading(Reading newReading)	0	2	0	0	0	0	2	1	1	0
	charge()	0	6	0	0	0	0	3	0	4	0
	charge(int fullUsage, Date start, Date end)	0	22	0	0	0	0	**14**	**3**	5	0
	dayOfYear(Date arg)	0	0	0	0	0	0	**43**	1	1	**1**
LifelineSite	addReading(Reading newReading)	0	0	4	0	0	0	3	1	1	0
	charge()	0	0	3	0	0	0	1	0	1	0
	charge (int usage)	0	0	2	0	0	0	7	1	4	0

ตารางที่ 5.6 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2

เมทธอด		มาตรวัด									
		NCDM + NCDA						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
Reading	date()	0	0	0	1	0	0	1	0	0	0
	amount()	0	0	0	1	0	0	1	0	0	0
ResidentialSite	addReading(Reading newReading)	0	0	0	0	2	0	3	1	1	0
	charge()	0	0	0	0	6	0	4	0	4	0
	charge(int usage, Date start, Date end)	0	0	0	0	20	0	**13**	**3**	4	0
	dayOfYear(Date arg)	0	0	0	0	0	0	**43**	1	1	**1**
Zone	persist()	0	0	0	0	0	0	2	0	0	0
	get (String name)	0	0	0	0	0	0	1	1	0	0
	summerEnd()	0	0	0	0	0	1	1	0	0	0
	summerStart()	0	0	0	0	0	1	1	0	0	0
	winterRate()	0	0	0	0	0	1	1	0	0	0
	summerRate()	0	0	0	0	0	1	1	0	0	0

ตารางที่ 5.7 ประเภทร่องรอยที่ไม่ดีที่พบ และวิธีรีแฟคทอริงที่เสนอของโปรแกรมที่ 2

ประเภท	ร่องรอยที่ไม่ดี	วิธีรีแฟคทอริง
Long Method	เมทธอด charge ในคลาส BusinessSite	Extract Method Replace Method with Method Object Introduce Parameter Object Preserve Whole Object Replace Temp with Query
	เมทธอด charge ในคลาส DisabilitySite	Extract Method Replace Method with Method Object Introduce Parameter Object Preserve Whole Object Replace Temp with Query
	เมทธอด dayOfYear ในคลาส DisabilitySite	Extract Method Replace Method with Method Object Introduce Parameter Object Preserve Whole Object Replace Temp with Query
	เมทธอด charge ในคลาส ResidentialSite	Extract Method Replace Method with Method Object Introduce Parameter Object Preserve Whole Object Replace Temp with Query

ตารางที่ 5.7 (ต่อ) ประเภทร่องรอยที่ไม่ดีที่พบในโปรแกรมที่ 2 วิธีรีแฟคทอริงที่เสนอ

ประเภท	ร่องรอยที่ไม่ดี	วิธีรีแฟคทอริง
Long Method	เมทธอด dayOfYear ในคลาส ResidentialSite	Extract Method Replace Method with Method Object Introduce Parameter Object Preserve Whole Object Replace Temp with Query
Long Parameter List	เมทธอด charge ในคลาส DisabilitySite	Replace Parameters with Method Preserve Whole Object Introduce Parameter Object
	เมทธอด charge ในคลาส ResidentialSite	Replace Parameters with Method Preserve Whole Object Introduce Parameter Object
Switch Statement	เมทธอด dayOfYear ภายในคลาส DisabilitySite	Extract Method Replace Conditional with Polymorphism Replace Parameter with Explicit Methods Introduce null object
	เมทธอด dayOfYear ภายในคลาส ResidentialSite	Extract Method Replace Conditional with Polymorphism Replace Parameter with Explicit Methods Introduce null object
Lazy Class	คลาส LifelineSite	Inline Class

จากผลของค่ามาตรวัดดังตารางที่ 5.6 พบว่าค่ามาตรวัดสำหรับเมทธอดอยู่ในช่วงของข้อกำหนดที่พิจารณา คือ เมทธอด charge ในคลาส BusinessSite เมทธอด charge และ dayOf Year ในคลาส DisabilitySite เมทธอด charge และ dayOfYear ในคลาส ResidentialSite ดังนั้น โปรแกรมที่ 2 พบร่องรอยที่ไม่ดี 3 ประเภท และได้เสนอวิธีรีแฟคทอริงสำหรับแก้ไขร่อยรอยที่ไม่ดีที่พบ รายละเอียดแสดงดังตารางที่ 5.7

จากนั้นผู้วิจัยเลือกวิธีการรีแฟคทอริงจากที่เครื่องมือนำเสนอ สำหรับปรับปรุงโปรแกรมที่ 2 แบ่งเป็น 11 กรณี มีรายละเอียดดังนี้

กรณีที่ 1 วิธี Replace Temp with Query ควรจะประยุกต์ใช้กับเมทธอด charge ในคลาส DisabilitySite โดยการแทนที่ตัวแปร fuel ด้วยการสร้างเมทธอด fuelCharge ขึ้นมาเพื่อให้เรียกใช้งานแทน

กรณีที่ 2 วิธี Extract Method ควรจะประยุกต์ใช้กับเมทธอด charge ในคลาส DisabilitySite โดยการย้ายบางส่วนของคลาสมาสร้างเมทธอดใหม่ คือ เมทธอด fuelCharge Taxes และเมทธอด summerFraction

กรณีที่ 3 วิธี Replace Temp with Query ควรจะประยุกต์ใช้กับเมทธอด charge ในคลาส ResidentialSite เช่นเดียวกับกรณีที่ 1

กรณีที่ 4 วิธี Extract Method ควรจะประยุกต์ใช้กับเมทธอด charge ในคลาส ResidentialSite เช่นเดียวกับกรณีที่ 2

กรณีที่ 5 วิธี Replace Parameters with Method ควรจะประยุกต์ใช้กับเมทธอด charge ในคลาส DisabilitySite โดยการแทนที่การใช้พารามิเตอร์ fullUsage ด้วยการสร้างเมทธอด lastUsage ขึ้นมาใหม่

กรณีที่ 6 วิธี Replace Parameters with Method ควรจะประยุกต์ใช้กับเมทธอด charge ในคลาส ResidentialSite เช่นเดียวกับกรณีที่ 5

กรณีที่ 7 วิธี Extract Method ควรจะประยุกต์ใช้กับเมทธอด charge ในคลาส BusinessSite โดยการย้ายโค้ดบางส่วนมาสร้างเป็นเมทธอดใหม่คือเมทธอด baseCharge

กรณีที่ 8 วิธี Extract Method ควรจะประยุกต์ใช้กับเมทธอด dayOfYear ในคลาส DisabilitySite โดยการย้ายโค้ดบางส่วนมาสร้างเป็นเมทธอดใหม่คือเมทธอด isLeapYear

กรณีที่ 9 วิธี Extract Method ควรจะประยุกต์ใช้กับเมทธอด dayOfYear ในคลาส ResidentialSite เช่นเดียวกับกรณีที่ 8

กรณีที่ 10 วิธี Extract Method ควรจะประยุกต์ใช้กับเมทธอด dayOfYear ภายในคลาส DisabilitySite โดยการสร้างเมทธอดใหม่แทนที่การทำงานของโค้ดเดิมคือเมทธอด daysToStart OfMonth

กรณีที่ 11 วิธี Extract Method ควรจะประยุกต์ใช้กับเมทธอด dayOfYear ภายในคลาส ResidentialSite เช่นเดียวกับกรณีที่ 10

5.1.3 ผลการคำนวณของโปรแกรมที่ 3

โปรแกรมที่ใช้ในการทดสอบโปรแกรมที่ 3 ได้มาจากตัวอย่างโปรแกรมในหนังสือ Java: How to program [15] เป็นโปรแกรมสำหรับจำลองการทำงานของลิฟต์ รายละเอียดแผนภาพคลาสอยู่ในภาคผนวก ค มีจำนวน 33 คลาส และขนาด 1947 LOC เมื่อนำโปรแกรมนี้ มาทำการตรวจจับร่องรอยที่ไม่ดีด้วยเครื่องมือที่พัฒนาขึ้น จะได้ค่ามาตรวัดแยกตามประเภทมาตรวัด 3 ประเภท คือ มาตรวัดสำหรับคลาส มาตรวัดสำหรับเมทธอด และมาตรวัดสำหรับคุณลักษณะ ดังแสดงรายละเอียดในภาคผนวก ง ซึ่งพบร่องรอยที่ไม่ดี 3 ประเภท คือ Large Class, Lazy class, Long Method, Long Parameter Lists และ Switch Statement แต่ผู้วิจัยเลือกวิธีการรีแฟคทอริงจากที่เครื่องมือนำเสนอ สำหรับปรับปรุงโปรแกรมที่ 3 โดยเลือกแก้ไขเฉพาะกรณีร่องรอยที่ไม่ดีที่ยังไม่พบในโปรแกรมที่ 1 และ 2 รายละเอียดแสดงดังตารางที่ 5.8 ซึ่งแยกเป็น 2 กรณี มีรายละเอียดดังนี้

กรณีที่ 1 วิธี Extract Class ควรจะประยุกต์ใช้กับคลาส ElevatorSimulation โดยการย้ายเมทธอด sendPersonMoveEvent, เมทธอด addPersonMoveListener, เมทธอด getFloor, คุณลักษณะ firstFloor, คุณลักษณะ secondFloor, คุณลักษณะ elevatorShaft และคุณลักษณะ personMoveListeners ออกจากคลาสเดิมมาสร้างเป็นคลาสใหม่ คือคลาส ElevatorSimulation Constants

กรณีที่ 2 วิธี Collapse Hierarchy ควรจะประยุกต์ใช้กับคลาส BellEvent, ButtonEvent, DoorEvent, ElevatorEvent และ LightEvent โดยการย้ายคุณสมบัติทั้งหมดภายในคลาสเหล่านี้ไปรวมกับคลาสแม่ คือคลาส ElevatorSimulationEvent และลบคลาสเดิมทิ้ง

ส่วนรายละเอียดของร่องรอยที่ไม่ดีประเภท Long Method, Long Parameter Lists และ Switch Statement ที่พบในโปรแกรมที่ 3 รายละเอียดแสดงในภาคผนวก ง ซึ่งวิธีการแก้ไขจะคล้ายกับในโปรแกรมที่ 1 และ 2 ดังกล่าวข้างต้น

ตารางที่ 5.8 ประเภทร่องรอยที่ไม่ดีที่พบ และวิธีรีแฟคทอริงที่เสนอของโปรแกรมที่ 3

ประเภท	ร่องรอยที่ไม่ดี	วิธีรีแฟคทอริง
Large Class	คลาส ElevatorSimulation	Extract class
Lazy Class	คลาส BellEvent	Collapse Hierarchy
	คลาส ButtonEvent	Collapse Hierarchy
	คลาส DoorEvent	Collapse Hierarchy
	คลาส ElevatorMoveEvent	Collapse Hierarchy
	คลาส LightEvent	Collapse Hierarchy

5.2 การประเมินความสามารถของมาตรวัดร่องรอยที่ไม่ดี

ในขั้นตอนนี้ จะประเมินความสามารถมาตรวัดร่องรอยที่ไม่ดี โดยประเมินจากคำนวณค่ามาตรวัดร่องรอยที่ไม่ดีอีกครั้ง เพื่อตรวจสอบว่าไม่พบร่องรอยที่ไม่ดีที่ถูกแก้ไขไปแล้ว โดยแบ่งเป็น 2 กรณี คือ การแก้ไขร่องรอยที่ไม่ดีแต่ละประเภท เป็นการนำซอร์ดโค้ดเดิมมาประยุกต์ใช้รีแฟคทอริง เพื่อแก้ไขร่องรอยที่ไม่ดีแต่ละประเภท และการแก้ไขร่องรอยที่ไม่ดีทุกประเภท เป็นการนำซอร์ดโค้ดที่ทำการแก้ไขร่องรอยที่ไม่ดีแล้วมาประยุกต์ใช้รีแฟคทอริง เพื่อแก้ไขร่องรอยที่ไม่ดีอื่นๆ ต่อไป ซึ่งมีรายละเอียดการทดสอบแต่ละโปรแกรมดังนี้

5.2.1 ผลการทดสอบโปรแกรมที่ 1

การแก้ไขร่องรอยที่ไม่ดีแต่ละประเภท

จากการแก้ไขร่องรอยที่ไม่ดีที่พบในโปรแกรมที่ 1 ได้ผลของค่ามาตรวัดร่องรอยที่ไม่ดีหลังการประยุกต์ใช้วิธีรีแฟคทอริงแต่ละกรณี 4 กรณี ดังนี้

ตารางที่ 5.9 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 1หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 1

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
Movie	4	2	0.33	0
Rental	3	2	0	0
Customer	5	3	0.17	0
CreditCard	4	3	0	0
Report	2	1	0	0

ตารางที่ 5.10 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 1หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 1

เมทธอด		มาตรวัด								
		NCDM + NCDA					*NOS*	*NOP*	*NOT*	*NOSS*
		Movie	Rental	Customer	CreditCard	Report				
คลาส Movie	getPriceCode()	1	0	0	0	0	1	0	0	0
	setPriceCode()	1	0	0	0	0	1	1	0	0
	getTitle()	1	0	0	0	0	1	0	0	0
คลาส Rental	getDaysRented()	0	1	0	0	0	1	0	0	0
	getMovie()	0	1	0	0	0	1	0	0	0
คลาส Customer	addRental()	0	0	1	0	0	1	1	0	0
	addCreditCard()	0	0	1	0	0	1	1	0	0
	getName()	0	0	1	0	0	1	0	0	0
	statement()	0	9	6	0	0	23	0	6	1
คลาส CreditCard	setTotalAmount()	0	0	0	1	0	1	1	0	0
	getID()	0	0	0	1	0	1	0	0	0
	getOwner()	0	0	0	1	0	1	0	0	0
คลาส Report	generate()	0	0	0	0	1	2	1	1	0

กรณีที่ 1 เป็นการประยุกต์ใช้วิธี Move Attribue เพื่อแก้ไขร่องรอยที่ไม่ดีประเภท Feature Envy จากตารางที่ 5.9 5.10 และ 5.11 แสดงให้เห็นถึงการเปลี่ยนแปลงของค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคลาส เมทธอด และคุณลักษณะหลังการทำรีแฟคทอริงตามลำดับ ซึ่งมีรายละเอียดดังนี้

1. ค่ามาตรวัดสำหรับคลาสไม่เปลี่ยนแปลง เมื่อเปรียบเทียบค่ามาตรวัดสำหรับคลาสก่อนการประยุกต์ใช้วิธี Move Attribue แสดงในตารางที่ 5.9
2. ค่ามาตรวัดสำหรับเมทธอด จากตารางที่ 5.10 แสดงให้เห็นว่าค่ามาตรวัด *NCDM*(*Customer*) + *NCDA*(*Customer*) = 6 ของเมทธอด statement ในคลาส Customer เพิ่มขึ้น เพราะเป็นการย้ายคุณลักษณะจากคลาส Movie มายังคลาส Customer แต่ไม่มีผลทำให้เกิดร่องรอยที่ไม่ดีประเภทอื่นเพิ่มขึ้น
3. ค่ามาตรวัดสำหรับคุณลักษณะ จากตารางที่ 5.11 แสดงให้เห็นว่าค่ามาตรวัด *NCRA* ไม่อยู่ในช่วงของค่าที่พิจารณาของ Feature Envy แล้ว

ตารางที่ 5.11 ค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคุณลักษณะของโปรแกรมที่ 1 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 1

คุณลักษณะ		มาตรวัด *NCRA*				
		Movie	Rental	Customer	CreditCard	Report
คลาส Customer	CHILDREN	0	0	1	0	0
	REGULAR	0	0	1	0	0
	NEW_RELEASE	0	0	2	0	0

กรณีที่ 2 เป็นการประยุกต์ใช้วิธี Extract Method เพื่อแก้ไขร่องรอยที่ไม่ดีประเภท Switch Statement ในเมทธอด statement จากตารางที่ 5.12 5.13 และ 5.14 แสดงให้เห็นถึงการเปลี่ยนแปลงของค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคลาส เมทธอด และคุณลักษณะหลังการทำรีแฟคทอริงตามลำดับ ซึ่งมีรายละเอียดดังนี้

1. ค่ามาตรวัดสำหรับคลาส จากตารางที่ 5.12 แสดงให้เห็นว่าค่ามาตรวัด *NIM* = 6 ของคลาส Customer เพิ่มขึ้นจากเดิม เพราะเป็นการสร้างเมทธอดใหม่ภายในคลาส Customer แต่ไม่มีผลทำให้เกิดร่องรอยที่ไม่ดีประเภทอื่นเพิ่มขึ้น

2. ค่ามาตรวัดสำหรับเมทธอด จากตารางที่ 5.14 ได้ค่ามาตรวัด *NOS* = 10 ซึ่งลดลงและ *NOSS* = 0 สำหรับเมทธอด statement ซึ่งไม่อยู่ในช่วงของค่าที่พิจารณาของ Switch Statement แล้ว แต่ทำให้เกิดร่องรอยที่ไม่ดีประเภท Feature Ennvy ที่เมทธอด amountFor เป็นเมทธอดที่สร้างขึ้นใหม่
3. ค่ามาตรวัดสำหรับคุณลักษณะไม่เปลี่ยนแปลง เมื่อเปรียบเทียบค่ามาตรวัดสำหรับคุณลักษณะก่อนการประยุกต์ใช้วิธี Extract Method แสดงในตารางที่ 5.13

ตารางที่ 5.12 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 1หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 2

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
Movie	4	2	0.33	0
Rental	3	2	0	0
Customer	**6**	3	0.1	0
CreditCard	4	3	0	0
Report	2	1	0	0

ตารางที่ 5.13 ค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคุณลักษณะของโปรแกรมที่ 1 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 2

คุณลักษณะ		มาตรวัด *NCRA*				
		Movie	Rental	Customer	CreditCard	Report
คลาส **Movie**	CHILDREN	0	0	1	0	0
	REGULAR	0	0	1	0	0
	NEW_RELEASE	0	0	2	0	0

ตารางที่ 5.14 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 1หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 2

เมทธอด		*NCDM + NCDA*					*NOS*	*NOP*	*NOT*	*NOSS*
		Movie	Rental	Customer	CreditCard	Report				
คลาส Movie	getPriceCode()	1	0	0	0	0	1	0	0	0
	setPriceCode()	1	0	0	0	0	1	1	0	0
	getTitle()	1	0	0	0	0	1	0	0	0
คลาส Rental	getDaysRented()	0	1	0	0	0	1	0	0	0
	getMovie()	0	1	0	0	0	1	0	0	0
คลาส Customer	addRental()	0	0	1	0	0	1	1	0	0
	addCreditCard()	0	0	1	0	0	1	1	0	0
	getName()	0	0	1	0	0	1	0	0	0
	statement()	1	3	3	0	0	10	0	6	0
	amountFor	3	0	0	0	0	15	1	1	1
คลาส CreditCard	setTotalAmount()	0	0	0	1	0	1	1	0	0
	getID()	0	0	0	1	0	1	0	0	0
	getOwner()	0	0	0	1	0	1	0	0	0
คลาส Report	generate()	0	0	0	0	1	2	1	1	0

กรณีที่ 3 เป็นการประยุกต์ใช้วิธี Move Method เพื่อแก้ไขร่องรอยที่ไม่ดีประเภท Feature Envy จากตารางที่ 5.15 5.16 และ 5.17 แสดงให้เห็นถึงการเปลี่ยนแปลงของค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคลาส เมทธอด และคุณลักษณะหลังการทำรีแฟคทอริงตามลำดับ ซึ่งมีรายละเอียดดังนี้

1. ค่ามาตรวัดสำหรับคลาส แสดงให้เห็นว่าค่ามาตรวัด *NIM* = 4 ของคลาส Rental เพิ่มขึ้นจากเดิม เพราะเป็นการย้ายเมทธอดจากคลาส Customer มายังคลาส Rental แต่ไม่มีผลทำให้เกิดร่องรอยที่ไม่ดีประเภทอื่นเพิ่มขึ้น
2. ค่ามาตรวัดสำหรับเมทธอด จากตารางที่ 5.17 แสดงให้เห็นว่าค่ามาตรวัด *NCDM* และ *NCDA* ของเมทธอด amountForไม่อยู่ในช่วงของค่าที่พิจารณาของ Feature Envy แล้ว
3. ค่ามาตรวัดสำหรับคุณลักษณะ จากตารางที่ 5.16 พบว่ามีผลทำให้ค่ามาตรวัด *NCRA*(Rental) = 1 ซึ่งมากกว่า *NCRA*(Customer) = 0 ทำให้เกิดร่องรอยที่ไม่ดีประเภท Feature Ennvy ของ CHILDREN และ REGULAR

ตารางที่ 5.15 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 1หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 3

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
Movie	4	2	0.33	0
Rental	**4**	2	0	0
Customer	5	3	0.17	0
CreditCard	4	3	0	0
Report	2	1	0	0

ตารางที่ 5.16 ค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคุณลักษณะของโปรแกรมที่ 1 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 3

คุณลักษณะ		มาตรวัด *NCRA*				
		Movie	Rental	Customer	CreditCard	Report
คลาส Customer	**CHILDREN**	0	**1**	**0**	0	0
	REGULAR	0	**1**	**0**	0	0
	NEW_RELEASE	0	1	1	0	0

ตารางที่ 5.17 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 1หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 3

เมทธอด		*NCDM + NCDA*					*NOS*	*NOP*	*NOT*	*NOSS*
		Movie	Rental	Customer	CreditCard	Report				
คลาส Movie	getPriceCode()	1	0	0	0	0	1	0	0	0
	setPriceCode()	1	0	0	0	0	1	1	0	0
	getTitle()	1	0	0	0	0	1	0	0	0
คลาส Rental	getDaysRented()	0	1	0	0	0	1	0	0	0
	getMovie()	0	1	0	0	0	1	0	0	0
	amountFor	0	6	3	0	0	15	1	1	1
คลาส Customer	addRental()	0	0	1	0	0	1	1	0	0
	addCreditCard()	0	0	1	0	0	1	1	0	0
	getName()	0	0	1	0	0	1	0	0	0
	statement()	0	3	3	0	0	10	0	6	0
คลาส CreditCard	setTotalAmount()	0	0	0	1	0	1	1	0	0
	getID()	0	0	0	1	0	1	0	0	0
	getOwner()	0	0	0	1	0	1	0	0	0
คลาส Report	generate()	0	0	0	0	1	2	1	1	0

กรณีที่ 4 เป็นการประยุกต์ใช้วิธี Replace Temp with Query เพื่อแก้ไขร่องรอยที่ไม่ดีประเภท Long Method จากตารางที่ 5.18 5.19 และ 5.20 แสดงให้เห็นถึงการเปลี่ยนแปลงของค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคลาส เมทธอด และคุณลักษณะหลังการทำรีแฟคทอริงตามลำดับ ซึ่งมีรายละเอียดดังนี้

1. ค่ามาตรวัดสำหรับคลาส แสดงให้เห็นว่าค่ามาตรวัด *NIM* = 6 ของคลาส Customer เพิ่มขึ้นจากเดิม เพราะเป็นการสร้างเมทธอดใหม่ขึ้นมา เพื่อเรียกใช้แทนตัวแปรชั่วคราว แต่ไม่มีผลทำให้เกิดร่องรอยที่ไม่ดีประเภทอื่นเพิ่มขึ้น
2. ค่ามาตรวัดสำหรับเมทธอด จากตารางที่ 5.20 เมทธอด statement ได้ค่ามาตรวัด *NOS* = 9 และ *NOT* = 5 ซึ่งค่ามาตรวัดลดลง เมื่อเปรียบเทียบกับก่อนทำรีแฟคทอริง
3. ค่ามาตรวัดสำหรับคุณลักษณะไม่เปลี่ยนแปลง เมื่อเปรียบเทียบค่ามาตรวัดสำหรับคุณลักษณะก่อนการประยุกต์ใช้วิธี Extract Method แสดงในตารางที่ 5.19

ตารางที่ 5.18 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 1หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 4

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
Movie	4	2	0.33	0
Rental	3	2	0	0
Customer	**6**	3	0.1	0
CreditCard	4	3	0	0
Report	2	1	0	0

ตารางที่ 5.19 ค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคุณลักษณะของโปรแกรมที่ 1 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 4

คุณลักษณะ		มาตรวัด *NCRA*				
		Movie	Rental	Customer	CreditCard	Report
คลาส **Movie**	CHILDREN	0	0	1	0	0
	REGULAR	0	0	1	0	0
	NEW_RELEASE	0	0	2	0	0

ตารางที่ 5.20 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 1หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 4

เมทธอด		*NCDM + NCDA*					*NOS*	*NOP*	*NOT*	*NOSS*
		Movie	Rental	Customer	CreditCard	Report				
คลาส Movie	getPriceCode()	1	0	0	0	0	1	0	0	0
	setPriceCode()	1	0	0	0	0	1	1	0	0
	getTitle()	1	0	0	0	0	1	0	0	0
คลาส Rental	getDaysRented()	0	1	0	0	0	1	0	0	0
	getMovie()	0	1	0	0	0	1	0	0	0
คลาส Customer	addRental()	0	0	1	0	0	1	1	0	0
	addCreditCard()	0	0	1	0	0	1	1	0	0
	getName()	0	0	1	0	0	1	0	0	0
	statement()	1	3	3	0	0	**9**	0	**5**	0
	amountFor	*3*	0	*0*	0	0	15	1	1	1
คลาส CreditCard	setTotalAmount()	0	0	0	1	0	1	1	0	0
	getID()	0	0	0	1	0	1	0	0	0
	getOwner()	0	0	0	1	0	1	0	0	0
คลาส Report	generate()	0	0	0	0	1	2	1	1	0

การแก้ไขร่องรอยที่ไม่ดีทุกประเภท

หลังจากการประยุกต์ใช้รีแฟคทอริงสำหรับโปรแกรมที่ 1 ทั้ง 4 กรณี ได้ค่า มาตรวัดร่องรอยที่ไม่ดีสำหรับคลาส เมทธอด และคุณลักษณะ แสดงรายละเอียดดังตารางที่ 5.21 5.22 และ 5.23 ซึ่งมีรายละเอียดดังนี้

1. ค่ามาตรวัดสำหรับคลาส จากตารางที่ 5.21 แสดงให้เห็นว่าค่ามาตรวัด *NIM* = 5 ของคลาส Rental เพิ่มขึ้นจากเดิม เพราะเป็นการสร้างเมทธอดใหม่ขึ้นมา และย้ายเมทธอดนั้นไปยังคลาส Rental แต่ไม่มีผลทำให้เกิดร่องรอยที่ไม่ดีประเภทอื่นเพิ่มขึ้น
2. ค่ามาตรวัดสำหรับเมทธอด จากตารางที่ 5.24
    1) ค่ามาตรวัด *NOS* = 9 และ *NOT* = 5 ของเมทธอด statement ซึ่งค่ามาตรวัดลดลง เมื่อเปรียบเทียบกับก่อนทำรีแฟคทอริง
    2) ค่ามาตรวัด *NCDM* และ *NCDA* ของเมทธอด amountFor และ statement ไม่อยู่ในช่วงของค่าที่พิจารณาของ Feature Envy แล้ว
    3) ค่ามาตรวัด *NOSS* = 0 สำหรับเมทธอด statement ซึ่งไม่อยู่ในช่วงของค่าที่พิจารณาของ Switch Statement แล้ว
3. ค่ามาตรวัดสำหรับคุณลักษณะ จากตารางที่ 5.23 แสดงให้เห็นว่าค่ามาตรวัด *NCRA* ไม่อยู่ในช่วงของค่าที่พิจารณาของ Feature Envy แล้ว

ตารางที่ 5.21 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 1 หลังประยุกต์ใช้รีแฟคทอริง ทั้ง 4 กรณี

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
Movie	4	2	0.33	0
Rental	**4**	2	0	0
Customer	5	3	0.17	0
CreditCard	4	3	0	0
Report	2	1	0	0

ตารางที่ 5.22 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 1หลังประยุกต์ใช้รีแฟคทอริงทั้ง 4 กรณี

เมทธอด		*NCDM + NCDA*					*NOS*	*NOP*	*NOT*	*NOSS*
		Movie	Rental	Customer	CreditCard	Report				
คลาส Movie	getPriceCode()	1	0	0	0	0	1	0	0	0
	setPriceCode()	1	0	0	0	0	1	1	0	0
	getTitle()	1	0	0	0	0	1	0	0	0
คลาส Rental	getDaysRented()	0	1	0	0	0	1	0	0	0
	getMovie()	0	1	0	0	0	1	0	0	0
	amountFor	**0**	**3**	**0**	**0**	**0**	**15**	**1**	**1**	**1**
คลาส Customer	addRental()	0	0	1	0	0	1	1	0	0
	addCreditCard()	0	0	1	0	0	1	1	0	0
	getName()	0	0	1	0	0	1	0	0	0
	statement()	**0**	**2**	**2**	**0**	**0**	**9**	**0**	**5**	**0**
คลาส CreditCard	setTotalAmount()	0	0	0	1	0	1	1	0	0
	getID()	0	0	0	1	0	1	0	0	0
	getOwner()	0	0	0	1	0	1	0	0	0
คลาส Report	generate()	0	0	0	0	1	2	1	1	0

ตารางที่ 5.23 ค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคุณลักษณะของโปรแกรมที่ 1 หลังประยุกต์ใช้รีแฟคทอริงทั้ง 4 กรณี

คุณลักษณะ		มาตรวัด *NCRA*				
		Movie	Rental	Customer	CreditCard	Report
คลาส Rental	CHILDREN	0	1	0	0	0
	REGULAR	0	1	0	0	0
	NEW_RELEASE	0	2	0	0	0

5.2.2 ผลการทดสอบโปรแกรมที่ 2

การแก้ไขร่องรอยที่ไม่ดีแต่ละประเภท

จากการแก้ไขร่องรอยที่ไม่ดีที่พบในโปรแกรมที่ 2 ได้ผลของค่ามาตรวัดร่องรอยที่ไม่ดีหลังการประยุกต์ใช้วิธีรีแฟคทอริงแต่ละกรณีทั้ง 11 กรณี ดังนี้

กรณีที่ 1 และ 2 เป็นการประยุกต์ใช้วิธี Replace Temp with Query และวิธี Extract Method เพื่อแก้ไขร่องรอยที่ไม่ดีประเภท Long Method สำหรับเมทธอด charge ในคลาส DisabilitySite จากตารางที่ 5.24 และ 5.25 แสดงให้เห็นถึงการเปลี่ยนแปลงของค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคลาส และเมทธอดหลังการทำรีแฟคทอริง ซึ่งมีรายละเอียดดังนี้

1. ค่ามาตรวัดสำหรับคลาส จากตารางที่ 5.25 แสดงให้เห็นว่าค่ามาตรวัด *NIM* = 8 ของคลาส DisabilitySite เพิ่มขึ้นจากเดิม เพราะเป็นการสร้างเมทธอดใหม่ขึ้นมา แต่ไม่มีผลทำให้เกิดร่องรอยที่ไม่ดีประเภทอื่นเพิ่มขึ้น
2. ค่ามาตรวัดสำหรับเมทธอด จากตารางที่ 5.24 ได้ค่ามาตรวัด *NOS* = 6 ซึ่งไม่อยู่ในช่วงของค่าที่พิจารณาของ Long Method แล้วและมาตรวัด *NOT* = 2 ซึ่งลดลงเมื่อเปรียบเทียบกับก่อนทำรีแฟคทอริง เพราะมีการลดการใช้ตัวแปรชั่วคราว

ตารางที่ 5.24 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 1 และ 2

เมทธอด		*NCDM + NCDA*						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
BusinessSite	addReading(Reading newReading)	2	0	0	0	0	0	1	1	0	0
	charge()	5	0	0	0	0	0	1	0	1	0
	charge(int usage)	12	0	0	0	0	0	10	1	5	0
DisabilitySite	addReading(Reading newReading)	0	2	0	0	0	0	2	1	1	0
	charge()	0	6	0	0	0	0	3	0	4	0
	charge(int fullUsage, Date start, Date end)	0	8	0	0	0	0	**6**	3	**2**	0
	fuelCharge()	0	1	0	0	0	0	1	1	0	0
	fuelChargeTaxes()	0	3	0	0	0	0	1	1	0	0
	summerFraction()	0	20	0	0	0	0	9	2	2	0
	dayOfYear(Date arg)	0	0	0	0	0	0	43	1	1	1
LifelineSite	addReading(Reading newReading)	0	0	4	0	0	0	3	1	1	0

ตารางที่ 5.24 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 1 และ 2

เมทธอด		*NCDM + NCDA*						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
	charge()	0	0	3	0	0	0	1	0	1	0
	charge (int usage)	0	0	2	0	0	0	7	1	4	0
Reading	date()	0	0	0	1	0	0	1	0	0	0
	amount()	0	0	0	1	0	0	1	0	0	0
Residential Site	addReading()	0	0	0	0	2	0	3	1	1	0
	charge()	0	0	0	0	6	0	4	0	4	0
	charge(int usage, Date start, Date end)	0	0	0	0	20	0	13	3	4	0
	dayOfYear(Date arg)	0	0	0	0	0	0	43	1	1	1
Zone	persist()	0	0	0	0	0	0	2	0	0	0
	get (String name)	0	0	0	0	0	0	1	1	0	0
	summerEnd()	0	0	0	0	0	1	1	0	0	0
	summerStart()	0	0	0	0	0	1	1	0	0	0
	winterRate()	0	0	0	0	0	1	1	0	0	0
	summerRate()	0	0	0	0	0	1	1	0	0	0

ตารางที่ 5.25 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริง ในกรณีที่ 1 และ 2

คลาส	*NIM*	*NIV*	*TCC*	*DIT*
BusinessSite	4	3	0.334	0
DisabilitySite	**8**	2	0.095	0
LifelineSite	3	1	0.334	0
Reading	3	2	0	0
ResidentialSite	5	2	0.167	0
Zone	6	5	0	0

กรณีที่ 3 และ 4 เป็นการประยุกต์ใช้วิธี Replace Temp with Query และวิธี Extract Method เพื่อแก้ไขร่องรอยที่ไม่ดีประเภท Long Method สำหรับเมทธอด charge ในคลาส ResidentialSite จากตารางที่ 5.26 และ 5.27 แสดงให้เห็นถึงการเปลี่ยนแปลงของค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคลาส และเมทธอดหลังการทำรีแฟคทอริง ซึ่งมีรายละเอียดดังนี้

1. ค่ามาตรวัดสำหรับคลาส จากตารางที่ 5.26 แสดงให้เห็นว่าค่ามาตรวัด *NIM* = 8 ของคลาส ResidentialSite เพิ่มขึ้นจากเดิม เพราะเป็นการสร้างเมทธอดใหม่ขึ้นมา แต่ไม่มีผลทำให้เกิดร่องรอยที่ไม่ดีประเภทอื่นเพิ่มขึ้น
2. ค่ามาตรวัดสำหรับเมทธอด จากตารางที่ 5.27 ได้ค่ามาตรวัด *NOS* = 5 ซึ่งไม่อยู่ในช่วงของค่าที่พิจารณาของ Long Method แล้วและมาตรวัด *NOT* = 1 ซึ่งลดลงเมื่อเปรียบเทียบกับก่อนทำรีแฟคทอริง เพราะลดการใช้ตัวแปรชั่วคราว

ตารางที่ 5.26 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริง ในกรณีที่ 3 และ 4

คลาส	*NIM*	*NIV*	*TCC*	*DIT*
BusinessSite	4	3	0.334	0
DisabilitySite	5	2	0.167	0
LifelineSite	3	1	0.334	0
Reading	3	2	0	0
ResidentialSite	**8**	2	0.095	0
Zone	6	5	0	0

ตารางที่ 5.27 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 3 และ 4

เมทธอด		*NCDM + NCDA*						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
BusinessSite	addReading(Reading newReading)	2	0	0	0	0	0	1	1	0	0
	charge()	5	0	0	0	0	0	1	0	1	0
	charge(int usage)	12	0	0	0	0	0	10	1	5	0
DisabilitySite	addReading(Reading newReading)	0	2	0	0	0	0	2	1	1	0
	charge()	0	6	0	0	0	0	3	0	4	0
	charge(int fullUsage, Date start, Date end)	0	22	0	0	0	0	14	3	5	0
	dayOfYear(Date arg)	0	0	0	0	0	0	43	1	1	1
LifelineSite	addReading(Reading newReading)	0	0	4	0	0	0	3	1	1	0
	charge()	0	0	3	0	0	0	1	0	1	0
	charge (int usage)	0	0	2	0	0	0	7	1	4	0

ตารางที่ 5.27 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 3 และ 4

เมทธอด		*NCDM + NCDA*						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
Reading	date()	0	0	0	1	0	0	1	0	0	0
	amount()	0	0	0	1	0	0	1	0	0	0
Residential Site	addReading()	0	0	0	0	2	0	3	1	1	0
	charge()	0	0	0	0	6	0	4	0	4	0
	charge()	0	0	0	0	7	0	**5**	3	**1**	0
	fuelCharge()	0	0	0	0	1	0	1	1	0	0
	fuelChargeTaxes()	0	0	0	0	2	0	1	1	0	0
	summerFraction()	0	0	0	0	20	0	9	2	2	0
	dayOfYear(Date arg)	0	0	0	0	0	0	43	1	1	1
Zone	persist()	0	0	0	0	0	0	2	0	0	0
	get (String name)	0	0	0	0	0	0	1	1	0	0
	summerEnd()	0	0	0	0	0	1	1	0	0	0
	summerStart()	0	0	0	0	0	1	1	0	0	0
	winterRate()	0	0	0	0	0	1	1	0	0	0
	summerRate()	0	0	0	0	0	1	1	0	0	0

กรณีที่ 5 เป็นการประยุกต์ใช้วิธี Replace Parameters with Method เพื่อแก้ไขร่องรอยที่ไม่ดีประเภท Long Parameter Lists สำหรับเมทธอด chargeในคลาส DisabilitySite จากตารางที่ 5.28 และ 5.29 แสดงให้เห็นถึงการเปลี่ยนแปลงของค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคลาส และเมทธอดหลังการทำรีแฟคทอริง ซึ่งมีรายละเอียดดังนี้

1. ค่ามาตรวัดสำหรับคลาส จากตารางที่ 5.28 แสดงให้เห็นว่าค่ามาตรวัด *NIM* = 6 ของคลาส DisabilitySite เพิ่มขึ้นจากเดิม เพราะเป็นการสร้างเมทธอดใหม่ขึ้นมา แต่ไม่มีผลทำให้เกิดร่องรอยที่ไม่ดีประเภทอื่นเพิ่มขึ้น
2. ค่ามาตรวัดสำหรับเมทธอด จากตารางที่ 5.29 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 5 ได้ค่ามาตรวัด *NOP* = 2 ของคลาส DisabilitySite ซึ่งลดลง เมื่อเปรียบเทียบกับก่อนทำรีแฟคทอริง เพราะลดการส่งค่าพารามิเตอร์ เป็นการเรียกใช้เมทธอดใหม่ที่สร้างขึ้นมา

ตารางที่ 5.28 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 5

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
BusinessSite	4	3	0.334	0
DisabilitySite	6	2	0.1	0
LifelineSite	3	1	0.334	0
Reading	3	2	0	0
ResidentialSite	5	2	0.167	0
Zone	6	5	0	0

ตารางที่ 5.29 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 5

เมทธอด		*NCDM + NCDA*						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
BusinessSite	addReading(Reading newReading)	2	0	0	0	0	0	1	1	0	0
	charge()	5	0	0	0	0	0	1	0	1	0
	charge(int usage)	12	0	0	0	0	0	10	1	5	0
DisabilitySite	addReading(Reading newReading)	0	2	0	0	0	0	2	1	1	0
	charge()	0	6	0	0	0	0	3	0	4	0
	charge(int fullUsage, Date start, Date end)	0	25	0	0	0	0	14	**2**	5	0
	dayOfYear(Date arg)	0	0	0	0	0	0	43	1	1	1
	lastUsage()	0	0	0	0	0	0	3	0	1	0
LifelineSite	addReading(Reading newReading)	0	0	4	0	0	0	3	1	1	0
	charge()	0	0	3	0	0	0	1	0	1	0
	charge (int usage)	0	0	2	0	0	0	7	1	4	0

ตารางที่ 5.29 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 5

		มาตรวัด									
		NCDM + NCDA						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
Reading	date()	0	0	0	1	0	0	1	0	0	0
	amount()	0	0	0	1	0	0	1	0	0	0
ResidentialSite	addReading(Reading newReading)	0	0	0	0	2	0	3	1	1	0
	charge()	0	0	0	0	6	0	4	0	4	0
	charge(int usage, Date start, Date end)	0	0	0	0	20	0	13	3	4	0
	dayOfYear(Date arg)	0	0	0	0	0	0	43	1	1	1
Zone	persist()	0	0	0	0	0	0	2	0	0	0
	get (String name)	0	0	0	0	0	0	1	1	0	0
	summerEnd()	0	0	0	0	0	1	1	0	0	0
	summerStart()	0	0	0	0	0	1	1	0	0	0
	winterRate()	0	0	0	0	0	1	1	0	0	0
	summerRate()	0	0	0	0	0	1	1	0	0	0

กรณีที่ 6 เป็นการประยุกต์ใช้วิธี Replace Parameters with Method เพื่อแก้ไขร่องรอยที่ไม่ดีประเภท Long Parameter Lists สำหรับเมทธอด chargeในคลาส ResidentialSite จากตารางที่ 5.30 และ 5.31 แสดงให้เห็นถึงการเปลี่ยนแปลงของค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคลาส และเมทธอดหลังการทำรีแฟคทอริง ซึ่งมีรายละเอียดดังนี้

1. ค่ามาตรวัดสำหรับคลาส จากตารางที่ 5.30 แสดงให้เห็นว่าค่ามาตรวัด *NIM* = 6 ของคลาส ResidentialSite เพิ่มขึ้นจากเดิม เพราะเป็นการสร้างเมทธอดใหม่ขึ้นมา แต่ไม่มีผลทำให้เกิดร่องรอยที่ไม่ดีประเภทอื่นเพิ่มขึ้น
2. ค่ามาตรวัดสำหรับเมทธอด จากตารางที่ 5.31 ได้ค่ามาตรวัด *NOP* = 2 ของเมทธอด chargeในคลาส ResidentialSite ซึ่งลดลง เมื่อเปรียบเทียบกับก่อนทำรีแฟคทอริงเพราะลดการส่งค่าพารามิเตอร์ เป็นการเรียกใช้เมทธอดใหม่ที่สร้างขึ้นมา

ตารางที่ 5.30 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 6

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
BusinessSite	4	3	0.334	0
DisabilitySite	5	2	0.167	0
LifelineSite	3	1	0.334	0
Reading	3	2	0	0
ResidentialSite	**6**	2	0.1	0
Zone	6	5	0	0

ตารางที่ 5.31 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 6

		มาตรวัด									
		NCDM + NCDA						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
BusinessSite	addReading(Reading newReading)	2	0	0	0	0	0	1	1	0	0
	charge()	5	0	0	0	0	0	1	0	1	0
	charge(int usage)	12	0	0	0	0	0	10	1	5	0
DisabilitySite	addReading(Reading newReading)	0	2	0	0	0	0	2	1	1	0
	charge()	0	6	0	0	0	0	3	0	4	0
	charge(int fullUsage, Date start, Date end)	0	22	0	0	0	0	14	3	5	0
	dayOfYear(Date arg)	0	0	0	0	0	0	43	1	1	1
LifelineSite	addReading(Reading newReading)	0	0	4	0	0	0	3	1	1	0
	charge()	0	0	3	0	0	0	1	0	1	0
	charge (int usage)	0	0	2	0	0	0	7	1	4	0

ตารางที่ 5.31 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 6

เมทธอด		*NCDM + NCDA*						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
Reading	date()	0	0	0	1	0	0	1	0	0	0
	amount()	0	0	0	1	0	0	1	0	0	0
Residential Site	addReading(Reading newReading)	0	0	0	0	2	0	3	1	1	0
	charge()	0	0	0	0	6	0	4	0	4	0
	charge(int usage, Date start, Date end)	0	0	0	0	23	0	13	**2**	4	0
	dayOfYear(Date arg)	0	0	0	0	0	0	43	1	1	1
	lastUsage()	0	0	0	0	0	0	3	0	1	0
Zone	persist()	0	0	0	0	0	0	2	0	0	0
	get (String name)	0	0	0	0	0	0	1	1	0	0
	summerEnd()	0	0	0	0	0	1	1	0	0	0
	summerStart()	0	0	0	0	0	1	1	0	0	0
	winterRate()	0	0	0	0	0	1	1	0	0	0
	summerRate()	0	0	0	0	0	1	1	0	0	0

กรณีที่ 7 เป็นการประยุกต์ใช้วิธี Extract Method เพื่อแก้ไขร่องรอยที่ไม่ดีประเภท Long Method สำหรับเมทธอด charge ในคลาส BusinessSite จากตารางที่ 5.32 และ 5.33 แสดงให้เห็นถึงการเปลี่ยนแปลงของค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคลาส และเมทธอดหลังการทำรี-แฟคทอริง ซึ่งมีรายละเอียดดังนี้

1. ค่ามาตรวัดสำหรับคลาส จากตารางที่ 5.32 แสดงให้เห็นว่าค่ามาตรวัด *NIM* = 5 ของคลาส BusinessSite เพิ่มขึ้นจากเดิม เพราะเป็นการสร้างเมทธอดใหม่ขึ้นมา แต่ไม่มีผลทำให้เกิดร่องรอยที่ไม่ดีประเภทอื่นเพิ่มขึ้น
2. ค่ามาตรวัดสำหรับเมทธอด จากตารางที่ 5.33 แสดงให้เห็นว่าค่ามาตรวัด *NOS* = 7 สำหรับเมทธอด charge ในคลาส BusinessSite ซึ่งไม่อยู่ในช่วงของค่าที่พิจารณาของ Long Method แล้ว

ตารางที่ 5.32 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 7

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
BusinessSite	**5**	3	0.167	0
DisabilitySite	5	2	0.167	0
LifelineSite	3	1	0.334	0
Reading	3	2	0	0
ResidentialSite	5	2	0.167	0
Zone	6	5	0	0

ตารางที่ 5.33 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 7

เมทธอด		*NCDM + NCDA*						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
BusinessSite	addReading(Reading newReading)	2	0	0	0	0	0	1	1	0	0
	charge()	5	0	0	0	0	0	1	0	1	0
	charge(int usage)	1	0	0	0	0	0	7	1	2	0
	baseCharge()	12	0	0	0	0	0	3	0	3	0
DisabilitySite	addReading(Reading newReading)	0	2	0	0	0	0	2	1	1	0
	charge()	0	6	0	0	0	0	3	0	4	0
	charge(int fullUsage, Date start, Date end)	0	22	0	0	0	0	14	3	5	0
	dayOfYear(Date arg)	0	0	0	0	0	0	43	1	1	1
LifelineSite	addReading(Reading newReading)	0	0	4	0	0	0	3	1	1	0
	charge()	0	0	3	0	0	0	1	0	1	0
	charge (int usage)	0	0	2	0	0	0	7	1	4	0

ตารางที่ 5.33 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 7

เมทธอด		*NCDM + NCDA*						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
Reading	date()	0	0	0	1	0	0	1	0	0	0
	amount()	0	0	0	1	0	0	1	0	0	0
ResidentialSite	addReading(Reading newReading)	0	0	0	0	2	0	3	1	1	0
	charge()	0	0	0	0	6	0	4	0	4	0
	charge(int usage, Date start, Date end)	0	0	0	0	20	0	13	3	4	0
	dayOfYear(Date arg)	0	0	0	0	0	0	43	1	1	1
Zone	persist()	0	0	0	0	0	0	2	0	0	0
	get (String name)	0	0	0	0	0	0	1	1	0	0
	summerEnd()	0	0	0	0	0	1	1	0	0	0
	summerStart()	0	0	0	0	0	1	1	0	0	0
	winterRate()	0	0	0	0	0	1	1	0	0	0
	summerRate()	0	0	0	0	0	1	1	0	0	0

กรณีที่ 8 เป็นการประยุกต์ใช้วิธี Extract Method เพื่อแก้ไขร่องรอยที่ไม่ดีประเภท Long Method สำหรับเมทธอด dayOfYear ในคลาส DisabilitySite จากตารางที่ 5.34 และ 5.35 แสดงให้เห็นถึงการเปลี่ยนแปลงของค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคลาส และเมทธอดหลังการทำรี-แฟคทอริง ซึ่งมีรายละเอียดดังนี้

1. ค่ามาตรวัดสำหรับคลาส จากตารางที่ 5.34 แสดงให้เห็นว่าค่ามาตรวัด *NIM* = 6 ของคลาส DisabilitySite เพิ่มขึ้นจากเดิม เพราะเป็นการสร้างเมทธอดใหม่ขึ้นมา แต่ไม่มีผลทำให้เกิดร่องรอยที่ไม่ดีประเภทอื่นเพิ่มขึ้น
2. ค่ามาตรวัดสำหรับเมทธอด จากตารางที่ 5.35 แสดงให้เห็นว่าค่ามาตรวัด *NOS* = 5 สำหรับเมทธอด dayOfYear ในคลาส DisabilitySite ซึ่งไม่อยู่ในช่วงของค่าที่พิจารณาของ Long Method แล้ว

ตารางที่ 5.34 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 8

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
BusinessSite	4	3	0.334	0
DisabilitySite	6	2	0.1	0
LifelineSite	3	1	0.334	0
Reading	3	2	0	0
ResidentialSite	5	2	0.167	0
Zone	6	5	0	0

ตารางที่ 5.35 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 8

เมทธอด		*NCDM + NCDA*						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
BusinessSite	addReading(Reading newReading)	2	0	0	0	0	0	1	1	0	0
	charge()	5	0	0	0	0	0	1	0	1	0
	charge(int usage)	12	0	0	0	0	0	10	1	5	0
DisabilitySite	addReading(Reading newReading)	0	2	0	0	0	0	2	1	1	0
	charge()	0	6	0	0	0	0	3	0	4	0
	charge(int fullUsage, Date start, Date end)	0	22	0	0	0	0	14	3	5	0
	dayOfYear(Date arg)	0	0	0	0	0	0	5	1	1	1
	isLeapYear()	0	0	0	0	0	0	1	0	0	0
LifelineSite	addReading(Reading newReading)	0	0	4	0	0	0	3	1	1	0
	charge()	0	0	3	0	0	0	1	0	1	0
	charge (int usage)	0	0	2	0	0	0	7	1	4	0

ตารางที่ 5.35 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 8

เมทธอด		*NCDM + NCDA*						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
Reading	date()	0	0	0	1	0	0	1	0	0	0
	amount()	0	0	0	1	0	0	1	0	0	0
ResidentialSite	addReading(Reading newReading)	0	0	0	0	2	0	3	1	1	0
	charge()	0	0	0	0	6	0	4	0	4	0
	charge(int usage, Date start, Date end)	0	0	0	0	20	0	13	3	4	0
	dayOfYear(Date arg)	0	0	0	0	0	0	43	1	1	1
Zone	persist()	0	0	0	0	0	0	2	0	0	0
	get (String name)	0	0	0	0	0	0	1	1	0	0
	summerEnd()	0	0	0	0	0	1	1	0	0	0
	summerStart()	0	0	0	0	0	1	1	0	0	0
	winterRate()	0	0	0	0	0	1	1	0	0	0
	summerRate()	0	0	0	0	0	1	1	0	0	0

กรณีที่ 9 เป็นการประยุกต์ใช้วิธี Extract Method เพื่อแก้ไขร่องรอยที่ไม่ดีประเภท Long Method สำหรับเมทธอด dayOfYear ในคลาส ResidentialSite จากตารางที่ 5.36 และ 5.37 แสดงให้เห็นถึงการเปลี่ยนแปลงของค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคลาส และเมทธอดหลังการทำรีแฟคทอริง ซึ่งมีรายละเอียดดังนี้

1. ค่ามาตรวัดสำหรับคลาส จากตารางที่ 5.36 แสดงให้เห็นว่าค่ามาตรวัด *NIM* = 6 ของคลาส ResidentialSite เพิ่มขึ้นจากเดิม เพราะเป็นการสร้างเมทธอดใหม่ขึ้นมา แต่ไม่มีผลทำให้เกิดร่องรอยที่ไม่ดีประเภทอื่นเพิ่มขึ้น
2. ค่ามาตรวัดสำหรับเมทธอด จากตารางที่ 5.37 แสดงให้เห็นว่าค่ามาตรวัด *NOS* = 5 สำหรับเมทธอด dayOfYear ในคลาส ResidentialSite ซึ่งไม่อยู่ในช่วงของค่าที่พิจารณาของ Long Method แล้ว

ตารางที่ 5.36 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 9

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
BusinessSite	4	3	0.334	0
DisabilitySite	5	2	0.167	0
LifelineSite	3	1	0.334	0
Reading	3	2	0	0
ResidentialSite	**6**	2	0.1	0
Zone	6	5	0	0

ตารางที่ 5.37 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 9

เมทธอด		*NCDM + NCDA*						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
BusinessSite	addReading(Reading newReading)	2	0	0	0	0	0	1	1	0	0
	charge()	5	0	0	0	0	0	1	0	1	0
	charge(int usage)	12	0	0	0	0	0	10	1	5	0
DisabilitySite	addReading(Reading newReading)	0	2	0	0	0	0	2	1	1	0
	charge()	0	6	0	0	0	0	3	0	4	0
	charge(int fullUsage, Date start, Date end)	0	22	0	0	0	0	14	3	5	0
	dayOfYear(Date arg)	0	0	0	0	0	0	43	1	1	1
LifelineSite	addReading(Reading newReading)	0	0	4	0	0	0	3	1	1	0
	charge()	0	0	3	0	0	0	1	0	1	0
	charge (int usage)	0	0	2	0	0	0	7	1	4	0

ตารางที่ 5.37 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 9

เมทธอด		*NCDM + NCDA*						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
Reading	date()	0	0	0	1	0	0	1	0	0	0
	amount()	0	0	0	1	0	0	1	0	0	0
Residential Site	addReading(Reading newReading)	0	0	0	0	2	0	3	1	1	0
	charge()	0	0	0	0	6	0	4	0	4	0
	charge()	0	0	0	0	20	0	13	3	4	0
	dayOfYear(Date arg)	0	0	0	0	0	0	**5**	1	1	1
	isLeapYear()	0	0	0	0	0	0	1	0	0	0
Zone	persist()	0	0	0	0	0	0	2	0	0	0
	get (String name)	0	0	0	0	0	0	1	1	0	0
	summerEnd()	0	0	0	0	0	1	1	0	0	0
	summerStart()	0	0	0	0	0	1	1	0	0	0
	winterRate()	0	0	0	0	0	1	1	0	0	0
	summerRate()	0	0	0	0	0	1	1	0	0	0

กรณีที่ 10 เป็นการประยุกต์ใช้วิธี Extract Method เพื่อแก้ไขร่องรอยที่ไม่ดีประเภท Switch Statement สำหรับเมทธอด dayOfYear ในคลาส DisabilitySite จากตารางที่ 5.38 และ 5.39 แสดงให้เห็นถึงการเปลี่ยนแปลงของค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคลาส และเมทธอดหลังการทำรีแฟคทอริง ซึ่งมีรายละเอียดดังนี้

1. ค่ามาตรวัดสำหรับคลาส จากตารางที่ 5.38 แสดงให้เห็นว่าค่ามาตรวัด *NIM* = 6 ของคลาส DisabilitySite เพิ่มขึ้นจากเดิม เพราะเป็นการสร้างเมทธอดใหม่ขึ้นมา แต่ไม่มีผลทำให้เกิดร่องรอยที่ไม่ดีประเภทอื่นเพิ่มขึ้น
2. ค่ามาตรวัดสำหรับเมทธอด จากตารางที่ 5.39 ได้ค่ามาตรวัด *NOSS* = 0 สำหรับเมทธอด dayOfYear ในคลาส DisabilitySite ซึ่งไม่อยู่ในช่วงของค่าที่พิจารณาของ Switch Statement แล้ว

ตารางที่ 5.38 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 10

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
BusinessSite	4	3	0.334	0
DisabilitySite	**6**	2	0.1	0
LifelineSite	3	1	0.334	0
Reading	3	2	0	0
ResidentialSite	5	2	0.167	0
Zone	6	5	0	0

ตารางที่ 5.39 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 10

เมทธอด		*NCDM + NCDA*						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
BusinessSite	addReading(Reading newReading)	2	0	0	0	0	0	1	1	0	0
	charge()	5	0	0	0	0	0	1	0	1	0
	charge(int usage)	12	0	0	0	0	0	10	1	5	0
DisabilitySite	addReading(Reading newReading)	0	2	0	0	0	0	2	1	1	0
	charge()	0	6	0	0	0	0	3	0	4	0
	charge(int fullUsage, Date start, Date end)	0	22	0	0	0	0	14	3	5	0
	dayOfYear(Date arg)	0	0	0	0	0	0	5	1	1	**0**
	daysToStartOfMonth()	0	0	0	0	0	0	1	0	1	0
LifelineSite	addReading(Reading newReading)	0	0	4	0	0	0	3	1	1	0
	charge()	0	0	3	0	0	0	1	0	1	0
	charge (int usage)	0	0	2	0	0	0	7	1	4	0

ตารางที่ 5.39 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 10

เมทธอด		*NCDM + NCDA*						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
Reading	date()	0	0	0	1	0	0	1	0	0	0
	amount()	0	0	0	1	0	0	1	0	0	0
ResidentialSite	addReading(Reading newReading)	0	0	0	0	2	0	3	1	1	0
	charge()	0	0	0	0	6	0	4	0	4	0
	charge(int usage, Date start, Date end)	0	0	0	0	20	0	13	3	4	0
	dayOfYear(Date arg)	0	0	0	0	0	0	43	1	1	1
Zone	persist()	0	0	0	0	0	0	2	0	0	0
	get (String name)	0	0	0	0	0	0	1	1	0	0
	summerEnd()	0	0	0	0	0	1	1	0	0	0
	summerStart()	0	0	0	0	0	1	1	0	0	0
	winterRate()	0	0	0	0	0	1	1	0	0	0
	summerRate()	0	0	0	0	0	1	1	0	0	0

กรณีที่ 11 เป็นการประยุกต์ใช้วิธี Extract Method เพื่อแก้ไขร่องรอยที่ไม่ดีประเภท Switch Statement สำหรับเมทธอด dayOfYear ในคลาส ResidentialSite จากตารางที่ 5.40 และ 5.41 แสดงให้เห็นถึงการเปลี่ยนแปลงของค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคลาส และเมทธอดหลังการทำรีแฟคทอริง ซึ่งมีรายละเอียดดังนี้

1. ค่ามาตรวัดสำหรับคลาส จากตารางที่ 5.40 แสดงให้เห็นว่าค่ามาตรวัด *NIM* = 6 ของคลาส ResidentialSite เพิ่มขึ้นจากเดิม เพราะเป็นการสร้างเมทธอดใหม่ขึ้นมา แต่ไม่มีผลทำให้เกิดร่องรอยที่ไม่ดีประเภทอื่นเพิ่มขึ้น
2. ค่ามาตรวัดสำหรับเมทธอด จากตารางที่ 5.41 ได้ค่ามาตรวัด *NOSS* = 0 สำหรับเมทธอด dayOfYear ในคลาส ResidentialSite ซึ่งไม่อยู่ในช่วงของค่าที่พิจารณาของ Switch Statement แล้ว

ตารางที่ 5.40 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 11

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
BusinessSite	4	3	0.334	0
DisabilitySite	5	2	0.167	0
LifelineSite	3	1	0.334	0
Reading	3	2	0	0
ResidentialSite	6	2	0.1	0
Zone	6	5	0	0

ตารางที่ 5.41 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 11

เมทธอด		*NCDM + NCDA*						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
BusinessSite	addReading(Reading newReading)	2	0	0	0	0	0	1	1	0	0
	charge()	5	0	0	0	0	0	1	0	1	0
	charge(int usage)	12	0	0	0	0	0	10	1	5	0
DisabilitySite	addReading(Reading newReading)	0	2	0	0	0	0	2	1	1	0
	charge()	0	6	0	0	0	0	3	0	4	0
	charge(int fullUsage, Date start, Date end)	0	22	0	0	0	0	14	3	5	0
	dayOfYear(Date arg)	0	0	0	0	0	0	43	1	1	1
LifelineSite	addReading(Reading newReading)	0	0	4	0	0	0	3	1	1	0
	charge()	0	0	3	0	0	0	1	0	1	0
	charge (int usage)	0	0	2	0	0	0	7	1	4	0

ตารางที่ 5.41 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 11

เมทธอด		*NCDM + NCDA*						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
Reading	date()	0	0	0	1	0	0	1	0	0	0
	amount()	0	0	0	1	0	0	1	0	0	0
Residential Site	addReading(Reading newReading)	0	0	0	0	2	0	3	1	1	0
	charge()	0	0	0	0	6	0	4	0	4	0
	charge()	0	0	0	0	20	0	13	3	4	0
	dayOfYear(Date arg)	0	0	0	0	0	0	5	1	1	**0**
	daysToStartOfMonth()	0	0	0	0	0	0	1	0	1	0
Zone	persist()	0	0	0	0	0	0	2	0	0	0
	get (String name)	0	0	0	0	0	0	1	1	0	0
	summerEnd()	0	0	0	0	0	1	1	0	0	0
	summerStart()	0	0	0	0	0	1	1	0	0	0
	winterRate()	0	0	0	0	0	1	1	0	0	0
	summerRate()	0	0	0	0	0	1	1	0	0	0

การแก้ไขร่องรอยที่ไม่ดีทุกประเภท

หลังจากการประยุกต์ใช้รีแฟคทอริงสำหรับโปรแกรมที่ 1 ทั้ง 11 กรณี ได้ค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคลาส และเมทธอด แสดงรายละเอียดดังตารางที่ 5.42 และ 5.43 ซึ่งมีรายละเอียดดังนี้

1. ค่ามาตรวัดสำหรับคลาส จากตารางที่ 5.42 ค่ามาตรวัด *NIM* สำหรับคลาส BusinessSite DisabilitySite และ ResidentialSite เพิ่มขึ้น เพราะเป็นการสร้างเมทธอดใหม่ แต่ไม่ได้ทำให้ร่องรอยที่ไม่ดีใหม่เกิดขึ้น
2. ค่ามาตรวัดสำหรับเมทธอด จากตารางที่ 5.43
    1) ค่ามาตรวัด *NOS* สำหรับเมทธอด charge ในคลาส BusinessSite เมทธอด charge และ dayOfYear ในคลาส DisabilitySite เมทธอด charge และ dayOfYear ในคลาส ResidentialSite ไม่อยู่ในช่วงของค่าที่พิจารณาของ Long Method แล้ว
    2) ค่ามาตรวัด *NOP* สำหรับเมทธอด charge ในคลาส DisabilitySite และ ResidentialSite ลดลง ซึ่งไม่อยู่ในช่วงของค่าที่พิจารณาของ Long Parameter Lists แล้ว
    3) ค่ามาตรวัด *NOSS* = 0 สำหรับเมทธอด dayOfYear ในคลาส DisabilitySite และ ResidentialSite ซึ่งไม่อยู่ในช่วงของค่าที่พิจารณาของ Switch Statement แล้ว

ตารางที่ 5.42 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงทั้ง 11 กรณี

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
BusinessSite	**5**	3	0.167	0
DisabilitySite	**11**	2	0.046	0
LifelineSite	3	1	0.334	0
Reading	3	2	0	0
ResidentialSite	**11**	2	0.046	0
Zone	6	5	0	0

ตารางที่ 5.43 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงทั้ง 11 กรณี

เมทธอด		*NCDM + NCDA*						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
BusinessSite	addReading()	2	0	0	0	0	0	1	1	0	0
	charge()	5	0	0	0	0	0	1	0	1	0
	charge(int usage)	1	0	0	0	0	0	**7**	1	2	0
	baseCharge()	12	0	0	0	0	0	3	0	3	0
DisabilitySite	addReading()	0	2	0	0	0	0	2	1	1	0
	charge()	0	4	0	0	0	0	3	0	3	0
	charge()	0	11	0	0	0	0	**6**	**2**	2	0
	fuelCharge()	0	1	0	0	0	0	1	1	0	0
	fuelChargeTaxes()	0	3	0	0	0	0	1	1	0	0
	summerFraction()	0	20	0	0	0	0	9	2	2	0
	dayOfYear(Date arg)	0	2	0	0	0	0	**5**	1	1	**0**
	lastUsage()	0	0	0	0	0	0	3	0	1	0
	isLeapYear()	0	0	0	0	0	0	1	0	0	0
	dayToStartOfMonth()	0	0	0	0	0	0	1	0	1	0
LifelineSite	addReading()	0	0	4	0	0	0	3	1	1	0
	charge()	0	0	3	0	0	0	1	0	1	0
	charge (int usage)	0	0	2	0	0	0	7	1	4	0

ตารางที่ 5.43 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 2 หลังประยุกต์ใช้รีแฟคทอริงทั้ง 11 กรณี

เมทธอด		*NCDM + NCDA*						*NOS*	*NOP*	*NOT*	*NOSS*
		BusinessSite	DisabilitySite	LifelineSite	Reading	ResidentialSite	Zone				
Reading	date()	0	0	0	1	0	0	1	0	0	0
	amount()	0	0	0	1	0	0	1	0	0	0
Residential Site	addReading()	0	0	0	0	2	0	3	1	1	0
	charge()	0	0	0	0	6	0	4	0	4	0
	charge()	0	0	0	0	10	0	**5**	**2**	1	0
	fuelCharge()	0	0	0	0	1	0	1	1	0	0
	fuelChargeTaxes()	0	0	0	0	2	0	1	1	0	0
	summerFraction()	0	0	0	0	20	0	9	2	2	0
	dayOfYear(Date arg)	0	0	0	0	2	0	**5**	1	1	**0**
	isLeapYear()	0	0	0	0	0	0	1	0	0	0
	dayToStartOfMonth()	0	0	0	0	0	0	1	0	1	0
	lastUsage()	0	0	0	0	0	0	3	0	1	0
Zone	persist()	0	0	0	0	0	0	2	0	0	0
	summerEnd()	0	0	0	0	0	1	1	0	0	0
	summerStart()	0	0	0	0	0	1	1	0	0	0
	winterRate()	0	0	0	0	0	1	1	0	0	0
	summerRate()	0	0	0	0	0	1	1	0	0	0

5.2.3 ผลการทดสอบโปรแกรมที่ 3

การแก้ไขร่องรอยที่ไม่ดีแต่ละประเภท

จากการแก้ไขร่องรอยที่ไม่ดีที่พบในโปรแกรมที่ 3 ได้ผลของค่ามาตรวัดร่องรอยที่ไม่ดีหลังการประยุกต์ใช้วิธีรีแฟคทอริงแต่ละกรณี 2 กรณี ดังนี้

กรณีที่ 1 เป็นการประยุกต์ใช้วิธี Extract Class เพื่อแก้ไขร่องรอยที่ไม่ดีประเภท Large Class สำหรับคลาส ElevatorSimulation จากตารางที่ 5.44 ได้ค่ามาตรวัด *NIM* = 23 และ *NIV* = 8 ซึ่งลดลง เมื่อเปรียบเทียบกับก่อนทำรีแฟคทอริง เพราะย้ายเมทธอดและคุณลักษณะบางส่วนไปสร้างเป็นคลาสใหม่

ตารางที่ 5.44 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 3 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 1

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
AnimatedPanel	14	10	0.077	3
Bell	4	1	0.167	0
BellEvent	1	0	0	1
Button	6	2	0.2	0
ButtonEvent	1	0	0	1
Door	6	4	0.4	0
DoorEvent	1	0	0	1
Elevator	20	15	0.058	1
ElevatorController	1	3	0	1
ElevatorDoor	4	0	0	1
ElevatorMoveEvent	1	0	0	1
ElevatorShaft	18	12	0.037	0
ElevatorSimulation	**23**	**8**	0.069	0
ElevatorSimulationConstants	4	3	0.334	0
ElevatorSimulationEvent	5	2	0.334	0
ElevatorView	28	56	0.043	1
Floor	5	1	0.167	1
ImagePanel	11	4	0.111	1

ตารางที่ 5.44 (ต่อ) ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 3 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 1

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
Light	6	4	0.2	0
LightEvent	1	0	0	1
Location	4	1	0.167	0
MovingPanel	7	3	0.2	2
Person	10	11	0.139	1
PersonMoveEvent	2	1	0	1
SoundEffects	3	1	1	0
AnimatedPanel	14	10	0.077	3

กรณีที่ 2 เป็นการประยุกต์ใช้วิธี Inline Class เพื่อแก้ไขร่องรอยที่ไม่ดีประเภท Lazy Class สำหรับคลาส BellEvent, ButtonEvent, DoorEvent, ElevatorEvent และ LightEvent จากตารางที่ 5.45 ได้ค่ามาตรวัด *NIM* = 0 และ *NIV* = 0

ตารางที่ 5.45 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 3 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 2

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
AnimatedPanel	14	10	0.077	3
Bell	4	1	0.167	0
BellEvent	**0**	**0**	0	0
Button	6	2	0.2	0
ButtonEvent	**0**	**0**	0	0
Door	6	4	0.4	0
DoorEvent	**0**	**0**	0	0
Elevator	20	15	0.058	1
ElevatorController	1	3	0	1
ElevatorDoor	4	0	0	1
ElevatorMoveEvent	**0**	**0**	0	0
ElevatorShaft	18	12	0.037	0

ตารางที่ 5.44 (ต่อ) ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 3 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 2

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
ElevatorSimulation	26	10	0.034	0
ElevatorSimulationEvent	5	2	0.334	0
ElevatorView	28	56	0.043	1
Floor	5	1	0.167	1
ImagePanel	11	4	0.111	1
Light	6	4	0.2	0
LightEvent	0	0	0	0
Location	4	1	0.167	0
MovingPanel	7	3	0.2	2
Person	10	11	0.139	1
PersonMoveEvent	2	1	0	1
SoundEffects	3	1	1	0
AnimatedPanel	14	10	0.077	3

การแก้ไขร่องรอยที่ไม่ดีทุกประเภท

หลังจากการประยุกต์ใช้รีแฟคทอริงสำหรับโปรแกรมที่ 3 ทั้ง 2 กรณี ได้ค่ามาตรวัดร่องรอยที่ไม่ดีสำหรับคลาส แสดงรายละเอียดดังตารางที่ 5.46 ซึ่งมีรายละเอียดดังนี้

1. ค่ามาตรวัด *NIM* = 23 และ *NIV* = 8 สำหรับคลาส ElevatorSimulation ซึ่งลดลงเมื่อเปรียบเทียบกับก่อนทำรีแฟคทอริง เพราะย้ายเมทธอดและคุณลักษณะบางส่วนไปสร้างเป็นคลาสใหม่
2. ค่ามาตรวัด *NIM* = 0 และ *NIV* = 0 สำหรับคลาส BellEvent, ButtonEvent, DoorEvent, ElevatorEvent และ LightEvent ซึ่งไม่อยู่ในช่วงของค่าที่พิจารณาสำหรับ Lazy Class แล้ว

ตารางที่ 5.46 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 3 หลังประยุกต์ใช้รีแฟคทอริงในกรณีที่ 1 และ 2

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
AnimatedPanel	14	10	0.077	3
Bell	4	1	0.167	0
BellEvent	**1**	**0**	0	1
Button	6	2	0.2	0
ButtonEvent	**1**	**0**	0	1
Door	6	4	0.4	0
DoorEvent	**1**	**0**	0	1
Elevator	20	15	0.058	1
ElevatorController	1	3	0	1
ElevatorDoor	4	0	0	1
ElevatorMoveEvent	**1**	**0**	0	1
ElevatorShaft	18	12	0.037	0
ElevatorSimulation	**23**	**8**	0.069	0
ElevatorSimulationConstants	4	3	0.334	0
ElevatorSimulationEvent	5	2	0.334	0
ElevatorView	28	56	0.043	1
Floor	5	1	0.167	1
ImagePanel	11	4	0.111	1
Light	6	4	0.2	0
LightEvent	**1**	**0**	0	1
Location	4	1	0.167	0
MovingPanel	7	3	0.2	2
Person	10	11	0.139	1
PersonMoveEvent	2	1	0	1
SoundEffects	3	1	1	0
AnimatedPanel	14	10	0.077	3

5.3 การเลือกวิธีรีแฟคทอริง

การประเมินความสามารถของมาตรวัดร่องรอยที่ไม่ดีทั้ง 3 โปรแกรม แสดงให้เห็นถึงผลกระทบของค่ามาตรวัดร่องรอยที่ไม่ดี หลังจากประยุกต์ใช้รีแฟคทอริงเพียง 1 วิธี เพื่อแก้ไขร่องรอยที่ไม่ดีเพียง 1 ประเภท และประยุกต์ใช้รีแฟคทอริงหลายวิธี เพื่อแก้ไขร่องรอยที่ไม่ดีทุกประเภทที่พบ จากผลการทดสอบพบว่า

1. การแก้ไขร่องรอยที่ไม่ดีแต่ละประเภท สามารถแก้ไขร่องรอยที่ไม่ดีที่พบได้ แต่มีวิธีการรีแฟคทอริงบางวิธี เช่น วิธี Move Method เมื่อประยุกต์ใช้แล้ว อาจทำให้เกิดร่องรอยที่ไม่ดีประเภทอื่นได้
2. การแก้ไขร่องรอยที่ไม่ดีทุกประเภท สามารถแก้ไขร่องรอยที่ไม่ดีที่พบทุกประเภทได้ โดยไม่ทำให้เกิดร่องรอยที่ไม่ดีประเภทอื่น

ดังนั้นการตัดสินใจว่าจะแก้ไขร่องรอยที่ไม่ดีประเภทใด ขึ้นอยู่กับการตัดสินใจของผู้พัฒนา

5.4 การตรวจสอบค่ามาตรวัดที่ได้จากเครื่องมือ

ผู้วิจัยได้ทำการตรวจสอบความถูกต้องของค่ามาตรวัดร่องรอยที่ไม่ดีทั้ง 13 มาตรวัดที่คำนวณได้จากเครื่องมือสำหรับการคำนวณมาตรวัดและทำนายความสามารถในการบำรุงรักษาซอฟต์แวร์ กับค่ามาตรวัดที่ได้จากวิธีการนับและการคำนวณด้วยมือ ด้วยโปรแกรมที่นำมาใช้ในการทดสอบทั้ง 3 โปรแกรม และทำการเปรียบเทียบค่าของมาตรวัดที่คำนวณได้จากทั้งสองวิธี ผลการเปรียบเทียบพบว่าเครื่องมือที่พัฒนาขึ้นสามารถคำนวณค่ามาตรวัดได้ถูกต้องตรงกับการคำนวณด้วยการนับและการคำนวณด้วยมือ

บทที่ 6

บทสรุปและข้อเสนอแนะ

6.1 บทสรุป

วิทยานิพนธ์ฉบับนี้ได้นำเสนอวิธีการตรวจจับร่องรอยที่ไม่ดี สำหรับซอร์สโค้ดภาษาจาวา โดยทำการออกแบบมาตรวัดร่องรอยที่ไม่ดีสำหรับร่องรอยที่ไม่ดี 6 ประเภท คือ Feature Envy, Large Class, Lazy Class, Long Method, Long Parameter Lists และ Switch Statement โดยการอธิบายและกำหนดนิยามของมาตรวัดสำหรับตรวจจับ และหาตำแหน่งของร่องรอยที่ไม่ดี จำแนกเป็น 6 ส่วนคือ นิยาม แรงจูงใจ วิธีการวัด มาตรวัดร่องรอยที่ไม่ดี ข้อกำหนดของค่าที่พิจารณา และการประยุกต์ใช้รีแฟคทอริง จากนั้นได้พัฒนาเครื่องมือตรวจจับร่องรอยที่ไม่ดีด้วยภาษาจาวา ให้สามารถคำนวณค่ามาตรวัดร่องรอยที่ไม่ดีจากซอร์สโค้ด รวมทั้งเสนอแนะวิธีรีแฟคทอริงเพื่อแก้ไขร่องรอยที่ไม่ดีแต่ละประเภทที่พบตามที่ได้ออกแบบไว้ และประเมินความสามารถของมาตรวัดร่องรอยที่ไม่ดี ซึ่งนำเครื่องมือมาทดสอบกับ 3 โปรแกรม โดยเปรียบเทียบค่ามาตรวัดร่องรอยที่ไม่ดีก่อนและหลังการประยุกต์ใช้รีแฟคทอริง เพื่อตรวจสอบว่าร่องรอยที่ไม่ดีได้ถูกแก้ไขหลังจากการรีแฟคทอริงแต่ละวิธี

ผลการทดสอบพบว่า หลังจากตรวจจับร่องรอยที่ไม่ดีด้วยเครื่องมือที่พัฒนาขึ้น และแก้ไขร่องรอยที่ไม่ดีที่พบด้วยวิธีรีแฟคทอริงที่เสนอแล้ว ทั้งการแก้ไขร่องรอยที่ไม่ดีแต่ละประเภท และการแก้ไขร่องรอยที่ไม่ดีทุกประเภท ทำให้ค่ามาตรวัดร่องรอยที่ไม่ดีไม่อยู่ในช่วงของข้อกำหนดของค่าที่พิจารณาสำหรับร่องรอยที่ไม่ดีแต่ละประเภทแล้ว แสดงว่าร่องรอยที่ไม่ดีที่พบในโปรแกรมทดสอบทั้ง 3 โปรแกรม ได้ถูกแก้ไขแล้ว

วิธีการตรวจจับร่องรอยที่ไม่ดี และมาตรวัดร่องรอยที่ไม่ดีที่ได้นำเสนอในวิทยานิพนธ์นี้เป็นแนวทางสำหรับช่วยในการตรวจตำแหน่งของร่องรอยที่ไม่ดีแต่ละประเภท ส่วนการตัดสินใจว่าจะแก้ไขร่องรอยที่ไม่ดีประเภทใด ขึ้นอยู่กับการตัดสินใจของผู้พัฒนา

6.2 ข้อเสนอแนะ

1. มาตรวัดร่องรอยที่ไม่ดีออกแบบไว้สำหรับร่องรอยที่ไม่ดี 6 ประเภท ดังรายละเอียดข้างต้น ซึ่งสามารถนำวิธีการออกแบบมาตรวัดร่องรอยที่ไม่ดีในงานวิจัยนี้เป็นแนวทางในการออกแบบมาตรวัดสำหรับร่องรอยที่ไม่ดีประเภทอื่นๆ ต่อไป

2. ข้อกำหนดของค่าที่พิจารณาของมาตรวัดร่องรอยที่ไม่ดีที่ใช้ในงานวิจัยนี้ บางค่าสำหรับมาตรวัดที่มีอยู่แล้ว เช่น มาตรวัด *NIM, NIV* และ *NOS* เป็นช่วงค่าที่ไม่ดีที่ใช้เป็นค่าสำหรับโปรแกรมภาษาซีพลัสพลัส ดังนั้นถ้าสามารถหาช่วงค่าสำหรับโปรแกรมภาษาจาวา และนำมาปรับปรุงเครื่องมือ จะสามารถตรวจจับร่องรอยที่ไม่ดีสำหรับโปรแกรมภาษาจาวาได้ดีขึ้น
3. มาตรวัดร่องรอยที่ไม่ดีในงานวิจัยนี้ ช่วยตรวจหาร่องรอยที่ไม่ดีในซอร์สโค้ดว่าอยู่ที่คลาส เมทธอด หรือคุณลักษณะใด และเสนอวิธีรีแฟคทอริงเพื่อประยุกต์ใช้ ซึ่งมีได้มากกว่าหนึ่งวิธี แต่ไม่ได้คำนึงถึงการเลือกวิธีรีแฟคทอริงมาประยุกต์ใช้ และจะแก้ไขร่องรอยที่ไม่ดีใดก่อน และหลังตามลำดับ ดังนั้นงานวิจัยนี้จะเป็นประโยชน์มากขึ้น ถ้าสามารถเปรียบเทียบความสามารถของการเลือกวิธีรีแฟคทอริงแต่ละวิธีมาแก้ไข และวิเคราะห์ถึงลำดับการแก้ไขร่องรอยที่ไม่ดีได้ จะทำให้ความสามารถในการบำรุงรักษาซอฟต์แวร์ดีขึ้นได้
4. ถ้าประเมินความสามารถของมาตรวัดร่องรอยที่ไม่ดี โดยเปรียบเทียบค่ามาตรวัดอื่นๆ ทางด้านการบำรุงรักษา เช่น มาตรวัดการขึ้นต่อกัน มาตรวัดการเกาะกันเป็นก้อน หรือมาตรวัดคุณภาพในด้านอื่นๆ เช่น มาตรวัดทางด้านการทำความเข้าใจซอฟต์แวร์ จะทำให้มาตรวัดร่องรอยที่ไม่ดีนี้ น่าเชื่อถือมากยิ่งขึ้น

6.3 ผลงานตีพิมพ์

ผลงานวิจัยนี้ ได้รับคัดเลือกให้ถูกตีพิมพ์ในงานประชุมทางวิชาการนานาชาติ “Computer Science, Software Engineering, Information Technology, e-business and Application 2004 (CSITeA-04)” ซึ่งได้จัดขึ้นที่เมืองไคโร ประเทศอียิปต์ ระหว่างวันที่ 27- 29 ธันวาคม 2547 โดยรายละเอียดของบทความแสดงอยู่ในภาคผนวก ฉ

รายการอ้างอิง

1. Simon, F., Steinbruckner, F. and Lewerentz, C. (2001). Metrics Based Refactoring. Proceeding of 5th European Conference on Software Maintenance and Reengineering.:30-38. IEEE Computer Society Press.
2. Tourwé, T. and Mens, T. (2004) A Survey of Software Refactoring. IEEE Transactions on Software Engineering. 30(2):126-139.
3. Ciupke, O. (1999). Automatic Detection of Design Problems in Object-Oriented Reengineering. Technology of Object-Oriented Languages and Systems – TOOLS 30. IEEE Computer Society.
4. Fenton, N. E. and Pfleeger, S. L. (1997). Software Metrics: A Rigorous and Practical Approach. PWS Publishing Company.
5. Fowler, M. (1999). Refactoring: Improving the Design of Existing Code. United States: Addison-Wesley.
6. Kataoka, Y., Imai, T., Andou, H. and Fukaya, T. (2002). A Quantitative Evaluation of Maintainability Enhancement by Refactoring. Proceedings of the International Conference on Software Maintenance (ICSM'02).:576-585.
7. Marinescu, R. (2001). Detecting Design Flaws via Metrics in Object-Oriented Systems. Proceeding of the 39th International Conference and Exhibition on Technology of Object-Oriented Languages and Systems (TOOLS39).:173-182.
8. Riel, A. J. (1996). Object-Oriented Design Heuristics. Addison-Wesley.
9. Chidamber, S. R. and Kemerer, C.F. (1994). A Metric Suit for Object-Oriented Design. IEEE Transaction on Software Engineering 20(6):476-493.
10. Bieman, J.M. and Kang, B.K. (1995). Cohesion and Reuse in Object-Oriented Systems. Proceeding of the ACM Symposium on Software Reusability.:259-266.
11. Lee, S., Yu-Seung M., Yong R. K. and Heung S. (2004). A Systematic Class Refactoring Approach Based On Metrics. Proceeding of the International Conference on Computer Science, Software Engineering, Information Technology, e-Business, and Applications (CSITeA'04).
12. Lorenz, M. and Kidd. J. (1994). Object-Oriented Software Metrics: A Practical Guide. New Jersey: Prentice Hall.

13. Genero, M., Piattini, M. and Calero, (2000). C. Early Measures for UML class diagrams. L'OBJECT: Software, Database, Network 6: 489-515.
14. สมหวัง แซ่ตั้ง. (2543). การออกแบบและพัฒนาเครื่องมือวัดซอฟต์แวร์สำหรับโปรแกรมเชิงวัตถุ. วิทยานิพนธ์ปริญญามหาบัณฑิต สาขาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย.
15. Deitel, H. M. and Deitel, P. J. (2004). Java: How to program. New Jersey: Prentice Hall.

ภาคผนวก

ภาคผนวก ก
วิธีรีแฟคทอริง

ในภาคผนวกนี้จะแสดงรายละเอียดวิธีการรีแฟคทอริงทั้งหมด 72 วิธี ซึ่งมีรายละเอียดดังนี้

Composing Method เป็นวิธีรีแฟคทอริงสำหรับการเปลี่ยนแปลงเมทธอด ดังนี้

1. วิธี Extract Method เป็นการย้ายโค้ดบางส่วนจากเมทธอดเดิม ไปสร้างเป็นเมทธอดใหม่
2. วิธี Inline Method เป็นการแทนที่การเรียกใช้เมทธอดด้วยโค้ดภายในเมทธอดนั้น แล้วทำการลบเมทธอดเดิมนั้นทิ้ง
3. วิธี Inline Temp เป็นการแทนที่ตัวแปรที่ใช้สำหรับเก็บค่าผลลัพธ์ ด้วยสมการ (Expression) ที่ใช้คำนวณค่าตัวแปรนั้น
4. วิธี Replace Temp with Query เป็นการนำสมการที่ใช้คำนวณค่าผลลัพธ์ไปสร้างเป็นเมทธอดใหม่ และแทนที่ที่อ้างอิงถึงตัวแปรที่ใช้เก็บค่าผลลัพธ์ของสมการนั้น ด้วยการเรียกใช้เมทธอดใหม่แทน
5. วิธี Introduce Explaining Variable เป็นการนำผลลัพธ์ของสมการ หรือผลลัพธ์บางส่วนของสมการไปเก็บในตัวแปร โดยตั้งชื่อตัวแปรนั้นให้สื่อความหมาย เพื่อช่วยให้เข้าใจได้ง่ายขึ้น
6. วิธี Split Temporary Variable เป็นการสร้างตัวแปรเพิ่ม สำหรับทุกที่ที่มีการกำหนดค่า ตัวแปรใดๆ
7. วิธี Remove Assignments to Parameters เป็นการใช้ตัวแปรมารับค่าจากค่าพารา-มิเตอร์ที่รับเข้ามา แทนการใช้ค่าในพารามิเตอร์โดยตรง
8. วิธี Replace Method with Method Object เป็นการสร้างเมทธอดใหม่แทนที่เมทธอดเดิม ซึ่งอยู่ภายในอ็อบเจกต์เดียวกัน
9. วิธี Substitute Algorithm เป็นการแทนที่โค้ดบางส่วนภายในเมทธอดด้วยวิธีการใหม่

Moving Feature between Objects เป็นวิธีรีแฟคทอริงสำหรับการย้ายคุณสมบัติระหว่างอ็อบเจกต์ ดังนี้

10. วิธี Move Method เป็นการย้ายเมทธอดจากคลาสหนึ่งไปยังอีกคลาสหนึ่ง แล้วทำการมอบหมายให้เมทธอดในคลาสเดิมทำหน้าที่ง่ายๆ อย่างอื่น หรือลบเมทธอดในคลาสเดิมทิ้ง
11. วิธี Move Attribute เป็นการย้ายคุณลักษณะจากคลาสหนึ่ง ไปยังอีกคลาสหนึ่ง
12. วิธี Extract Class เป็นการย้ายคุณลักษณะ หรือเมทธอดของคลาสเดิมที่มีความสัมพันธ์กันไปสร้างเป็นคลาสใหม่
13. วิธี Inline Class เป็นการย้ายคุณลักษณะ และเมทธอดทั้งหมดจากคลาสหนึ่งไปไว้ในอีกคลาสหนึ่ง และลบคลาสเดิมทิ้ง
14. วิธี Hide Delegate เป็นการสร้างเมทธอดใหม่บนคลาส เพื่อซ่อนการดีลีเกชัน
15. วิธี Remove Middle Man เป็นการให้ไคลเอนท์เรียกใช้การดีลีเกชันได้โดยตรง
16. วิธี Introduce Foreign Method เป็นการสร้างเมทธอดในไคลเอนท์ ด้วยอินสแตนท์ของเซิร์ฟเวอร์ที่อากรูเมนท์ (Argument) แรกของคลาสนั้น
17. วิธี Introduce Local Extension เป็นการสร้างคลาสใหม่ซึ่งภายในมีเมทธอดที่ทำการขยายจากคลาสลูก

Organizing Data เป็นวิธีรีแฟคทอริงสำหรับการจัดการข้อมูล ดังนี้

18. วิธี Self Encapsulate Field เป็นการสร้างเอคเซสเซอร์เมทธอดสำหรับข้อมูล
19. วิธี Replace Data Value with Object เป็นการสร้างอ็อบเจกต์ใหม่มาแทนที่คุณลักษณะของอ็อบเจกต์เดิม
20. วิธี Change Value to Reference เป็นการทำให้ค่าของอ็อบเจกต์ไปอ้างถึงอ็อบเจกต์แทน
21. วิธี Change Reference to Value เป็นการทำให้การอ้างถึงอ็อบเจกต์ไปเป็นค่าของอ็อบเจกต์แทน
22. วิธี Replace Array with Object เป็นการแทนที่อาร์เรย์ด้วยอ็อบเจกต์
23. วิธี Duplicate Observed Data เป็นการคัดลอกข้อมูลจากไปยังอ็อบเจกต์ โดเมน (Domain object) และสร้างออบเซิร์บเวอร์ (Observer) เพื่อซิงโครไนส์ (Synchronize) ข้อมูลทั้ง 2 โดเมน

24. วิธี Change Unidirectional Association to Bi-directional เป็นการเปลี่ยนความสัมพันธ์ระหว่างอ็อบเจกต์ให้อ้างอิงได้ทั้ง 2 อ็อบเจกต์
25. วิธี Change Bi-directional Association to Unidirectional เป็นการเปลี่ยนความสัมพันธ์ระหว่างอ็อบเจกต์ให้อ้างอิงได้เพียงอ็อบเจกต์เดียว
26. วิธี Replace Magic Number with Symbolic Constant เป็นการสร้างตัวแปรขั่วคราวสำหรับค่าคงที่ และตั้งชื่อให้สื่อความหมาย
27. วิธี Encapsulate Field เป็นการเปลี่ยนขอบเขตของคุณลักษณะให้เป็นไพรเวท และสร้างเอคเซสเซอร์เมทธอด
28. วิธี Encapsulate Collection เป็นการสร้างเมทธอดเพิ่ม (Add) และลบ (Remove) ค่าตัวแปรอินสแตนท์ที่เก็บข้อมูลเป็นเซต
29. วิธี Replace Record with Data Class เป็นการสร้างคลาสข้อมูล (Data class) สำหรับเก็บข้อมูลที่เป็นเรคอร์ด (Record)
30. วิธี Replace Type Code with Class เป็นการแทนที่ประเภทของโค้ด (Type code) ด้วยการสร้างคลาสใหม่
31. วิธี Replace Type Code with Subclasses เป็นการแทนที่ประเภทของโค้ด ด้วยการสร้างคลาสลูก
32. วิธี Replace Type Code with State/Strategy เป็นการแทนที่ประเภทของโค้ด (Type code) ด้วยสเตทอ็อบเจกต์ (State object)
33. วิธี Replace Subclass with Fields เป็นการแทนที่เมทธอดของคลาสลูกด้วยคุณลักษณะของคลาสแม่ และลบคลาสลูกทิ้ง

Simplifying Conditional Expressions เป็นวิธีรีแฟคทอริงสำหรับเปลี่ยนแปลงส่วนเงื่อนไข ดังนี้

34. วิธี Decompose Conditional เป็นการย้ายโค้ดบางส่วนที่อยู่หลังเงื่อนไข ไปสร้างเป็นเมทธอดใหม่
35. วิธี Consolidate Conditional Expression เป็นการรวมโค้ดที่อยู่หลังเงื่อนไข หลายๆ เงื่อนไข ที่ให้ผลลัพธ์เหมือนกันเป็นเงื่อนไขเดียว โดยนำไปสร้างเป็นเมทธอดใหม่ และเรียกใช้เมธอดนั้นหลังเงื่อนไขแทน

36. วิธี Consolidate Duplicate Conditional Fragments เป็นการย้ายโค้ดส่วนที่ซ้ำกันที่มีอยู่ทุกเงื่อนไขออกจากส่วนของเงื่อนไขนั้น

37. วิธี Remove Control Flag เป็นการใช้คำสั่งเบรก (Break) หรือรีเทอร์น (Return) แทนการใช้แฟลก (Flag) ในส่วนเงื่อนไข

38. วิธี Replace Nested Conditional with Guard Clauses เป็นการใช้การ์ดคอร์ส (Guard clause) แทนการใช้เงื่อนไขภายในเงื่อนไข

39. วิธี Replace Conditional with Polymorphism เป็นการแทนที่โค้ดส่วนที่เป็นเงื่อนไขด้วยการสร้างเมทธอดโอเวอร์ไรด์ในคลาสลูก และทำให้เมทธอดเดิมเป็นแอบสเตรค (Abstract)

40. วิธี Introduce Null Objects เป็นการใช้ค่าว่างด้วยอ็อบเจกต์ว่างแทน

41. วิธี Introduce Assertion เป็นการตรวจสอบเงื่อนไขที่ตั้งไว้ให้เป็นจริงด้วยการใช้เอสเสอร์-ชัน (Assertion)

Making Method Calls Simpler เป็นวิธีรีแฟคทอริงสำหรับทำให้เมทธอดเรียกใช้ได้ง่าย ดังนี้

42. วิธี Rename Method เป็นการเปลี่ยนชื่อเมทธอดเดิม เพื่อให้เข้าใจได้ง่ายขึ้น

43. วิธี Add Parameter เป็นการเพิ่มพารามิเตอร์เข้าไปในอ็อบเจกต์ เพื่อใช้ส่งผ่านข้อมูลระหว่างเมทธอด

44. วิธี Remove Parameter เป็นการลบพารามิเตอร์ที่ส่งผ่านข้อมูลในเมทธอดออก

45. วิธี Separate Query from Modifier เป็นการสร้างเมทธอดขึ้นใหม่ 2 เมทธอดคือ ใช้เพื่อดึงข้อมูล และใช้เพื่อแก้ไขข้อมูล

46. วิธี Parameterize Method เป็นการสร้างเมทธอดใหม่ เพื่อใช้พารามิเตอร์แทนค่าที่ต่างกัน

47. วิธี Replace Parameter with Explicit Methods เป็นการแบ่งเมทธอดมาสร้างใหม่สำหรับแต่ละค่าของพารามิเตอร์

48. วิธี Preserve Whole Object เป็นการส่งค่าอ็อบเจกต์ไปยังเมทธอด แทนการส่งค่าหลายๆ พารามิเตอร์

49. วิธี Replace Parameter with Explicit Methods เป็นการสร้างเมทธอดขึ้นใหม่สำหรับใช้แทนพารามิเตอร์ที่ส่งไปให้เมทธอดและทำให้เมทธอดที่เรียกใช้พารามิเตอร์นั้น เรียกใช้เมทธอดที่สร้างใหม่นั้นแทน
50. วิธี Introduce Parameter Object เป็นการแทนที่พารามิเตอร์ด้วยอ็อบเจกต์
51. วิธี Remove Setting Method เป็นการลบเมทธอดทำหน้าที่กำหนดค่าให้ข้อมูลออก
52. วิธี Hide Method เป็นการเปลี่ยนขอบเขตของเมทธอดเปลี่ยนเป็นไพรเวท
53. วิธี Replace Constructor with Factory Method เป็นการแทนที่คอนสตรัคเตอร์ด้วยเฟคทอรีเมทธอด (Factory method)
54. วิธี Encapsulate Downcast เป็นการทำดาวน์แคส (Downcast) ภายในเมทธอด
55. วิธี Replace Error Code with Exception เป็นการใช้ข้อยกเว้น (Exception)
56. วิธี Replace Exception with Test เป็นการทดสอบเงื่อนไขก่อนแทนการใช้ข้อยกเว้น

Dealing with Generalization เป็นวิธีรีแฟคทอริงสำหรับความสัมพันธ์เป็นลำดับชั้น ดังนี้

57. วิธี Pull Up Field เป็นการย้ายคุณลักษณะจากคลาสลูก ไปยังคลาสแม่
58. วิธี Pull Up Method เป็นการย้ายเมทธอดจากคลาสลูกไปยังคลาสแม่
59. วิธี Pull Up Constructor Body เป็นการสร้างคอนสตรัคเตอร์ในคลาสแม่ เพื่อให้คลาสลูกเรียกใช้
60. วิธี Push Down Method เป็นการย้ายเมทธอดจากคลาสแม่ไปยังคลาสลูก
61. วิธี Push Down Field เป็นการย้ายคุณลักษณะจากคลาสแม่ไปยังคลาสลูก
62. วิธี Extract Subclass เป็นการสร้างคลาสลูกขึ้นใหม่ เพื่อใช้แทนบางฟังก์ชันในคลาสแม่
63. วิธี Extract Superclass เป็นการสร้างคลาสใหม่ขึ้นมาเป็นคลาสแม่ และย้ายเมทธอดที่ใช้ร่วมกันไปที่คลาสที่สร้างขึ้นมาใหม่นั้น
64. วิธี Extract Interface เป็นการสร้างอินเตอร์เฟสขึ้นใหม่
65. วิธี Collapse Hierarchy เป็นการรวมคลาสแม่ และคลาสลูกเข้าด้วยกัน
66. วิธี Form Template Method เป็นการสร้างเมทธอดใหม่ในคลาสแม่ แทนการใช้เมทธอด ในคลาสลูก 2 คลาสที่มีซิกเนเจอร์ และฟังก์ชันเหมือนกัน

67. วิธี Replace Inheritance with Delegation เป็นการสร้างคุณลักษณะใหม่สำหรับคลาสแม่ และเปลี่ยนเมทธอดโดยการดีลีเกชันไปยังคลาสแม่ และทำการลบคลาสลูกนั้นทิ้ง

68. วิธี Replace Delegation with Inheritance เป็นการสร้างคลาสลูกแทนการดีลีเกชัน

Big Refactorings เป็นวิธีการรีแฟคทอริงสำหรับการเปลี่ยนแปลงหลายๆ อย่าง ดังนี้

69. วิธี Tease Apart Inheritance เป็นการกำหนดโครงสร้างการสืบทอดคุณสมบัติ 2 โครงสร้างและใช้การดีลีเกชันระหว่างคลาสแม่ เพื่อคลาสหนึ่งเรียกใช้อีกคลาสหนึ่งได้

70. วิธี Convert Procedure Design to Objects เป็นการเปลี่ยนข้อมูลไปเป็นออบเจ็คและย้ายบางฟังก์ชันไปยังออบเจ็คนั้น

71. วิธี Separate Domain from Presentation เป็นการแยกคลาสที่ทำงานโดเมนอื่นออกจากคลาสที่เป็นส่วนติดต่อกับผู้ใช้งาน

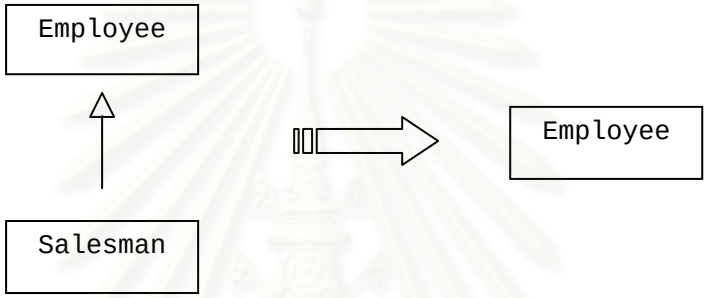
72. วิธี Extract Hierarchy เป็นการกำหนดโครงสร้างการสืบทอดคุณสมบัติของคลาสใดๆ ซึ่งคลาสลูกนั้นใช้สำหรับกรณีพิเศษ

ภาคผนวก ข
ขั้นตอนการทำวิธีรีแฟคทอริงที่ใช้ในงานวิจัย

ในภาคผนวกนี้จะแสดงรายละเอียดขั้นตอนการทำวิธีรีแฟคทอริงที่กล่าวถึงในงานวิจัยนี้ 20 วิธี ซึ่งมีรายละเอียดดังนี้

ข.1 Collapse Hierarchy

เป็นการรวมคลาสแม่ (Superclass) และคลาสลูก (Subclass) เข้าด้วยกัน แสดงตัวอย่างดังรูปที่ ข.1



รูปที่ ข.1 แผนภาพคลาสแสดงตัวอย่างการทำวิธี Collapse Hierarchy

โดยทำตามขั้นตอนดังนี้

1. เลือกคลาสแม่ หรือคลาสลูกที่จะถูกลบทิ้ง
2. ใช้วิธี Pull Up Field และวิธี Pull Up Method ถ้าเป็นการย้ายเมทธอดและคุณลักษณะจากคลาสลูกไปยังคลาสแม่ หรือวิธี Pull Down Method และวิธี Push Down Field ถ้าเป็นการย้ายเมทธอดและคุณลักษณะจากคลาสแม่ไปยังคลาสลูก
3. คอมไพล์ (Compile) และทดสอบ (Test) เมทธอดและคุณลักษณะที่ถูกย้าย
4. แก้ไขการอ้างไปยังคลาสที่จะถูกลบ ซึ่งจะกระทบกับการประกาศตัวแปร (Variable declaration) ชนิดของพารามิเตอร์ (Parameter types) และคอนสตรัคเตอร์ (Constructor)
5. ลบคลาสแม่ หรือคลาสลูกที่เลือกไว้
6. คอมไพล์ และทดสอบ

ข.2 Decompose Conditional

เป็นการย้ายโค้ดบางส่วนที่อยู่หลังเงื่อนไข (if-then-else) ไปสร้างเป็นเมทธอดใหม่ แสดงตัวอย่างดังรูปที่ ข.2

```
If(date.before(SUMMER_START)||date.after(SUMMER_END))
charge = quantity * _winterRate + _winterServiceCharge;
else    charge = quantity * _summerRate;
```

รูปที่ ข.2 ตัวอย่างโค้ดก่อนการทำวิธี Decompose Conditional

โดยทำตามขั้นตอนดังนี้

1. ย้ายโค้ดที่อยู่ในส่วนเงื่อนไข (if) ไปสร้างเป็นเมทธอดใหม่ ภายในเมทธอดเดิม
2. ย้ายโค้ดที่อยู่ในส่วนเดน (then) และส่วนเอลส์ (else) ไปสร้างเป็นเมทธอดใหม่ ภายในเมทธอดเดิม

```
If(notSummer(date))
        charge = winterCharge(quantity);
else    charge = summerCharge(quantity);

private boolean notSummer(Date date){
      return date.before(SUMMER_START)||date.after(SUMMER_END);             }

private double summerCharge(int quantity){
      return quantity * _summerRate;                                        }

private double winterCharge(int quantity){
      return quantity * _winterRate + _winterServiceCharge;                 }
```

รูปที่ ข.3 ตัวอย่างโค้ดหลังการทำวิธี Decompose Conditional

ข.3 Duplicate Observed Data

เป็นการคัดลอกข้อมูลจากไปยังอ็อบเจกต์ โดเมน และสร้างออบเซิร์บเวอร์ เพื่อซิงโครไนส์ข้อมูลทั้ง 2 โดเมน

```
public class IntervalWindow extends Frame…
     java.awt.TextField _startField;
     java.awt.TextField _endField;
     java.awt.TextField _lengthField;
     class SymFocus extends java.awt.event.FocusAdapter {
     public void focusLost(java.awt.event.FocusEvent event) {
        Object object = event.getSource();
        if(object == _startField)   StartField_FocusLost(event);
        else if (object == _endField)   EndField_FocusLost(event);
        else if (object == _lengthField) LengthField_FocusLost(event);          }
     void StartField_FocusLost(java.awt.event.FocusEvent event) {
        if(isNotInteger(_startField.getText())) _startField.setText("0");
        calculateLength();                                                      }
     void EndField_FocusLost(java.awt.event.FocusEvent event) {
        if(isNotInteger(_endField.getText())) _endField.setText("0");
        calculateLength();                                                      }
     void LengthField_FocusLost(java.awt.event.FocusEvent event) {
        if(isNotInteger(_lengthField.getText())) _lengthField.setText("0");
        calculateEnd();                                                         }
     void calculateLength() {
        try {
           int start = Integer.parseInt(_startField.getText());
           int end = Integer.parseInt(_endField.getText());
           int length = end – start;
           _lengthField.setText(String.valueOf(length));
        } catch (NumberFormatException e) {
        throw new RuntimeException ("Unexpected Number Format Error");          }
}
```

รูปที่ ข.4 ตัวอย่างโค้ดก่อนการทำวิธี Duplicate Observed Data

```
          void calculateEnd() {
               try {
                    int start = Integer.parseInt(_startField.getText());
                    int length = Integer.parseInt(_lengthField.getText());
                    int end = start + length;
                    _endField.setText(String.valueOf(end));
               } catch (NumberFormatException e) {
                    throw new RuntimeException (“Unexpected Number Format Error”);
               }
          }
     }
…
```

รูปที่ ข.4 (ต่อ) ตัวอย่างโค้ดก่อนการทำวิธี Duplicate Observed Data

โดยทำตามขั้นตอนดังนี้

1. สร้างคลาสพรีเซนเตชั่น (Presentation class) เป็นออบเซิร์บเวอร์ของคลาสโดเมน (Domain class) โดยมีเงื่อนไขดังนี้
   a. ถ้ายังไม่มีคลาสโดเมน ให้สร้างคลาสโดเมน
   b. ถ้ายังไม่มีลิงค์ (Link) จากคลาสพรีเซนเตชั่นไปยังคลาสโดเมน ให้นำคลาสโดเมนไว้ในฟิวส์ (Field) ของคลาสพรีเซนเตชั่น

```
class Interval extends Observable {…}
public class IntervalWindow extends Frame implements Observer
     private Interval _subject;
     public IntervalWindow() {
          …
          _subject = new Interval();
          _subject.addObserver(this);
          update(_subject, null);
          …
     }
     public void update(Observable observed, Object arg) {     }
}
```

2. ใช้วิธี Self Encapsulate Field กับโดเมนข้อมูลภายในคลาสของส่วนที่ติดต่อกับผู้ใช้งาน

```
          String getEnd() {
               return _endField.getText();
          }
          void setEnd(String arg) {
               _endField.setText(arg);
          }
          void calculateLength() {
               try {
                    int start = Integer.parseInt(_startField.getText());
                    int end = Integer.parseInt(getEnd());
                    int length = end – start;
                    _lengthField.setText(String.valueOf(length));
               } catch (NumberFormatException e) {
                    throw new RuntimeException (“Unexpected Number Format Error”);
               }
          }
          void calculateEnd() {
               try {
                    int start = Integer.parseInt(_startField.getText());
                    int length = Integer.parseInt(_lengthField.getText());
                    int end = start + length;
                    setEnd(String.valueOf(end));
               } catch (NumberFormatException e) {
                    throw new RuntimeException (“Unexpected Number Format Error”);
               }
          }
          void EndField_FocusLost(java.awt.event.FocusEvent event) {
               if(isNotInteger(getEnd()))
                    setEnd(“0”);
               calculateLength();
          }
```

3. คอมไพล์ และทดสอบ
4. เพิ่มการเรียกใช้เมทธอดที่กำหนดคุณลักษณะ (Setting method) ในการรองรับเหตุการณ์ (Event handler) เพื่อปรับปรุงคอมโพเนนท์ (Component) ด้วยค่าปัจจุบันที่ใช้อยู่ในขณะนั้น

```
void EndField_FocusLost(java.awt.event.FocusEvent event) {
     setEnd(_endField.getText());
     if(isNotInteger(getEnd()))     setEnd("0");
     calculateLength();                                    }
```

5. คอมไพล์ และทดสอบ
6. กำหนดข้อมูล และเอคเซสเซอร์เมทธอดในคลาสโดเมน

```
class Interval…
     private String _end = "0";
…
```

7. ให้เอคเซสเซอร์กำหนดข้อมูลในฟิวส์โดเมน (Domain field)

```
class Inteval…
     String getEnd() { return _end; }
     void setEnd (String arg) {
          _end = arg;
          setChanged();
          notifyObservers();
     }
…
```

8. แก้ไขเมทธอดที่ใช้ปรับปรุงค่าของออบเซิร์บเวอร์เพื่อคัดลอกข้อมูลจากฟิวส์โดเมนไปยังส่วนที่ติดต่อกับผู้ใช้งาน

```
class IntervalWindow…
     String getEnd() { return _subject.getEnd(); }
     void setEnd(String arg) { _subject.setEnd(arg); }
     public void update(Observable observed, Object arg) {
          _endField.setText(_subject.getEnd());
     }
…
```

9. คอมไพล์และทดสอบ

```
class Interval extends Observable{}
public class IntervalWindow extends Frame…
     java.awt.TextField _startField;
     java.awt.TextField _endField;
     java.awt.TextField _lengthField;
     class SymFocus extends java.awt.event.FocusAdapter{
     public void focusLost(java.awt.event.FocusEvent event){
        Object object = even.getSource();
        If(object == _startField)          StartField_FocusLost(event);
        else if(object == _endField)       EndField_FocusLost(event);
        else if(object == _lengthField)  LengthField_FocusLost(event);          }
     }
     void StartField_FocusLost(java.awt.event.FocusEvent event){
       if(isNoInteger(_startField.getText()))       _startField.setText("0");
       calculateLength();                                                    }
     void EndField_FocusLost(java.awt.event.FocusEvent event){
       if(isNoInteger(_endField.getText()))  _endField.setText("0");
       calculateLength();                                                    }
     void LengthField_FocusLost(java.awt.event.FocusEvent event){
       if(isNoInteger(_lengthField.getText()))       _lengthField.setText("0");
       calculateLength();                                                    }
     void calculateLength(){
       try{
        int start = Integer.parseInt(_startField.getText());
        int end = Integer.parseInt(_endField.getText());
        int length = end – start;
        _lengthField.setText(String.valueOf(length));                         }
```

รูปที่ ข.5 ตัวอย่างโค้ดหลังการทำวิธี Duplicate Observed Data

```
        catch (NumberFormatException e) {
        throw new RuntimeException("Unexpected Number Format Error");           }
        }
        void calculatedEnd(){
                try{
        int start = Integer.parseInt(_startField.getText());
        int length = Integer.parseInt(_lengthField.getText());
        int end = start + length;
        _endField.setText(String.valueOf(end));
                }       catch (NumberFormatException e){
        throw new RuntimeException("Unexpected Number Format Error");
               }
        }
```

รูปที่ ข.5 (ต่อ) ตัวอย่างโค้ดหลังการทำวิธี Duplicate Observed Data

ข.4 Extract Class

เป็นการสร้างคลาสใหม่ และย้ายคุณลักษณะและเมทธอดจากคลาสเดิมไปยังคลาสใหม่ โดยทำตามขั้นตอนดังนี้

1. ตัดสินใจเลือกว่าจะแบ่งความรับผิดชอบของคลาสอย่างไร

```
class Person…
     public String getName() {  return _name; }
     public String getTelephoneNumber() {
          return ("(" + _officeAreaCode  + ")" + _officeNumber);
     }
     String getOfficeAreaCode() { return _officeAreaCode; }
     void setOfficeAreaCode() { _OfficeAreaCode = arg; }
     String getOfficeNumber() { return _officeNumber;  }
     void setOfficeNumber() { _officeNumber = arg;      }
     private String _name;
     private String _officeAreaCode;
     private String _officeNumber;
```

รูปที่ ข.6 ตัวอย่างโค้ดก่อนการทำวิธี Extract Class

```
class TelephoneNumber {
     String getAreaCode() { return _areaCode;    }
     void setAreaCode() { _areaCode = arg;  }
     String getNumber() { return _number;   }
     void setNumber() { _number = arg;      }
     public String getTelephoneNumber() {
          return ("(" + _areaCode  + ")" + _number);
     }
     private String _areaCode;
     private String _number;
}
class Person {
     public String getName() { return _name;     }
     public String getTelephoneNumber() {
          return _officeTelephone.getTelephone();
     }
     TelephoneNumber getOfficeTelephone() { return _officeTelephone;   }
     private String _name;
     private TelephoneNumber _officeTelephone = new TelephoneNumber();
}
```

รูปที่ ข.7 ตัวอย่างโค้ดหลังการทำวิธี Extract Class

2. สร้างคลาสใหม่ เพื่อแบ่งความรับผิดชอบของคลาสนั้นออกไป และถ้าความรับผิดชอบของคลาสเดิมไม่สัมพันธ์กับชื่อคลาส ควรจะเปลี่ยนชื่อคลาสเดิมนั้น
3. สร้างลิงค์จากคลาสเดิมไปยังคลาสใหม่

4. ประยุกต์ใช้วิธี Move Field กับทุกฟิวส์ที่ต้องการย้าย
5. คอมไพล์ และทดสอบหลังจากการย้ายทุกครั้ง
6. ประยุกต์ใช้วิธี Move Method เพื่อย้ายเมทธอดจากคลาสเดิมไปยังคลาสใหม่
7. คอมไพล์ และทดสอบหลังจากการย้ายทุกครั้ง

ข.5 Extract Interface

เป็นการย้ายบางส่วนไปสร้างเป็นอินเตอร์เฟส

```
double charge(Employee emp, int days) {
     int base = emp.getRate() * days;
     if(emp.hasSpecialSkill())   return base * 1.05;
     else return base;                                                    }
```

รูปที่ ข.8 ตัวอย่างโค้ดก่อนการทำวิธี Extract Interface

โดยทำตามขั้นตอนดังนี้

1. สร้างอินเตอร์เฟสใหม่
2. ประกาศเมทธอดหรือคุณลักษณะที่ใช้ร่วมกันในอินเตอร์เฟส

```
interface Billable {
     public int getRate();
     public boolean hasSpecialSkill();
}
```

3. ประกาศคลาสที่สัมพันธ์กันตามที่สร้างในอินเตอร์เฟส

```
class Employee implements Billable…
```

4. ปรับปรุงการประกาศประเภทไคลเอนท์ที่ใช้อินเตอร์เฟส

```
interface Billable {
     public int getRate();
     public boolean hasSpecialSkill();
}
class Employee implements Billable {
     double charge(Billable emp, int days) {
          int base = emp.getRate() * days;
          if(emp.hasSpecialSkill())   return base * 1.05;
          else return base;                                               }
}
```

รูปที่ ข.9 ตัวอย่างโค้ดหลังการทำวิธี Extract Interface

ข.6 Extract Method

เป็นการย้ายบางส่วนของโค้ดมาสร้างเป็นเมทธอดใหม่ และตั้งชื่อเมทธอดนั้นตามจุดประสงค์ของการสร้างเมทธอดนั้น

```
void printOwing(double amount){
     printBaner();
     System.out.printIn(“name :”+ _name);
     System.out.println(“amount “+ amount);
}
```

รูปที่ ข.10 ตัวอย่างโค้ดก่อนการทำวิธี Extract Method

```
void printOwing(double amount){
     printBaner();
     printDetails(amount);
}
void printDetails(double amount){
     System.out.printIn(“name :”+ _name);
     System.out.println(“amount “+ amount);
}
```

รูปที่ ข.11 ตัวอย่างโค้ดหลังการทำวิธี Extract Method

โดยทำตามขั้นตอนดังนี้

1. สร้างเมทธอดใหม่ และตั้งชื่อตามวัตถุประสงค์ของเมทธอดใหม่นั้น
2. คัดลอกโค้ดที่จะถูกย้ายจากเมทธอดเดิม ไปยังเมทธอดใหม่ที่สร้างขึ้น
3. ตรวจสอบการเรียกใช้ตัวแปรภายในโค้ดที่ถูกย้ายว่า ถ้าตัวแปรนั้นเรียกใช้ได้เฉพาะในเมทธอดเดิม ต้องมีการส่งผ่านค่าพารามิเตอร์ไปยังเมทธอดใหม่ด้วย
4. ตรวจสอบภายในโค้ดที่ถูกย้ายว่า ถ้ามีการใช้ตัวแปรชั่วคราวเฉพาะภายในโค้ดที่ถูกย้าย ต้องประกาศค่าตัวแปรชั่วคราวนั้นด้วย
5. คอมไพล์เมื่อจัดการกับตัวแปรโลคอลทั้งหมดแล้ว
6. แทนที่โค้ดที่ถูกย้ายด้วยการเรียกใช้เมทธอดใหม่ที่สร้างขึ้น
7. คอมไพล์ และทดสอบ

ข.7 Extract Subclass

เป็นการสร้างคลาสลูกสำหรับบางส่วนของคุณสมบัติของคลาสเดิม เพราะคุณสมบัตินั้นถูกใช้เพียงบางกรณีของตัวแปรอินสเตนท์ของคลาสนั้น

```
class JobItem {
  public JobItem(int unitPrice, int quantity, boolean isLabor, Employee employee) {
          _unitPrice = unitPrice;
          _quantity = quantity;
          _isLabor = isLabor;
          _employee = employee;
     }
     public int getTotalPrice() { return getUnitPrice() * _quantity; }
     public int getUnitPrice() {
          return(_isLabor)? _employee.getRate() : _unitPrice;
     }
     public int getQuantity() { return _quantity; }
     public Employee getEmployee() { return  _employee; }
     private int _unitPrice;
     private int _quantity;
     private Employee _employee;
     private boolean _isLabor;
}
class Employee {
     public Employee (int rate) {
          _rate = rate;
     }
     public int getRate() { return _rate; }
     private int _rate;
}
```

รูปที่ ข.12 ตัวอย่างโค้ดก่อนการทำวิธี Extract Subclass

จากตัวอย่างต้องการแยกคลาส LaborItem มาสร้างเป็นคลาสลูกของคลาส JobItem โดยทำตามขั้นตอนดังนี้

1. สร้างคลาสลูกใหม่

```
class LaborItem extend Employee { ... }
```

2. สร้างคอนสตรัคเตอร์สำหรับคลาสลูกที่สร้างขึ้น
   a. ในกรณีทั่วไป คัดลอกอากรูเมนต์ของคลาสแม่และเรียกใช้คอนสตรัคเตอร์ของคลาสแม่ด้วย super()

```
public LaborItem (int unitPrice, int quantity, boolean isLabor, Employee employee){
     super(unitPrice, quantity, isLabor, employee);                          }
```

   b. แต่ถ้าไม่ต้องการให้ไคลเอนท์เรียกใช้งานได้ ให้ประยุกต์ใช้วิธี Replace Constructor with Factory Method
3. ตรวจสอบการเรียกใช้คอนสตรัคเตอร์ของคลาสแม่ ถ้าสามารถเรียกใช้คลาสลูกได้ ให้แทนที่ด้วยการเรียกใช้คอนสตรัคเตอร์ของคลาสลูกแทน

```
JobItem j1 = new LaborItem(0, 5, true, kent)
```

   a. ถ้าคอนสตรัคเตอร์ของคลาสลูกต้องการอากรูเมนต์ต่างกัน และถ้าบางพารามิเตอร์ในคอนสตรัคเตอร์ของคลาสแม่ไม่ได้ใช้ให้ใช้วิธี Rename Method

```
class JobItem {
 protected JobItem(int unitPrice, int quantity, boolean isLabor, Employee employee)
 {
          _unitPrice = unitPrice;
          _quantity = quantity;
          _isLabor = isLabor;
          _employee = employee;                                                  }
 public JobItem(int unitPrice, int quantity) {
          this(unitPrice, quantity, false, null)                                 }
…
}
class LaborItem extend Employee { ...
 public LaborItem (int quantity, Employee employee) {
          super(0, quantity, true, employee);                                    }
…
}
```

   b. แต่ถ้าไม่สามารถประกาศใช้คลาสแม่ได้โดยตรงให้ประกาศเป็นแอบสเตรค
4. ประยุกต์ใช้วิธี Push Down Method และ Push Down Field เพื่อย้ายคุณสมบัติต่างๆ ไปยังคลาสลูก
   a. ขั้นตอนนี้จะต่างกับ Extract Class เพราะควรจะย้ายเมทธอดก่อน แล้วย้ายคุณลักษณะตามมา
   b. เมื่อพับบลิคเมทธอดถูกย้าย ต้องแก้ไขตัวแปรที่เรียกใช้ และชนิดของพารามิเตอร์เพื่อให้เรียกใช้เมทธอดนั้นได้

```
class LaborItem…
     public LaborItem (int quantity, Employee employee) {
          super(0, quantity, true, employee);
          _employee = employee;
     }
     public Employee getEmpolyee() { return _employee;      }
     private Employee _employee;
…
class JobItem…
     protected JobItem(int unitPrice, int quantity, boolean isLabor) {
          _unitPrice = unitPrice;
          _quantity = quantity;
          _isLabor = isLabor;
     }
…
```

5. หาคุณลักษณะที่บอกถึงความแตกต่างของโครงสร้างคลาสที่สืบทอด (ส่วนใหญ่จะเป็นบูลีน (Boolean) หรือชนิดของโค้ด) แล้วกำจัดด้วยวิธี Self Encapsulate Field และเปลี่ยนเมทธอดที่ให้ค่าคุณลักษณะนั้นด้วยการใช้โพลีมอร์ฟิซึม ด้วยวิธี Replace Condition with Polymorphism

```
class LaborItem…
 protected boolean isLabor() { return true;     }
 public int getUnitPrice() { return _employee.getRate(); }
…
class JobItem…
 protected boolean isLabor() { return false; }
 public int getUnitPrice() { return _unitPrice; }
…
```

6. คอมไพล์ และทดสอบหลังการย้ายแต่ละครั้ง

```
class LaborItem extends JobItem {
  public LaborItem(int unitPrice, int quantity, boolean isLabor, Employee employee)
  { super(unitPrice, quantity, isLabor, employee);     }
  public LaborItem(int quantity, Employee employee){
       super(0, quantity, true);
       _employee = employee;
  }
  public Employee getEmployee() { return  _employee;     }
  protected boolean _isLabor() {  return true;     }
  public int getUnitPrice() { return _employee.getRate();     }
  private Employee _employee;
}

class JobItem {
  protected JobItem(int unitPrice, int quantity) {
       _unitPrice = unitPrice;
       _quantity = quantity;
  }
  public JobItem(int unitPrice, int quantity) {
          this(unitPrice, quantity, false, null);     }
  public int getTotalPrice() { return getUnitPrice() * _quantity;     }
  public int getUnitPrice() {  return _unitPrice;     }
     public int getQuantity() {return _quantity;     }
     protected boolean _isLabor() { return false;     }
     private int _unitPrice;
     private int _quantity;
}

class Employee {
     public Employee (int rate) { _rate = rate;  }
     public int getRate() { return _rate;        }
     private int _rate;
}
```

รูปที่ ข.13 ตัวอย่างโค้ดหลังการทำวิธี Extract Subclass

ข.8 Inline Class

เป็นการย้ายคุณสมบัติทั้งหมดไปยังคลาสอื่น และลบคลาสเดิมทิ้ง โดยทำตามขั้นตอนดังนี้

1. ประยุกต์ใช้วิธี Move Method และ Move Filed จากคลาสเดิมไป ยังคลาสที่ต้องการย้าย
2. คอมไพล์และทดสอบ

```
class Person {
     public String getName() { return _name;  }
     public String getTelephoneNumber() {
          return _officeTelephone.getTelephone();
     }
     TelephoneNumber getOfficeTelephone() {
          return _officeTelephone;
     }
     private String _name;
     private TelephoneNumber _officeTelephone = new TelephoneNumber();
}
class TelephoneNumber {
     String getAreaCode() { return _areaCode;  }
     void setAreaCode() { _areaCode = arg;     }
     String getNumber() { return _number;      }
     void setNumber() { _number = arg;         }
     public String getTelephoneNumber() {
          return (“(“ + _areaCode  + “)” + _number);
     }
     private String _areaCode;
     private String _number;
}
```

รูปที่ ข.14 ตัวอย่างโค้ดก่อนการทำวิธี Inline Class

```
class Person…
     public String getName() { return _name;     }
     public String getTelephoneNumber() {
          return (“(“ + _officeAreaCode  + “)” + _officeNumber);
     }
     String getOfficeAreaCode() { return _officeAreaCode;     }
     void setOfficeAreaCode() { _OfficeAreaCode = arg;        }
     String getOfficeNumber() { return _officeNumber;         }
     void setOfficeNumber() {  _officeNumber = arg;           }
     private String _name;
     private String _officeAreaCode;
     private String _officeNumber;
```

รูปที่ ข.15 ตัวอย่างโค้ดหลังการทำวิธี Inline Class

ข.9 Introduce Null Object

เป็นการแทนที่ค่าว่าง (Null value) ด้วยอ็อบเจกต์ ว่าง (Null object)

```
class Site…
     Customer getCustomer() {  return _customer;   }
     Customer _customer;
     …
class Customer…
     public String getName() { … }
     public BillingPlan getPlan() { … }
     public PaymentHistory getHistory() { … }
     …
public class PaymentHistory…
     int getWeeksDelinguentInLastYear()
     …
```

รูปที่ ข.16 ตัวอย่างโค้ดก่อนการทำวิธี Introduce Null Object

```
…
Customer customer = site.getCustomer();
BillingPlan plan;
if(Customer == null) plan = BillingPlan.basic();
else plan = customer.getPlan();
…
String customerName;
if(Customer == null) customerName = “occupant”;
else customerName = customer.getName();
…
int weekDelinquent;
if(Customer == null) weeksDelnquent = 0;
else weeksDelnquent = customer.getHistory().getWeeksDelinquentInLastYear();
…
```

รูปที่ ข.16 (ต่อ) ตัวอย่างโค้ดก่อนการทำวิธี Introduce Null Object

โดยทำตามขั้นตอนดังนี้

1. สร้างคลาสลูกของคลาสเดิม เพื่อทำหน้าที่เป็นอ็อบเจกต์ ว่างของคลาส และสร้างเมทธอด isNull ในคลาสเดิมให้ส่งค่ากลับเป็นไม่จริง (False) และในคลาสว่าง (Null class) ให้ส่งค่ากลับเป็นจริง (True)

```
class NullCustomer extends Customer {
     public boolean isNull() { return true; }
}
class Customer…
     public boolean isNull() { return false;  }
…
```

   a. ถ้าไม่ต้องการแก้ไขภายในโค้ดของคลาสเดิม สามารถสร้างอินเตอร์เฟสทดสอบ (Testing interface) แทนการสร้างเมทธอด isNull ได้

```
interface Nullable { boolean isNull(); }
class Customer implements Nullable…
```

2. คอมไพล์
3. แทนที่ภายในโค้ดที่มีการอ้างถึงค่าว่าง เมื่อมีการเรียกใช้อ็อบเจกต์ ของคลาสเดิม ด้วยอ็อบเจกต์ ว่างแทน

```
…
class Customer…
     static Customer newNull() { return new NullCustomer();  }
…
class Site…
     Customer getCustomer() {
          return (_customer == null) ? Customer.newNull() : _customer;
     }
…
```

4. แทนที่ภายในโค้ดที่มีการอ้างถึงค่าว่าง เมื่อมีการเปรียบเทียบค่าของคลาสเดิมกับค่าว่าง ด้วยการเรียกใช้เมทธอด isNull แทน

```
…
Customer customer = site.getCustomer();
BillingPlan plan;
if(customer.isNull()) plan = BillingPlan.basic();
else plan = customer.getPlan();
…
String customerName = customer.getName();
if(customer.isNull()) customerName = “occupant”;
else customerName = customer.getName();
…
```

```
int weekDelinquent;
if(customer.isNull()) weeksDelnquent = 0;
else weeksDelnquent = customer.getHistory().getWeeksDelinquentInLastYear();
…
```

5. คอมไพล์ และทดสอบ
6. ตรวจสอบกรณีที่ไคลเอนต์เรียกใช้การทำงานของเมทธอดสำหรับถ้าเป็นค่าไม่ว่าง (if not null) และสำหรับกรณีการทำงานสำหรับค่าว่าง (if null) ให้โอเวอร์ไรด์ (Override) การทำงานนั้นในคลาสว่าง ด้วยการทำงานอื่นแทน

```
class NullCustomer…
     public void setPlan(BillingPlan arg) {}
     public String getName() { return “occupant”;   }
     public PaymentHistory getHistory() { return PaymentHistory.newNull();     }
…
class NullPaymentHistory extends PaymentHistory…
     int getWeeksDelinquentInLastYear() { return 0;     }
…
```

7. ลบการตรวจสอบเงื่อนไขที่เป็นค่าว่างออก และเปลี่ยนมาเรียกใช้เมทธอดที่โอเวอร์ไรด์ ในคลาสว่างแทน

```
class NullCustomer extends Customer {
     public boolean isNull() { return true;  }
     public void setPlan(BillingPlan arg) {}
     public String getName() { return “occupant”;  }
     public PaymentHistory getHistory() { return PaymentHistory.newNull();     }
}
class Customer…
     public boolean isNull() { return false; }
     static Customer newNull() { return new NullCustomer(); }
class Site…
     Customer getCustomer() {
          return (_customer == null) ? Customer.newNull() : _customer;
     }
…
class NullPaymentHistory extends PaymentHistory…
     int getWeeksDelinquentInLastYear() { return 0; }
…
```

```
     …
     Customer.setPlan(BillingPlan.special());
     String customerName = customer.getName();
     int weekDelinquent = customer.getHistory().getWeeksDelinquentInLastYear();
     …
```

รูปที่ ข.17 ตัวอย่างโค้ดหลังการทำวิธี Introduce Null Object

ข.10 Introduce Parameter Object

เป็นการแทนที่กลุ่มของพารามิเตอร์ด้วยอ็อบเจกต์

```
class Entry…
     Entry (double value, Data chargeDate) {
          _value = value;
          _chargeDate = chargeDate;
     }
     Date getDate() { return _chargeDate; }
     double getValue() { return _value;   }
     private Date _chargeDate;
     private double _value;
…
```

รูปที่ ข.18 ตัวอย่างโค้ดก่อนการทำวิธี Introduce Parameter Object

```
class Account…
     double getFlowBetween(Date start, Date end) {
          double result = 0;
          Enumeration e = _entires.elements();
          while(e.hasMoreElements()) {
               Entry each = (Entry) e.nextElement();
               if(each.getDate().equals(start) || each.getDate().equals(end) ||
                 (each.getDate().after(start) && each.getDate().before(end))) {
                    result += each.getValue();
               }
          }
          return result;
     }
     private Vector _entires = new Vector();
…
client code…
     double flow = anAccount.getFlowBetween(startDate, endDate);
```

รูปที่ ข.18 (ต่อ) ตัวอย่างโค้ดก่อนการทำวิธี Introduce Parameter Object

โดยทำตามขั้นตอนดังนี้

1. สร้างคลาสใหม่เพื่อใช้แทนกลุ่มของพารามิเตอร์ที่ต้องการแทนที่

```
class DateRange {
     DateRange( Date start, Date end) {
          _start = start;
          _end = end;
     }
     Date getStart() { return _start; }
     Date getEnd() { return _end;     }
     private final Date_start;
     private final Date _end;
}
```

2. คอมไพล์
3. ใช้วิธี Add Parameter สำหรับเพิ่มพารามิเตอร์ใหม่เข้าไปในซิกเนเจอร์ (Signature) ของเมทธอดที่ต้องการแทนที่ด้วยอ็อบเจกต์ ของคลาสใหม่ที่สร้างขึ้น และส่งค่าว่าง (null) สำหรับพารามิเตอร์ที่เพิ่มในการเรียกใช้เมทธอดนั้น

```
class Account…
     double getFlowBetween(Date start, Date end, DateRange range) {
          double result = 0;
          Enumeration e = _entires.elements();
          while(e.hasMoreElements()) {
               Entry each = (Entry) e.nextElement();
               if(each.getDate().equals(start) || each.getDate().equals(end) ||
                 (each.getDate().after(start) && each.getDate().before(end))) {
                    result += each.getValue();
               }
          }
          return result;
     }
     private Vector _entires = new Vector();
client code…
     double flow = anAccount.getFlowBetween(startDate, endDate, null);
```

4. ลบพารามิเตอร์เดิมในซิกเนเจอร์ของเมทธอดนั้น แก้ไขโค้ดภายในเมทธอดที่มีการเรียกใช้พารามิเตอร์ที่ลบไปด้วยการเรียกใช้อ็อบเจกต์ ที่ส่งเข้าไปใหม่แทน และแก้ไขโค้ดที่มีการเรียกใช้เมทธอดนั้น

```
class Account…
     double getFlowBetween(DateRange range) {
          double result = 0;
          Enumeration e = _entires.elements();
          while(e.hasMoreElements()) {
               Entry each = (Entry) e.nextElement();
```

```
                if(each.getDate().equals(range.getStart()) ||
                   each.getDate().equals(range.getEnd()) ||
                  (each.getDate().after(range.getStart()) &&
                   each.getDate().before(range.getEnd()))) {
                     result += each.getValue();
                }
           }
           return result;
      }
      private Vector _entires = new Vector();
…
client code…
      double flow = anAccount.getFlowBetween(new DateRange(startDate, endDate));
```

5. คอมไพล์ และทดสอบหลังจากลบพารามิเตอร์ออก

```
class Account…
      double getFlowBetween(DateRange range) {
           double result = 0;
           Enumeration e = _entires.elements();
           while(e.hasMoreElements()) {
                Entry each = (Entry) e.nextElement();
                if(range.include(each.getDate)) {
                     result += each.getValue();
                }
           }
           return result;
      }
      private Vector _entires = new Vector();
…
client code…
      double flow = anAccount.getFlowBetween(new DateRange(startDate, endDate));
…
class DateRange {
      DateRange( Date start, Date end) {
           _start = start;
           _end = end;
      }
      Date getStart() { return _start; }
      Date getEnd() {   return _end;   }
      boolean includes(Date arg) {
      return (arg.equals(_start) || arg.equals(_end) || (arg.after(_start)
              && arg.before(_end));                                        }
      private final Date_start;
      private final Date _end;
}
```

รูปที่ ข.19 ตัวอย่างโค้ดหลังการทำวิธี Introduce Parameter Object

6. เมื่อลบพารามิเตอร์ออกแล้ว ให้พิจารณาว่าเมทธอดใดที่สามารถย้ายไปยังพารามิเตอร์อ็อบเจกต์ ที่สร้างขึ้นมาใหม่ ถ้ามีให้ประยุกต์ใช้วิธี Move Method เพื่อย้ายเมทธอดนั้น หรือถ้าต้องการย้ายบางส่วนของเมทธอดให้ประยุกต์ใช้วิธี Extract Method เพื่อสร้างเมทธอดใหม่ก่อน แล้วจึงย้ายเมทธอดที่สร้างใหม่นั้น

ข.11 Move Attribute

เป็นการย้ายคุณลักษณะจากคลาสเดิมไปยังคลาสที่มีการเรียกใช้มากกว่า และแก้ไขการเรียกใช้คุณลักษณะนั้นทั้งหมด

```
class Account…
      private AccountType _type;
      private double _interestRate;
      double interestForAmount_days( double amount, int days) {
           return _interestRate * amount * days / 365;        }
```

รูปที่ ข.20 ตัวอย่างโค้ดก่อนการทำวิธี Move Attribute

โดยทำตามขั้นตอนดังนี้

1. ถ้าคุณลักษณะนั้นเป็นพับบลิก ให้ประยุกต์ใช้วิธี Encapsulate Field แต่ถ้าเป็นไพรเวทให้ประยุกต์ใช้วิธี Self Encapsulate Field

```
class Account…
     private AccountType _type;
     private double _interestRate;
     void setInterestRate( double arg ) { _interestRate = arg;     }
     double getInterestRate() {  return _interestRate;          }
     double interestForAmount_days( double amount, int days) {
          return _interestRate * amount * days / 365;           }
```

2. คอมไพล์ และทดสอบ
3. สร้างคุณลักษณะใหม่ในคลาสที่มีการเรียกใช้มากกว่า และใช้เมทธอดที่กำหนดค่าคุณลักษณะ และเมทธอดที่ให้ค่าคุณลักษณะ

```
class AccountType…
     private double _interestRate;
     void setInterestRate( double arg ) { _interestRate = arg;     }
     double getInterestRate() { return _interestRate; }
…
```

4. คอมไพล์คลาสที่ย้ายคุณลักษณะไป
5. ลบคุณลักษณะบนคลาสเดิมออก
6. แก้ไขการอ้างอิงถึงคุณลักษณะที่ย้ายไป แทนที่ด้วยเมทธอดที่กำหนดค่าคุณลักษณะ และเมทธอดที่ให้ค่าคุณลักษณะ

```
class Account…
     private AccountType _type;
     double interestForAmount_days( double amount, int days) {
          return _type.getInterestRate() * amount * days / 365;     }
```

7. คอมไพล์ และทดสอบ

```
class Account…
     private AccountType _type;
     double interestForAmount_days( double amount, int days) {
          return _type.getInterestRate() * amount * days / 365;     }
…
class AccountType…
     private double _interestRate;
     void setInterestRate( double arg ) { _interestRate = arg;     }
     double getInterestRate() { return _interestRate;     }
…
```

รูปที่ ข.21 ตัวอย่างโค้ดหลังการทำวิธี Move Attribute

ข.12 Move Method

เป็นการสร้างเมทธอดใหม่ที่มีโค้ดภายในเหมือนเมทธอดเดิม ภายในคลาสที่มีการเรียกใช้เมทธอดนั้นมากกว่า และแก้ไขโค้ดในเมทธอดเดิมเป็นการเรียกใช้เมทธอดใหม่แทนหรือลบเมทธอดเดิมออก

```
class Account…
     double overdraftCharge() {
          if(_type.isPremium()) {
               double result = 10;
               if(_daysOverdrawn > 7) result += (_daysOverdrawn – 7) * 0.85;
               return result;
          }
          else return _daysOverdrawn * 1.75;
     }
     double bankCharge() {
          double result = 4.5;
          if(_daysOverdrawn > 0) result += overdraftCharge();
          return result;
     }
     private AccountType _type;
     private int _daysOverdrawn;
```

รูปที่ ข.22 ตัวอย่างโค้ดก่อนการทำวิธี Move Method

โดยทำตามขั้นตอนดังนี้

1. ตรวจสอบคุณสมบัติทั้งหมดที่ถูกเรียกใช้ภายในเมทธอดที่ต้องการย้ายว่า ถูกกำหนดไว้ภายในคลาสเจ้าของเมทธอดนั้นหรือไม่ ถ้ามีคุณสมบัติที่ถูกเรียกใช้โดยเมทธอดอื่นให้พิจารณาว่า ควรย้ายเมทธอดนั้นหรือไม่
2. ตรวจสอบคลาสลูกหรือคลาสแม่ของคลาสเจ้าของเมทธอดว่ามีการประกาศเมท ธอดนี้ไว้หรือไม่ ถ้ามีการประกาศไว้ อาจจะไม่สามารถย้ายเมทธอดนั้นได้ เพราะมีการใช้คุณสมบัติของโพลีมอฟิซึม
3. สร้างเมทธอดใหม่ภายในคลาสที่ต้องการ ซึ่งสามารถตั้งชื่อใหม่ได้ตามการใช้งาน
4. คัดลอกโค้ดจากเมทธอดเดิมไปยังเมทธอดใหม่ และแก้ไขการอ้างอิงถึงอ็อบเจกต์ ภายในโค้ดให้ทำงานในคลาสใหม่ได้

```
class AccountType…
     double overdraftCharge(int daysOverdrawn) {
          if(isPremium()) {
               double result = 10;
               if(daysOverdrawn > 7) result += (daysOverdrawn – 7) * 0.85;
               return result;
          }
          else return _daysOverdrawn * 1.75;
     }
}
```

5. คอมไพล์
6. แก้ไขโค้ดภายในเมทธอดเดิมเป็นการเรียกใช้เมทธอดใหม่แทน

```
class Account…
     double overdraftCharge() {
          return _type.overdraftCharge(_daysOverdrawn);     }
…
```

7. คอมไพล์ และทดสอบ
8. พิจารณาว่าควรจะลบเมทธอดเดิมหรือไม่ แต่ถ้ามีการเรียกใช้เมทธอดเดิมมาก ก็ไม่ควรจะลบเมทธอดเดิมออก

```
class Account…
     double overdraftCharge() {
          return _type.overdraftCharge(_daysOverdrawn);     }
…
```

9. ถ้าลบเมทธอดเดิมออก ให้แทนที่การเรียกใช้เมทธอดเดิมด้วยการเรียกใช้เมทธอดใหม่แทน

10. คอมไพล์ และทดสอบ

```
class Account…
     double bankCharge() {
          double result = 4.5;
          if(_daysOverdrawn > 0) result += _type.overdraftCharge(_daysOverdrawn);
          return result;
     }
     int getDaysOverdrawn() { return _daysOverdrawn; }
     private AccountType _type;
     private int _daysOverdrawn;
…
class AccountType…
     double overdraftCharge( Account account) {
          if(isPremium()) {
               double result = 10;
               if(account.getDaysOverdrawn() > 7)
               result += (account.getDaysOverdrawn() – 7) * 0.85;
               return result;
          }
          else return account.getDaysOverdrawn() * 1.75;
     }
}
…
```

รูปที่ ข.23 ตัวอย่างโค้ดหลังการทำวิธี Move Method

ข.13 Preserve Whole Object

เป็นการส่งอ็อบเจกต์ แทนการส่งผ่านค่าพารามิเตอร์ทั้งหมดไปยังการเรียกใช้เมทธอด

```
class Room…
     boolean withinPlan(HeatingPlan plan) {
          int low = daysTempRange().getLow();
          int high = daysTempRange().getHigh();
          return plan.withinRange(low, high);
     }
…
class HeatingPlan…
     boolean withinRange (int low, int high) {
       return (low >= _range.getLow() && high <= _range.getHigh());}
     private TempRange _range;
…
```

รูปที่ ข.24 ตัวอย่างโค้ดก่อนการทำวิธี Preserve Whole Object

โดยทำตามขั้นตอนดังนี้

1. สร้างพารามิเตอร์ใหม่จากอ็อบเจกต์ ที่ให้ค่าข้อมูลที่ส่งไป

```
class HeatingPlan …
     boolean withinRange( TempRange roomRange, int low, int high ) {
          return (low >= _range.getLow() && high <= _range.getHigh());     }
…
class Room …
     boolean withinPlan(HeatingPlan plan) {
          int low = daysTempRange().getLow();
          int high = daysTempRange().getHigh();
          return plan.withinRange(daysTempRange(), low, high);
     }
…
```

2. คอมไพล์ และทดสอบ

3. พิจารณาว่าค่าพารามิเตอร์ใดได้มาจากอ็อบเจกต์ ที่ส่งไป

4. ลบค่าพารามิเตอร์ และแทนที่การอ้างอิงพารามิเตอร์นั้น ด้วยการเรียกใช้เมทธอดของอ็อบเจกต์ ที่ส่งไปแทน

```
class HeatingPlan …
     boolean withinRange( TempRange roomRange ) {
return (roomRange.getLow()>=_range.getLow()&&roomRange.getHigh()<=_range.getHigh());
     }
…
class Room …
     boolean withinPlan(HeatingPlan plan) {
          int low = daysTempRange().getLow();
          int high = daysTempRange().getHigh();
          return plan.withinRange(daysTempRange());
     }
…
```

5. คอมไพล์ และทดสอบ

6. ลบโค้ดในส่วนของการเรียกใช้เมทธอด เพื่อกำหนดค่าให้กับพารามิเตอร์ที่ส่งไป

```
class HeatingPlan …
     boolean withinRange( TempRange roomRange ) {
return (roomRange.getLow()>=_range.getLow()&&roomRange.getHigh()<=_range.getHigh());
     }
…
class Room …
     boolean withinPlan(HeatingPlan plan) {
          return plan.withinRange(daysTempRange());     }
…
```

7. คอมไพล์ และทดสอบ

```
class HeatingPlan …
     boolean withinRange( TempRange roomRange ) {
          return (_range.includes(roomRange));  }
class Room …
     boolean withinPlan(HeatingPlan plan) {
          return plan.withinRange(daysTempRange());     }
class TempRange…
     boolean includes( TempRange arg ) {
      return arg.getLow() >= this.getLow() && arg.getHigh() <= this.getHigh();     }
…
```

รูปที่ ข.25 ตัวอย่างโค้ดหลังการทำวิธี Preserve Whole Object

ข.14 Replace Conditional with Polymorphism

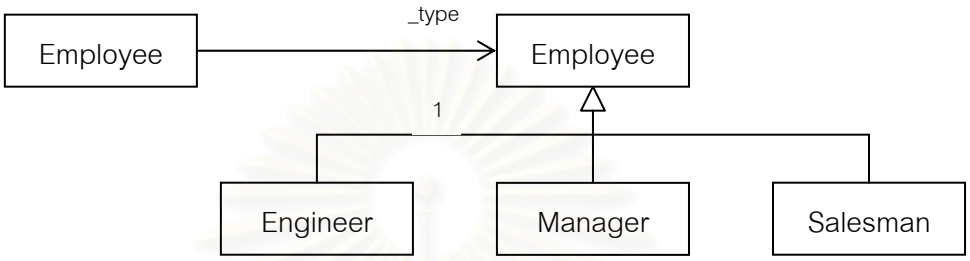
เป็นการแทนที่โค้ดส่วนที่เป็นเงื่อนไขด้วยการสร้างเมทธอดโอเวอร์ไรด์ในคลาสลูก และทำให้เมทธอดเดิมเป็นแอบสเตรค

```
class Employee…
     int payAmount() {
          switch (getType()) {
               case EmployeeType.ENGINEER :
                    return _monthlySalary;
               case EmployeeType.SALESMAN :
                    return _monthlySalary + _commission;
               case EmployeeType.MANAGER :
                    return _monthlySalary + _bonus;
               default :
                    throw new RuntimeException(“Incorrect Employee”);
          }
     }
…
```

รูปที่ ข.26 ตัวอย่างโค้ดก่อนการทำวิธี Replace Conditional with Polymorphism

โดยทำตามขั้นตอนดังนี้

1. ก่อนจะประยุกต์ใช้วิธี Replace Conditional with Polymorphism นั้น ภายในโค้ดต้องมีโครงสร้างการสืบทอดคุณสมบัติ โดยประยุกต์ใช้วิธี Replace Type Code with State/Strategy หรือ Replace Type Code with Subclasses (ตามรายละเอียดในข้อ 19 และ 20)



รูปที่ ข.27 แผนภาพคลาสโครงสร้างการสืบทอดคุณสมบัติ

2. ถ้าสเตทเมนต์ภายในส่วนของเงื่อนไขเป็นส่วนหนึ่งของเมทธอดที่มีขนาดใหญ่ ให้ประยุกต์ใช้วิธี Extract Method เพื่อย้ายโค้ดส่วนนั้นมาสร้างเป็นเมทธอดใหม่
3. ถ้าต้องการย้ายเมทธอดที่ใช้เป็นส่วนของเงื่อนไข ให้ไปอยู่บนสุดของโครงสร้างการสืบทอดคุณสมบัติ ให้ประยุกต์ใช้วิธี Move Method
4. เลือกคลาสลูกมาสร้างเมทธอดที่โอเวอร์ไรด์เมทธอดที่ใช้เป็นส่วนของเงื่อนไข และคัดลอกโค้ดของแต่ละส่วนของเงื่อนไขไปยังเมทธอดที่โอเวอร์ไรด์นั้น
5. คอมไพล์ และทดสอบ
6. ลบโค้ดในแต่ละส่วนของเงื่อนไขออก
7. คอมไพล์ และทดสอบ
8. ทำให้เมทธอดในคลาสแม่เป็นแอบสเตรค

ข.15 Replace Method with Method Object

เป็นการสร้างคลาสใหม่ขึ้นมา เพื่อทำหน้าที่แทนเมทธอดเดิม

```
class Account
     int gamma(int inputVal, int quality, int yearToDate) {
          int importantValue1 = (intputVal * quantity) + delta();
          int importantValue2 = (inputVal * yearToDate) + 100;
          if((yearToDate – importantValue1) > 100)
               importantValue2 -= 20;
          int importantValue3 = importantValue2 * 7;
          return importantValue3 – 2 * importantValue1;
     }
```

รูปที่ ข.28 ตัวอย่างโค้ดก่อนการทำวิธี Replace Method with Method Object

โดยทำตามขั้นตอนดังนี้

1. สร้างคลาสใหม่ และตั้งชื่อ หลังจากสร้างเมทธอด
2. สร้างไฟนอลฟิวส์ (Final field) สำหรับอ็อบเจกต์ ที่เมทธอดเดิมอยู่ และฟิวส์สำหรับแต่ละตัวแปรชั่วคราว และพารามิเตอร์ในเมทธอด

```
class Gamma…
     private final Account _account;
     private int inputVal;
     private int quantity;
     private int yearToDate;
     private int importantValue1;
     private int importantValue2;
     private int importantValue3;…
```

3. สร้างคอนสตรัคเตอร์เมทธอดภายในคลาสใหม่

```
Gamma (Account source, int inputValArg, int quantityArg, int yearToDateArg)
{
          _account = source;
          inputVal = inputValArg;
          quantity = quantityArg;
          yearToDate = yearToDateArg;
}
```

4. สร้างเมทธอด compute
5. คัดลอดโค้ดภายในเมทธอดเดิมไปยังเมทธอด compute และเรียกใช้ฟิวส์ของ อ็อบเจกต์เดิมสำหรับการเรียกใช้เมทธอดบนอ็อบเจกต์ เดิม

```
int compute() {
     importantValue1 = (inputVal * quantity) + _account.date();
     importantValue2 = (inputVal * yearToDate) + 100;
     if((yearToDate – importantValue1) > 100)
          importantValue2 -= 20;
     int importantValue3 = importantValue2 * 7;
     return importantValue3 – 2 * importantValue1;
}
```

6. คอมไพล์
7. แทนที่เมทธอดเดิมด้วยการสร้างอ็อบเจกต์ ใหม่และเรียกใช้เมทธอด compute

```
class Account…
     int gamma(int inputVal, int quantity, int yearToDate) {
          return new Gamma(this, inputVal, quantity, yearToDate).compute();
     }
…
class Gamma…
     private final Account _account;
     private int inputVal;
     private int quantity;
     private int yearToDate;
     private int importantValue1;
     private int importantValue2;
     private int importantValue3;
     Gamma (Account source, int inputValArg, int quantityArg, int yearToDateArg) {
          _account = source;
          inputVal = inputValArg;
          quantity = quantityArg;
          yearToDate = yearToDateArg;
     }
     int compute() {
          importantValue1 = (inputVal * quantity) + _account.date();
          importantValue2 = (inputVal * yearToDate) + 100;
          if((yearToDate – importantValue1) > 100)
               importantValue2 -= 20;
          int importantValue3 = importantValue2 * 7;
          return importantValue3 – 2 * importantValue1;
     }
…
```

รูปที่ ข.29 ตัวอย่างโค้ดหลังการทำวิธี Replace Method with Method Object

ข.16 Replace Parameter with Explicit Methods

เป็นการสร้างเมทธอดใหม่ สำหรับโค้ดที่ต่างกันของค่าพารามิเตอร์แต่ละตัว

```
static final int ENGINEER = 0;
static final int SALESMAN = 1;
static final int MANAGER = 2;
static Employee create(int type) {
     switch (type) {
          case ENGINEER:
               return new Engineer();
          case SALESMAN:
               return new Salesman();
          case MANAGER:
               return new Manager();
          default:
               throw new IllegalArgumentException(“Incorrect type code value”);
     }
}
```

รูปที่ ข.30 ตัวอย่างโค้ดก่อนการทำวิธี Replace Parameter with Explicit Method

โดยทำตามขั้นตอนดังนี้

1. สร้างเมทธอดใหม่สำหรับค่าพารามิเตอร์แต่ละตัว

```
static Employee createEngineer() { return new Engineer(); }
static Employee createSalesman() { return new Salesman(); }
static Employee createManager() {  return new Manager();  }
```

2. กำหนดให้มีการเรียกใช้เมธทอดใหม่ ทีละเงื่อนไข

```
static final int ENGINEER = 0;
static final int SALESMAN = 1;
static final int MANAGER = 2;
static Employee create(int type) {
     switch (type) {
          case ENGINEER:
               return Employee.createEngineer();
          case SALESMAN:
               return Employee.createSalesman();
          case MANAGER:
               return Employee.createManager();
          default:
               throw new IllegalArgumentException(“Incorrect type code value”);
     }
}
```

รูปที่ ข.31 ตัวอย่างโค้ดหลังการทำวิธี Replace Parameter with Explicit Method

3. คอมไพล์และทดสอบ

ข.17 Replace Parameters with Method

เป็นการแทนที่พารามิเตอร์ ด้วยการเรียกใช้เมทธอดที่สร้างขึ้นใหม่แทน

```
public double getPrice() {
     int basePrice = _quantity * _itemPrice;
     int discountLevel;
     if(_quantity > 100) discountLevel = 2;
     else discountLevel = 1;
     double finalPrice = discountedPrice (basePrice, discountLevel);
     return finalPrice;                                              }
private double discountedPrice(int basePrice, int discountLevel) {
     if(discountLevel == 2)  return basePrice * 0.1;
     else return basePrice * 0.05;                                   }
…
```

รูปที่ ข.32 ตัวอย่างโค้ดก่อนการทำวิธี Replace Parameters with Method

โดยทำตามขั้นตอนดังนี้

1. ย้ายโค้ดส่วนที่มีการคำนวณของค่าพารามิเตอร์มาสร้างเป็นเมทธอดใหม่

```
…
public double getPrice() {
     int basePrice = getBasePrice();
     int discountLevel = getDiscountLevel();
     double finalPrice = discountedPrice (basePrice, discountLevel);
     return finalPrice;
}
private int getDiscountLevel() {
     if(_quantity > 100) return 2;
     else return 1;
}
private double getBasePrice() { return _quantity * _itemPrice; }
…
```

2. แทนที่การอ้างอิงไปยังพารามิเตอร์ในเมทธอดด้วยการเรียกใช้เมทธอดใหม่แทน

```
private double discountedPrice(int basePrice, int discountLevel) {
     if(getDiscountLevel() == 2)  return getBasePrice * 0.1;
     else return getBasePrice * 0.05;
}
```

3. คอมไพล์และทดสอบ

4. ใช้วิธี Remove Parameter สำหรับค่าพารามิเตอร์แต่ละตัว

```
…
public double getPrice() { return discountedPrice(); }
private double discountedPrice(int basePrice) {
     if(getDiscountLevel() == 2)  return getBasePrice * 0.1;
     else return getBasePrice * 0.05;
}
private int getDiscountLevel() {
     if(_quantity > 100) return 2;
     else return 1;
}
private double getBasePrice() { return _quantity * _itemPrice; }
…
```

รูปที่ ข.33 ตัวอย่างโค้ดหลังทำวิธี Replace Parameters with Method

ข.18 Replace Temp with Query

เป็นการแทนที่ตัวแปรชั่วคราวด้วยการเรียกใช้เมทธอด

```
…
double getPrice() {
     int basePrice = _quantity * _itemPrice;
     double discountFactor;
     if (basePrice > 1000) discountFactor = 0.95;
     else discountFactor = 0.98;
     return basePrice * discountFactor;
}
…
```

รูปที่ ข.34 ตัวอย่างโค้ดก่อนทำวิธี Replace Temp with Query

โดยทำตามขั้นตอนดังนี้

1. พิจารณาตัวแปรชั่วคราวที่มีการกำหนดค่าเพียงครั้งเดียว ถ้ามีการกำหนดค่ามากกว่าหนึ่งครั้งให้ประยุกต์ใช้วิธี Split Temporary Variable แทน
2. กำหนดตัวแปรชั่วคราวนั้นเป็นไฟนอล

```
double getPrice() {
     final int basePrice = _quantity * _itemPrice;
     final double discountFactor;
     if (basePrice > 1000) discountFactor = 0.95;
     else discountFactor = 0.98;
     return basePrice * discountFactor;
}
```

3. คอมไพล์ เพื่อตรวจสอบว่ามีการกำหนดค่าตัวแปรนั้นเพียงครั้งเดียว
4. ย้ายโค้ดทางด้านขวาที่กำหนดค่าให้ตัวแปรนั้นไปสร้างเป็นเมทธอดใหม่

```
double getPrice() {
     final int basePrice = basePrice();
     final double discountFactor = discountFactor();
     return basePrice * discountFactor;
}
private int basePrice() { return _quantity * _itemPrice; }
private double discountFactor() {
     if (basePrice > 1000) return 0.95;
     else return 0.98;
}
```

5. คอมไพล์และทดสอบ
6. แทนที่ตัวแปรชั่วคราวด้วยเมทธอดใหม่ที่สร้างขึ้น และลบตัวแปรชั่วคราวนั้นออก

```
double getPrice() { return basePrice() * discountFactor(); }
private int basePrice() { return _quantity * _itemPrice; }
private double discountFactor() {
     if (basePrice > 1000) return 0.95;
     else return 0.98;
}
```

รูปที่ ข.35 ตัวอย่างโค้ดหลังจากทำวิธี Replace Temp with Query

ข.19 Replace Type Code with State/Strategy

เป็นการแทนที่ประเภทของโค้ด ด้วยสเตทอ็อบเจกต์

```
class Employee…
     private int _type;
     static final int ENGINEER = 0;
     static final int SALESMAN = 1;
     static final int MANAGER = 2;
     Employee (int type) {
          _type = type;
     }
     int payAmount() {
          switch (_type) {
               case ENGINEER:
                    return _monthlySalary;
               case SALESMAN:
                    return _monthlySalary + _commission;
               case MANAGER:
                    return _monthlySalary + _bonus;
               default:
                    throw new RuntimeExeption(“Incorrect Employee”);
          }
     }
…
```

รูปที่ ข.36 ตัวอย่างโค้ดก่อนการทำวิธี Replace Type Code with State/Strategy

โดยทำตามขั้นตอนดังนี้

1. ประยุกต์ใช้วิธี Self Encapsulate Field

```
…
     Employee (int type) { setType(type);  }
     int getType() { return _type;  }
     void setType(int arg) { _type = arg;  }
     void payAmount() {
          switch (getType()) {
               case ENGINEER:
                    return _monthlySalary;
               case SALESMAN:
                    return _monthlySalary + _commission;
               case MANAGER:
                    return _monthlySalary + _bonus;
               default:
                    throw new RuntimeExeption(“Incorrect Employee”);
          }
     }
…
```

2. สเตทอ็อบเจกต์ คือ การสร้างคลาสใหม่ สำหรับแต่ละประเภทของโค้ด
3. เพิ่มคลาสลูกของสเตทอ็อบเจกต์ โดยคลาสลูกหนึ่งคลาสสำหรับแต่ละประเภทของโค้ด
4. สร้างแอบสเตรกเมทธอดในคลาสใหม่ และสร้างโอเวอร์ไรด์เมทธอดในคลาสลูก เพื่อส่งค่าประเภทของโค้ด

```
…
     abstract class EmployeeType { abstract int getTypeCode();  }
     class Engineer extends EmployeeType {
          int getTypeCode() { return Employee.ENGINEER;         }
     }
     class Manager extends EmployeeType {
          int getTypeCode() { return Employee.MANAGER;          }
     }
     class Salesman extends EmployeeType {
          int getTypeCode() { return Employee.SALESMAN;         }
     }
…
```

5. คอมไพล์
6. สร้างฟิวส์ในคลาสเดิมสำหรับเรียกใช้สเตทอ็อบเจกต์ ใหม่

```
class Employee…
     private EmployeeType _type;
     int getType() {
          return _type.getTypeCode();
     }
     void setType(int arg) {
          switch(arg) {
               case ENGINEER:
                    _type = new Engineer();
                    break;
               case SALESMAN:
                    _type = new Salesman();
                    break;
               case MANAGER:
                    _type = new Manager();
                    break;
               default:
                    throw new RuntimeExeption(“Incorrect Employee”);
          }
     }
…
```

7. เปลี่ยนการเรียกใช้ประเภทของโค้ดภายในคลาสเดิม ให้อ้างอิงไปยังสเตทอ็อบเจกต์
8. เปลี่ยนการเรียกใช้ประเภทของโค้ดภายในเมทธอดที่กำหนดคุณลักษณะของคลาสเดิม ด้วยการกำหนดค่าให้กับอินสเตนท์ของสเตทอ็อบเจกต์ ในคลาสลูกแทน

```
class Employee…
     private EmployeeType _type;
     int getType() {
          return _type.getTypeCode();
     }
     void setType(int arg) {
          _type = EmployeeType.newType(arg);
     }
…
class EmployeeType…
     static EmployeeType newType(int code){
          switch(code) {
               case ENGINEER:
                    _type = new Engineer();
                    break;
               case SALESMAN:
                    _type = new Salesman();
                    break;
               case MANAGER:
                    _type = new Manager();
                    break;
               default:
                    throw new RuntimeExeption(“Incorrect Employee”);
          }
     }
     static final int ENGINEER = 0;
     static final int SALESMAN = 1;
     static final int MANAGER = 2;
…
```

9. คอมไพล์และทดสอบ

```
class Employee…
     private EmployeeType _type;
     int getType() {
          return _type.getTypeCode();
     }
     void setType(int arg) {
          _type = EmployeeType.newType(arg);
     }
     int payAmount() {
          switch (getType()) {
               case EmployeeType.ENGINEER:
                    return _monthlySalary;
               case EmployeeType.SALESMAN:
                    return _monthlySalary + _commission;
               case EmployeeType.MANAGER:
                    return _monthlySalary + _bonus;
               default:
                    throw new RuntimeExeption(“Incorrect Employee”);
          }
     }
…
class EmployeeType…
     static EmployeeType newType(int code){
          switch(code) {
               case ENGINEER:
                    _type = new Engineer();
                    break;
               case SALESMAN:
                    _type = new Salesman();
                    break;
               case MANAGER:
                    _type = new Manager();
                    break;
               default:
                    throw new RuntimeExeption(“Incorrect Employee”);
          }
     }
     static final int ENGINEER = 0;
     static final int SALESMAN = 1;
     static final int MANAGER = 2;
…
```

รูปที่ ข.37 ตัวอย่างโค้ดหลังการทำ Replace Type Code with State/Strategy

ข.20 Replace Type Code with Subclasses

เป็นการแทนที่ประเภทของโค้ด ด้วยการสร้างคลาสลูก

```
class Employee…
     private int _type;
     static final int ENGINEER = 0;
     static final int SALESMAN = 1;
     static final int MANAGER = 2;
     Employee (int type) {
          _type = type;
     }
```

รูปที่ ข.38 ตัวอย่างโค้ดก่อนการทำวิธี Replace Type Code with Subclasses

โดยทำตามขั้นตอนดังนี้

1. ประยุกต์ใช้วิธี Self Encapsulate Field

```
class Employee…
     int getType() {
          return _type;
     }
     Employee create (int type) {
          return new Employee(type);
     }
     private Employee (int type) {
          _type = type;
     }
…
```

2. สร้างคลาสลูกใหม่ สำหรับแต่ละประเภทของโค้ด และโอเวอร์ไรด์เมทธอดที่ให้ค่าคุณลักษณะในคลาสลูก เพื่อให้ค่าที่เหมาะสม

```
class Engineer extends Employee {
     int getType() {
          return Employee.ENGINEER;
     }
}
class Employee…
     static Employee create (int type) {
          if (type == ENGINEER) return new Engineer();
          else return new Employee(type);
     }
…
```

3. คอมไพล์ และทดสอบ หลังจากแทนที่ค่าแต่ละประเภทของโค้ดด้วยสร้างลูกที่สร้างขึ้น
4. ลบค่าประเภทของโค้ดจากคลาสแม่ และประกาศเอคเซสเซอร์สำหรับประเภทของโค้ดเป็นแอบสเตรค

```
abstract int getType();

static Employee create(int type) {
     switch (type) {
          case ENGINEER :
               return new Engineer();
          case SALESMAN :
               return new Salesman();
          case MANAGER :
               return new Manager();
     }
}
```

5. คอมไพล์ และทดสอบ

```
class Employee…
     private int _type;
     static final int ENGINEER = 0;
     static final int SALESMAN = 1;
     static final int MANAGER = 2;
     Employee (int type) {
          _type = type;
     }
     abstract int getType();
     private Employee (int type) {
          _type = type;
     }
     static Employee create(int type) {
          switch (type) {
               case ENGINEER :
                    return new Engineer();
               case SALESMAN :
                    return new Salesman();
               case MANAGER :
                    return new Manager();
          }
     }
}
class Engineer extends Employee {
     int getType() {
          return Employee.ENGINEER;
     }
}
class Salesman extends Employee {
     int getType() {
          return Employee.SALESMAN;
     }
}
class Manager extends Employee {
     int getType() {
          return Employee.MANAGER;
     }
}
```

รูปที่ ข.39 ตัวอย่างโค้ดหลังการทำวิธี Replace Type Code with Subclasses

ภาคผนวก ค

ซอร์สโค้ดของโปรแกรมที่นำมาทดสอบก่อนและหลังการทำรีแฟคทอริง

ค.1 โปรแกรมที่ 2

ค.1.1 ซอร์สโค้ดของโปรแกรมที่ 2 ก่อนการทำรีแฟคทอริง

```
import java.util.Date;

class BusinessSite
{
	private int lastReading;
	private Reading[] _readings = new Reading[1000];
	private static final double START_RATE = 0.09;
	private static final double END_RATE = 0.05;
	private static final int END_AMOUNT = 1000;
	private Zone _zone;

	BusinessSite(Zone zone) {
		_zone = zone;
	}

	public void addReading(Reading newReading) {
		_readings[++lastReading] = newReading;
	}
	public Dollars charge()
	{
int usage = _readings[lastReading].amount() - _readings[lastReading -1].amount();
		return charge(usage);
	}
	private Dollars charge(int usage)
	{
		Dollars result;
		if (usage == 0) return new Dollars(0);
double t1 = START_RATE - ((END_RATE * END_AMOUNT) - START_RATE) / (END_AMOUNT - 1);
double t2 = ((END_RATE * END_AMOUNT) - START_RATE) * Math.min(END_AMOUNT, usage)
/(END_AMOUNT - 1);
double t3 = Math.max(usage - END_AMOUNT, 0) * END_RATE;
		result = new Dollars (t1 + t2 + t3);
		result = result.plus(new Dollars (usage * 0.0175));
		Dollars base = new Dollars (result.min(new Dollars (50)).times(0.07));
		if (result.isGreaterThan(new Dollars (50))) {
		base = new Dollars (base.plus(result.min(new Dollars(75)).minus(new
Dollars(50)).times(0.06)));
		};
		if (result.isGreaterThan(new Dollars (75))) {
	base = new Dollars (base.plus(result.minus(new Dollars(75)).times(0.05)));
		};
		result = result.plus(base);
		return result;
	}
}
class DisabilitySite {
	private Reading[] _readings = new Reading[1000];
	private static final Dollars FUEL_TAX_CAP = new Dollars (0.10);
	private static final double TAX_RATE = 0.05;
	private Zone _zone;
	private static final int CAP = 200;
	DisabilitySite(Zone zone) {
		_zone = zone;
	}
	public void addReading(Reading newReading)
	{
		int i;
		for (i = 0; _readings[i] != null; i++);
			_readings[i] = newReading;
	}
	public Dollars charge()
	{
```

```
        int i;
        for (i = 0; _readings[i] != null; i++);
        int usage = _readings[i-1].amount() - _readings[i-2].amount();
        Date end = _readings[i-1].date();
        Date start = _readings[i-2].date();
        start.setDate(start.getDate() + 1); //set to begining of period
        return charge(usage, start, end);
}
private Dollars charge(int fullUsage, Date start, Date end)
{
        Dollars result;
        double summerFraction;
        int usage = Math.min(fullUsage, CAP);
        if (start.after(_zone.summerEnd()) || end.before(_zone.summerStart()))
                summerFraction = 0;
        else if (!start.before(_zone.summerStart())
       && !start.after(_zone.summerEnd())&&!end.before(_zone.summerStart())
       && !end.after(_zone.summerEnd()))
                summerFraction = 1;
        else {
                double summerDays;
if (start.before(_zone.summerStart()) || start.after(_zone.summerEnd())) {
                // end is in the summer
        summerDays = dayOfYear(end) - dayOfYear (_zone.summerStart()) + 1;
                } else {
                // start is in summer
        summerDays = dayOfYear(_zone.summerEnd()) - dayOfYear (start) + 1;
                };
        summerFraction = summerDays / (dayOfYear(end) - dayOfYear(start) + 1);
        };
result = new Dollars ((usage * _zone.summerRate() *
summerFraction)+(usage * _zone.winterRate() * (1 - summerFraction)));
result = result.plus(new Dollars (Math.max(fullUsage - usage, 0) * 0.062));
result = result.plus(new Dollars (result.times(TAX_RATE)));
Dollars fuel = new Dollars(fullUsage * 0.0175);
result = result.plus(fuel);
result = new Dollars (result.plus(fuel.times(TAX_RATE).min(FUEL_TAX_CAP)));
return result;
}
int dayOfYear(Date arg) {
        int result;
        switch (arg.getMonth()) {
        case 0:
                result = 0;
                break;
        case 1:
                result = 31;
                break;
        case 2:
                result = 59;
                break;
        case 3:
                result = 90;
                break;
        case 4:
                result = 120;
                break;
        case 5:
                result = 151;
                break;
        case 6:
                result = 181;
                break;
        case 7:
                result = 212;
                break;
        case 8:
                result = 243;
                break;
        case 9:
                result = 273;
                break;
        case 10:
                result = 304;
                break;
        case 11:
                result = 334;
```

```
                        break;
                default :
                        throw new IllegalArgumentException();
                };
                result += arg.getDate();
                //check leap year
if ((arg.getYear()%4 == 0) && ((arg.getYear() % 100 != 0) ||((arg.getYear() + 1900) %
400 == 0)))
                {
                        result++;
                };
                return result;
        }
}
public class LifelineSite
{
        private Reading[] _readings = new Reading[1000];
        private static final double TAX_RATE = 0.05;
        LifelineSite(){ }
        public void addReading(Reading newReading)
        {
                Reading[] newArray = new Reading[_readings.length + 1];
                System.arraycopy(_readings, 0, newArray, 1, _readings.length);
                newArray[0] = newReading;
                _readings = newArray;
        }
        public Dollars charge()
        {
                int usage = _readings[0].amount() - _readings[1].amount();
                return charge(usage);
        }
        private Dollars charge(int usage)
        {
                double base = Math.min(usage,100) * 0.03;
                if (usage > 100) {
                        base += (Math.min (usage,200) - 100) * 0.05;
                };
                if (usage > 200) {
                        base += (usage - 200) * 0.07;
                };
                Dollars result = new Dollars (base);
                Dollars tax = new Dollars (result.minus(new Dollars(8)).max(new Dollars
(0)).times(TAX_RATE));
                result = result.plus(tax);
                Dollars fuelCharge = new Dollars (usage * 0.0175);
                result = result.plus (fuelCharge);
                return result.plus (new Dollars (fuelCharge.times(TAX_RATE)));
        }
}

class Reading {
        private Date _date;
        private int _amount;

        public Reading(int amount, Date date) {
                _amount = amount;
                _date = date;
        }

        public Date date() {
                return _date;
        }
        public int amount() {
                return _amount;
        }
}

class ResidentialSite {

        private Reading[] _readings = new Reading[1000];
        private static final double TAX_RATE = 0.05;
        private Zone _zone;

        ResidentialSite (Zone zone) {
        _zone = zone;
        }
```

```
        public void addReading(Reading newReading)
        {
                // add reading to end of array
                int i = 0;
                while (_readings[i] != null) i++;
                        _readings[i] = newReading;
        }
        public Dollars charge()
        {
                // find last reading
                int i = 0;
                while (_readings[i] != null) i++;
                int usage = _readings[i-1].amount() - _readings[i-2].amount();
                Date end = _readings[i-1].date();
                Date start = _readings[i-2].date();
                start.setDate(start.getDate() + 1); //set to begining of period
                return charge(usage, start, end);
        }

        private Dollars charge(int usage, Date start, Date end)
        {
                Dollars result;
                double summerFraction;
                // Find out how much of period is in the summer
                if (start.after(_zone.summerEnd()) || end.before(_zone.summerStart()))
                        summerFraction = 0;
                else                if                (!start.before(_zone.summerStart())
&& !start.after(_zone.summerEnd()) &&
                !end.before(_zone.summerStart()) && !end.after(_zone.summerEnd()))
                        summerFraction = 1;
                else { // part in summer part in winter
                        double summerDays;
                        if             (start.before(_zone.summerStart())             ||
start.after(_zone.summerEnd())) {
                        // end is in the summer
                                summerDays     =     dayOfYear(end)     -     dayOfYear
(_zone.summerStart()) + 1;
                        } else {
                        // start is in summer
                                summerDays  =  dayOfYear(_zone.summerEnd())  -  dayOfYear
(start) + 1;
                        };
                                summerFraction   =   summerDays   /   (dayOfYear(end)   -
dayOfYear(start) + 1);
                };
                result = new Dollars ((usage * _zone.summerRate() * summerFraction) +
(usage * _zone.winterRate() * (1 - summerFraction)));
                result = result.plus(new Dollars (result.times(TAX_RATE)));
                Dollars fuel = new Dollars(usage * 0.0175);
                result = result.plus(fuel);
                result = new Dollars (result.plus(fuel.times(TAX_RATE)));
                return result;
        }

                int dayOfYear(Date arg) {
                int result;
                switch (arg.getMonth()) {
                case 0:
                        result = 0;
                        break;
                case 1:
                        result = 31;
                        break;
                case 2:
                        result = 59;
                        break;
                case 3:
                        result = 90;
                        break;
                case 4:
                        result = 120;
                        break;
                case 5:
                        result = 151;
                        break;
                case 6:
                        result = 181;
```

```
                        break;
                case 7:
                        result = 212;
                        break;
                case 8:
                        result = 243;
                        break;
                case 9:
                        result = 273;
                        break;
                case 10:
                        result = 304;
                        break;
                case 11:
                        result = 334;
                        break;
                default :
                        throw new IllegalArgumentException();
                };
                result += arg.getDate();
                //check leap year
                if  ((arg.getYear()%4  ==  0)  &&  ((arg.getYear()  %  100  !=  0)
||((arg.getYear() + 1900) % 400 == 0)))
                {
                        result++;
                };
                return result;
        }
}

class Zone {
        private String _name;
        private Date _summerEnd;
        private Date _summerStart;
        private double _winterRate;
        private double _summerRate;

        Zone (String name, double summerRate, double winterRate,Date summerStart, Date
summerEnd) {
                _name = name;
                _summerRate = summerRate;
                _winterRate = winterRate;
                _summerStart = summerStart;
                _summerEnd = summerEnd;
        }

        public Zone persist() {
                Registrar.add("Zone", this);
                return this;
        }
        public static Zone get (String name) {
                return (Zone) Registrar.get("Zone", name);
        }
        public Date summerEnd() {
                return _summerEnd;
        }
        public Date summerStart() {
                return _summerStart;
        }
        public double winterRate() {
                return _winterRate;
        }
        public double summerRate() {
                return _summerRate;
        }
}
```

ค.1.2 ซอร์สโค้ดของโปรแกรมที่ 2 หลังจากประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 1

```
class DisabilitySite {
…
        protected static final double FUEL_CHARGE_RATE = 0.0175;
…
        private Dollars charge(int fullUsage, Date start, Date end)
        {
                Dollars result;
```

```
                double summerFraction;
                int usage = Math.min(fullUsage, CAP);
                if (start.after(_zone.summerEnd()) || end.before(_zone.summerStart()))
                        summerFraction = 0;
                else if (!start.before(_zone.summerStart())
               && !start.after(_zone.summerEnd())&&!end.before(_zone.summerStart())
               && !end.after(_zone.summerEnd()))
                        summerFraction = 1;
                else {
                        double summerDays;
                        if (start.before(_zone.summerStart()) ||
                       start.after(_zone.summerEnd())) {
                        // end is in the summer
                summerDays = dayOfYear(end) - dayOfYear (_zone.summerStart()) + 1;
                        } else {
                        // start is in summer
                summerDays = dayOfYear(_zone.summerEnd()) - dayOfYear (start) + 1;
                        };
                summerFraction = summerDays / (dayOfYear(end) - dayOfYear(start) + 1);
                };
result = new Dollars ((usage * _zone.summerRate() * summerFraction)+(usage *
_zone.winterRate() * (1 - summerFraction)));
result = result.plus(new Dollars (Math.max(fullUsage - usage, 0) * 0.062));
result = result.plus(new Dollars (result.times(TAX_RATE)));
result = result.plus(fuelCharge(fullUsage));
result = new Dollars (result.plus(fuel.times(TAX_RATE).min(FUEL_TAX_CAP)));
return result;
        }
        protected Dollars fuelCharge(int fullUsage){
                return new Dollars(fullUsage * FUEL_CHARGE_RATE);
        }
…
```

ค.1.3 ซอร์สโค้ดของโปรแกรมที่ 2 หลังจากประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 2

```
…
class DisabilitySite {
…
        private Dollars charge(int fullUsage, Date start, Date end)
        {
                Dollars result;
                int usage = Math.min(fullUsage, CAP);
                result = new Dollars ((usage * _zone.summerRate() *
               summerFraction(start,end))+(usage * _zone.winterRate() * (1 –
               summerFraction(start,end))));
                result = result.plus(new Dollars (Math.max(fullUsage - usage, 0) *
               0.062));
                result = result.plus(new Dollars (result.times(TAX_RATE)));
                result = result.plus(fuelCharge(fullUsage));
                result = result.plus(fuelChargeTaxes(fullUsage));
                return result;
        }
        protected Dollars fuelChargeTaxes(int usage){
                return new Dollars
              (fuelCharge(usage).time(TAX_RATE).min(FUEL_TAX_CAP));
        }
        private double summerFraction(Date start, Date end){
                double summerFraction;
                // Find out how much of period is in the summer
                if (start.after(_zone.summerEnd()) || end.before(_zone.summerStart()))
                        summerFraction = 0;
                else if (!start.before(_zone.summerStart())
               && !start.after(_zone.summerEnd()) &&
                !end.before(_zone.summerStart()) && !end.after(_zone.summerEnd()))
                        summerFraction = 1;
                else { // part in summer part in winter
                        double summerDays;
                        if (start.before(_zone.summerStart()) ||
                       start.after(_zone.summerEnd())) {
                        // end is in the summer
                                summerDays = dayOfYear(end) - dayOfYear
                             (_zone.summerStart()) + 1;
                        } else {
                        // start is in summer
                                summerDays = dayOfYear(_zone.summerEnd()) - dayOfYear
                             (start) + 1;
```

```
                        };
                                summerFraction = summerDays / (dayOfYear(end) –
                             dayOfYear(start) + 1);
                };
                return summerFraction;
        }
…
}
…
```

ค.1.4 ซอร์สโค้ดของโปรแกรมที่ 2 หลังจากประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 3

```
…
class ResidentialSite {
…
        protected static final double FUEL_CHARGE_RATE = 0.0175;
        private Dollars charge(int usage, Date start, Date end)
        {
                Dollars result;
                double summerFraction;
                // Find out how much of period is in the summer
                if (start.after(_zone.summerEnd()) || end.before(_zone.summerStart()))
                        summerFraction = 0;
                else if (!start.before(_zone.summerStart())
               && !start.after(_zone.summerEnd()) &&
                !end.before(_zone.summerStart()) && !end.after(_zone.summerEnd()))
                        summerFraction = 1;
                else { // part in summer part in winter
                        double summerDays;
                        if (start.before(_zone.summerStart()) ||
                       start.after(_zone.summerEnd())) {
                        // end is in the summer
                                summerDays = dayOfYear(end) - dayOfYear
                             (_zone.summerStart()) + 1;
                        } else {
                        // start is in summer
                                summerDays = dayOfYear(_zone.summerEnd()) - dayOfYear
                             (start) + 1;
                        };
                                summerFraction = summerDays / (dayOfYear(end) –
                              dayOfYear(start) + 1);
                };
                result = new Dollars ((usage * _zone.summerRate() * summerFraction) +
              (usage * _zone.winterRate() * (1 - summerFraction)));
                result = result.plus(new Dollars (result.times(TAX_RATE)));
                result = result.plus(fuelCharge(usage));
                result = new Dollars (result.plus(fuel.times(TAX_RATE)));
                return result;
        }

        protected Dollars fuelCharge(int fullUsage){
                return new Dollars(fullUsage * FUEL_CHARGE_RATE);
        }
…
}
…
```

ค.1.5 ซอร์สโค้ดของโปรแกรมที่ 2 หลังจากประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 4

```
…
class ResidentialSite {
…
        private Dollars charge(int usage, Date start, Date end)
        {
                Dollars result;
                result = new Dollars ((usage * _zone.summerRate() *
               summerFraction(start, end)) + (usage * _zone.winterRate() * (1 –
               summerFraction(start,end))));
                result = result.plus(new Dollars (result.times(TAX_RATE)));
                result = result.plus(fuelCharge(usage));
                result = result.plus(fuelChargeTaxes(usage));
                return result;
        }

        protected Dollars fuelChargeTaxes(int usage){
```

```
                return new Dollars(fuelCharge(usage).times(TAX_RATE));
        }

        private double summerFraction(Date start, Date end){
                double summerFraction;
                // Find out how much of period is in the summer
                if (start.after(_zone.summerEnd()) || end.before(_zone.summerStart()))
                        summerFraction = 0;
                else if (!start.before(_zone.summerStart())
               && !start.after(_zone.summerEnd()) &&
                !end.before(_zone.summerStart()) && !end.after(_zone.summerEnd()))
                        summerFraction = 1;
                else { // part in summer part in winter
                        double summerDays;
                        if (start.before(_zone.summerStart()) ||
                      start.after(_zone.summerEnd())) {
                        // end is in the summer
                                summerDays = dayOfYear(end) - dayOfYear
                             (_zone.summerStart()) + 1;
                        } else {
                        // start is in summer
                                summerDays = dayOfYear(_zone.summerEnd()) - dayOfYear
                             (start) + 1;
                        };
                                summerFraction = summerDays / (dayOfYear(end) –
                             dayOfYear(start) + 1);
                };
                return summerFraction;
        }
…
}
```

ค.1.6 ซอร์สโค้ดของโปรแกรมที่ 2 หลังจากประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 5

```
…
class DisabilitySite {
…
        private int lastUsage(){
                int i=0;
                while(_reading[i]!= null) i++;
                return _reading[i-1].amount - _reading[i-2].amount;
        }

        private Dollars charge(Date start, Date end)
        {
                Dollars result;
                int usage = Math.min(lastUsage(), CAP);
                result = new Dollars ((usage * _zone.summerRate() *
               summerFraction(start,end))+(usage * _zone.winterRate() * (1 –
               summerFraction(start,end))));
                result = result.plus(new Dollars (Math.max(fullUsage - usage, 0) *
               0.062));
                result = result.plus(new Dollars (result.times(TAX_RATE)));
                result = result.plus(fuelCharge(lastUsage()));
                result = result.plus(fuelChargeTaxes(lastUsage()));
                return result;
        }
…
}
…
```

ค.1.7 ซอร์สโค้ดของโปรแกรมที่ 2 หลังจากประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 6

```
…
class ResidentialSite {
       …
        private Dollars charge(Date start, Date end)
        {
                Dollars result;
                result = new Dollars ((lastUsage() * _zone.summerRate() *
               summerFraction(start, end)) + (lastUsage() * _zone.winterRate() * (1 –
               summerFraction(start,end))));
                result = result.plus(new Dollars (result.times(TAX_RATE)));
                result = result.plus(fuelCharge());
                result = result.plus(fuelChargeTaxes(lastUsage()));
```

```
                return result;
        }

        private int lastUsage(){
                int i=0;
                while(_reading[i]!= null) i++;
                return _reading[i-1].amount - _reading[i-2].amount;
        }
…
}
…
```

ค.1.8 ซอร์สโค้ดของโปรแกรมที่ 2 หลังจากประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 7

```
…
class BusinessSite
{
       …
        private Dollars charge(int usage)
        {
                Dollars result = baseCharge();
                //fuel charge
                result = result.plus(new Dollars (usage * 0.0175));
                //add in the taxes
                Dollars base = new Dollars (result.min(new Dollars (50)).times(0.07));
                if (result.isGreaterThan(new Dollars (50))) {
                        base = new Dollars (base.plus(result.min(new
                      Dollars(75)).minus(new Dollars(50)).times(0.06)));
                };
                if (result.isGreaterThan(new Dollars (75))) {
                        base = new Dollars (base.plus(result.minus(new
                      Dollars(75)).times(0.05)));
                };
                result = result.plus(base);
                return result;
        }

        public Dollars baseCharge(){
                if (usage == 0) return new Dollars(0);
                double t1 = START_RATE - ((END_RATE * END_AMOUNT) - START_RATE) /
               (END_AMOUNT - 1);
                double t2 = ((END_RATE * END_AMOUNT) - START_RATE) *
               Math.min(END_AMOUNT, usage) /(END_AMOUNT - 1);
                double t3 = Math.max(usage - END_AMOUNT, 0) * END_RATE;
                return new Dollars (t1 + t2 + t3);
        }
}
…
```

ค.1.9 ซอร์สโค้ดของโปรแกรมที่ 2 หลังจากประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 8

```
…
class DisabilitySite {
…
                int dayOfYear(Date arg) {
                int result;
                switch (arg.getMonth()) {
                case 0:
                        result = 0;
                        break;
                case 1:
                        result = 31;
                        break;
                case 2:
                        result = 59;
                        break;
                case 3:
                        result = 90;
                        break;
                case 4:
                        result = 120;
                        break;
                case 5:
                        result = 151;
                        break;
```

```
                case 6:
                        result = 181;
                        break;
                case 7:
                        result = 212;
                        break;
                case 8:
                        result = 243;
                        break;
                case 9:
                        result = 273;
                        break;
                case 10:
                        result = 304;
                        break;
                case 11:
                        result = 334;
                        break;
                default :
                        throw new IllegalArgumentException();
                };
                result += arg.getDate();
                //check leap year
                if(isLeapYear()) result++;
                return result;
        }

        boolean isLeapYear(){
                //check leap year
                 return (arg.getYear()%4 == 0) && ((arg.getYear() % 100 != 0)
                ||((arg.getYear() + 1900) % 400 == 0));
        }
}
…
```

ค.1.10 ซอร์สโค้ดของโปรแกรมที่ 2 หลังจากประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 9

```
…
class ResidentialSite {
…
                int dayOfYear(Date arg) {
                int result;
                switch (arg.getMonth()) {
                case 0:
                        result = 0;
                        break;
                case 1:
                        result = 31;
                        break;
                case 2:
                        result = 59;
                        break;
                case 3:
                        result = 90;
                        break;
                case 4:
                        result = 120;
                        break;
                case 5:
                        result = 151;
                        break;
                case 6:
                        result = 181;
                        break;
                case 7:
                        result = 212;
                        break;
                case 8:
                        result = 243;
                        break;
                case 9:
                        result = 273;
                        break;
                case 10:
                        result = 304;
                        break;
```

```
                case 11:
                        result = 334;
                        break;
                default :
                        throw new IllegalArgumentException();
                };
                result += arg.getDate();
                //check leap year
                if(isLeapYear()) result++;
                return result;
        }
        boolean isLeapYear(){
                //check leap year
                 return (arg.getYear()%4 == 0) && ((arg.getYear() % 100 != 0)
                ||((arg.getYear() + 1900) % 400 == 0));
        }
}
…
```

ค.1.11 ซอร์สโค้ดของโปรแกรมที่ 2 หลังจากประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 10

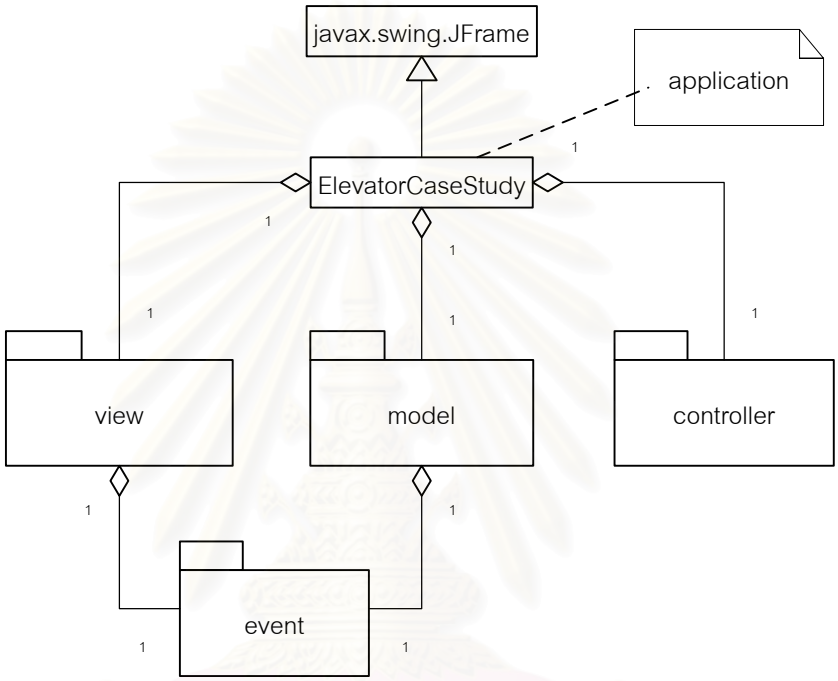
```
…
class ResidentialSite {
…
        int dayOfYear(Date arg) {
                int result;
                //daysToStartsOfMonth
                result = daysToStartOfMonth();
                result += arg.getDate();
                //check leap year
                if(isLeapYear()) result++;
                return result;
        }

        private int daysToStartOfMonth(){
          int[] monthNumbers = {0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334};
          return monthNumbers[getMonth()];
        }
…
}
…
```

ค.1.12 ซอร์สโค้ดของโปรแกรมที่ 2 หลังจากประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 11

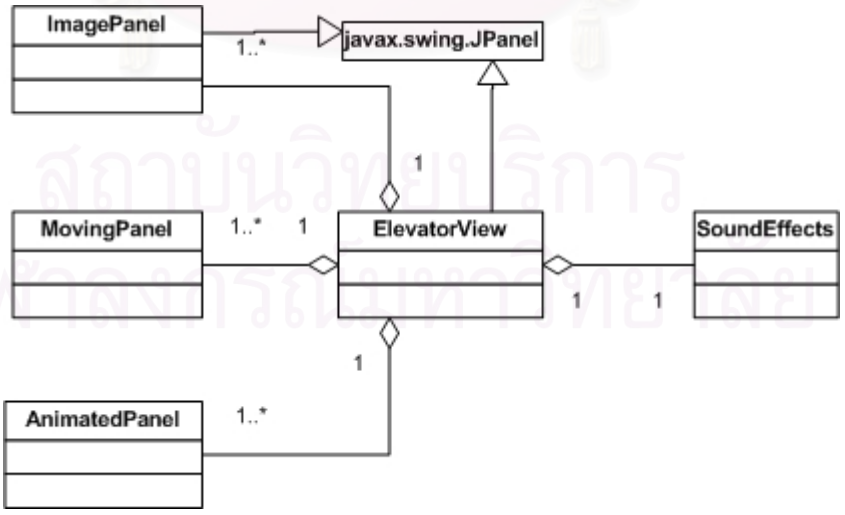
```
…
class DisabilitySite {
…
        int dayOfYear(Date arg) {
                int result;
                //daysToStartsOfMonth
                result = daysToStartOfMonth();
                result += arg.getDate();
                //check leap year
                if(isLeapYear()) result++;
                return result;
        }

        private int daysToStartOfMonth(){
          int[] monthNumbers = {0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334};
          return monthNumbers[getMonth()];
        }
…
}
…
```

ค.2 โปรแกรมที่ 3

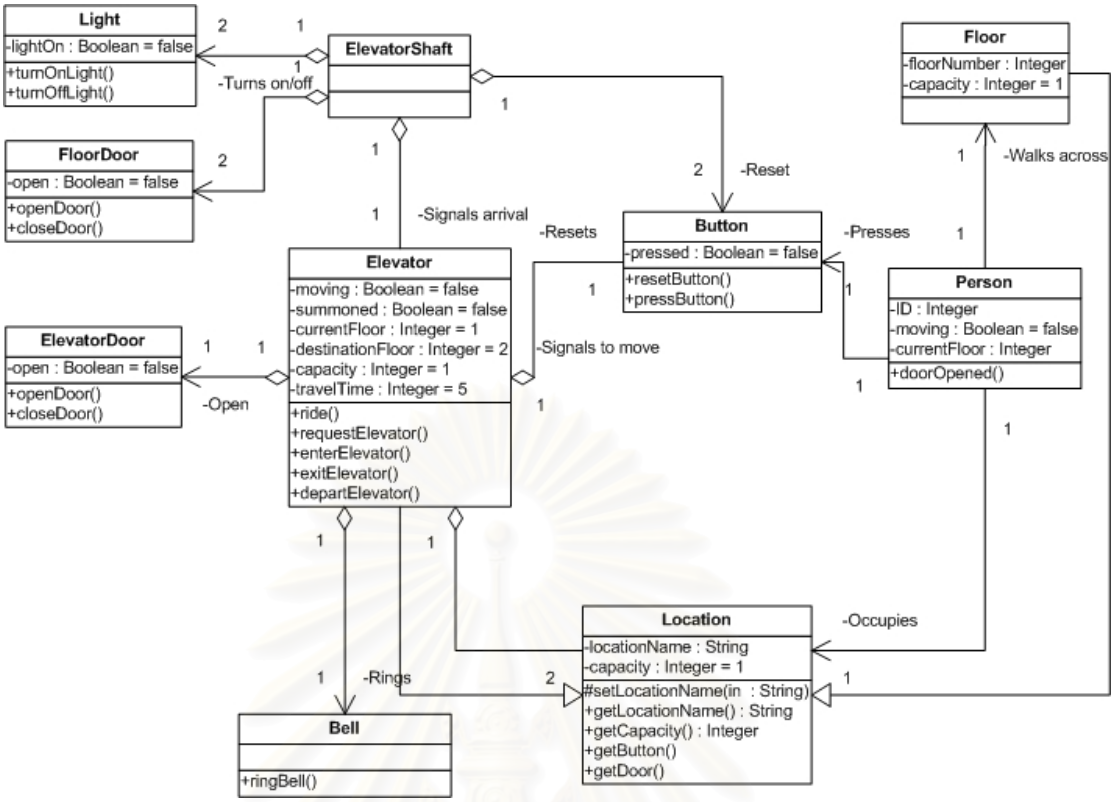
โปรแกรมแบบจำลองการทำงานของลิฟต์ แบ่งเป็น 4 แพ็กเกจ คือ แพ็กเกจแบบจำลอง (Model) แพ็กเกจแสดงผล (View) แพ็กเกจควบคุม (Controller) และแพ็กเกจเหตุการณ์ (Event) ซึ่งสามารถแสดงความสัมพันธ์ระหว่างแพ็กเกจต่าง ๆ ของระบบดังรูปที่ ค.1 ซึ่งรายละเอียดของความสัมพันธ์ระหว่างคลาสของแพ็จเกจต่างๆ รายละเอียดแสดงดังรูปที่ ง.2, ง.3, ง.4 และ ง.5 ตามลำดับ



รูปที่ ค.1 แผนภาพคลาสแสดงความสัมพันธ์ระหว่างแพ็กเกจต่าง ๆ ของโปรแกรมที่ 3

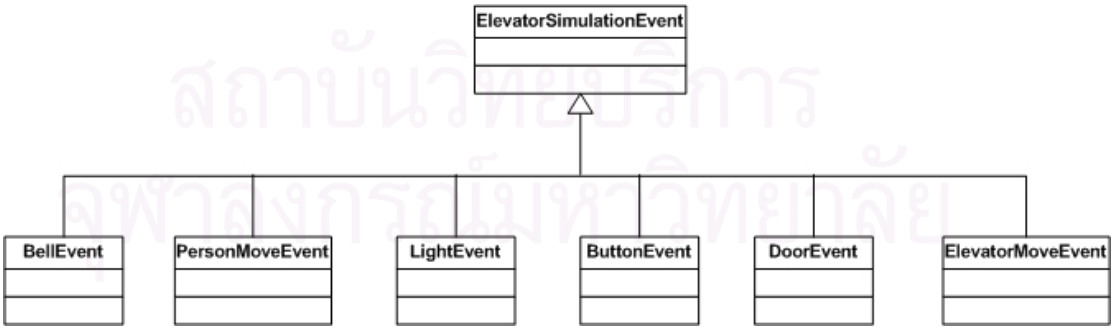


รูปที่ ค.2 แผนภาพคลาสแสดงความสัมพันธ์ภายในแพ็กเกจแสดงผล



รูปที่ ค.3 แผนภาพคลาสแสดงความสัมพันธ์ภายในแพ็กเกจแบบจำลอง

รูปที่ ค.4 แผนภาพคลาสแสดงความสัมพันธ์ภายในแพ็กเกจควบคุม



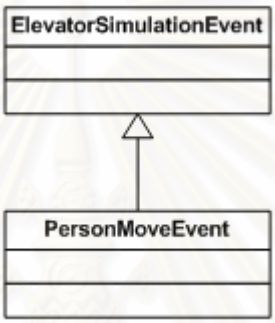
รูปที่ ค.5 แผนภาพคลาสแสดงความสัมพันธ์ภายในแพ็กเกจเหตุการณ์

ค.2.1 ซอร์สโค้ดของโปรแกรมที่ 3 หลังจากประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 1

```
…
package com.deitel.jhtp5.elevator.event;
import com.deitel.jhtp5.elevator.ElevatorConstants;
import com.deitel.jhtp5.elevator.model.Floor;
import com.deitel.jhtp5.elevator.model.ElevatorShaft;
import com.deitel.jhtp5.elevator.model.Person;
import java.util.Iterator;
import java.util.Set;
import java.util.HashSet;
public class ElevatorSimulationConstants implements ElevatorConstants {
  // declare two-Floor architecture in simulation
  public Floor firstFloor; //move to new class
  public Floor secondFloor; //move to new class
  private Set personMoveListeners;
  public ElevatorSimulationConstants() {
    // instantiate firstFloor and secondFloor objects
    firstFloor = new Floor( FIRST_FLOOR_NAME );
    secondFloor = new Floor( SECOND_FLOOR_NAME );
    // instantiate Set for ElevatorMoveListener objects
    personMoveListeners = new HashSet( 1 );
  }
  // return Floor with given name
  public Floor getFloor( String name )  {
     if ( name.equals( FIRST_FLOOR_NAME ) )
        return firstFloor;
     else
        if ( name.equals( SECOND_FLOOR_NAME ) )
           return secondFloor;
        else
           return null;
  } // end method getFloor
  // send PersonMoveEvent to listener, depending on event type
  public void sendPersonMoveEvent(int eventType, PersonMoveEvent event )  {
     Iterator iterator = personMoveListeners.iterator();
     while ( iterator.hasNext() ) {
        PersonMoveListener listener =( PersonMoveListener ) iterator.next();
        // send Event to this listener, depending on eventType
        switch ( eventType ) {
           // Person has been created
           case Person.PERSON_CREATED:
              listener.personCreated( event );
              break;
           // Person arrived at Elevator
           case Person.PERSON_ARRIVED:
              listener.personArrived( event );
              break;
           // Person entered Elevator
           case Person.PERSON_ENTERING_ELEVATOR:
              listener.personEntered( event );
              break;
           // Person pressed Button object
           case Person.PERSON_PRESSING_BUTTON:
              listener.personPressedButton( event );
              break;
           // Person exited Elevator
           case Person.PERSON_EXITING_ELEVATOR:
              listener.personDeparted( event );
              break;
           // Person exited simulation
           case Person.PERSON_EXITED:
              listener.personExited( event );
              break;
           default:
              break;
        }
     }
  } // end method sendPersonMoveEvent
  // set listener for PersonMoveEvents
  public void addPersonMoveListener(PersonMoveListener listener ){
     personMoveListeners.add( listener );
  }
}
…
```

ค.2.2 แผนภาพคลาสของโปรแกรมที่ 3 หลังจากประยุกต์ใช้วิธีรีแฟคทอริงในกรณีที่ 2

วิธีรีแฟคทอริงในกรณีที่ 2 เป็นการประยุกต์ใช้วิธี Collapse Hierarchy คือ ย้ายคุณสมบัติภายในคลาส BellEvent, ButtonEvent, DoorEvent, ElevatorMoveEvent และ LightEvent ไปยังคลาสแม่ คือคลาส ElevatorSimulationEvent ภายในแพ็กเกจเหตุการณ์ แต่คุณสมบัติภายในคลาสดังกล่าว ไม่มีเมทธอดและคุณลักษณะอื่นที่ประกาศเพิ่ม มีเพียงการเรียกใช้คอนสตรัคเตอร์ของคลาสแม่เท่านั้น จึงลบคลาสทั้ง 5 ออก รายละเอียดแสดงในรูปที่ ค.6 และเปลี่ยนการเรียกใช้คลาสเหล่านั้น ไปเรียกใช้คลาสแม่แทน ดังนั้นซอร์สโค้ดของคลาสอื่น ภายในแพ็กเกจอื่นๆ จึงไม่มีการเปลี่ยนแปลง



รูปที่ ค.6 แผนภาพคลาสแสดงความสัมพันธ์ภายในแพ็กเกจเหตุการณ์

ภาคผนวก ง
ผลของค่ามาตรวัดร่องรอยที่ไม่ดีของโปรแกรมที่ 3

ตารางที่ ง.1 ค่ามาตรวัดสำหรับคลาสของโปรแกรมที่ 3

คลาส	มาตรวัด			
	NIM	*NIV*	*TCC*	*DIT*
AnimatedPanel	14	10	0.077	3
Bell	4	1	0.167	0
BellEvent	**1**	**0**	**0**	**1**
Button	6	2	0.2	0
ButtonEvent	**1**	**0**	**0**	**1**
Door	6	4	0.4	0
DoorEvent	**1**	**0**	**0**	**1**
Elevator	20	15	0.058	1
ElevatorController	1	3	0	1
ElevatorDoor	4	0	0	1
ElevatorMoveEvent	**1**	**0**	**0**	**1**
ElevatorShaft	18	12	0.037	0
ElevatorSimulation	**26**	**10**	**0.034**	**0**
ElevatorSimulationEvent	5	2	0.334	0
ElevatorView	28	56	0.043	1
Floor	5	1	0.167	1
ImagePanel	11	4	0.111	1
Light	6	4	0.2	0
LightEvent	**1**	**0**	**0**	**1**
Location	4	1	0.167	0
MovingPanel	7	3	0.2	2
Person	10	11	0.139	1
PersonMoveEvent	2	1	0	1
SoundEffects	3	1	1	0
AnimatedPanel	14	10	0.077	3

ตารางที่ ง.2 ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 3

		มาตรวัด *NCDM + NCDA*											
		Animated Panel	Bell	BellEvent	Button	Button Event	Door	Door Event	Elevator	Elevator Door	Elevator MoveEvent	Elevator Shaft	Elevator Simulation
Animated Panel	animate	7	0	0	0	0	0	0	0	0	0	0	0
	determineNextFrame	11	0	0	0	0	0	0	0	0	0	0	0
	addFrameSequence	1	0	0	0	0	0	0	0	0	0	0	0
	isAnimating	1	0	0	0	0	0	0	0	0	0	0	0
	setAnimating	2	0	0	0	0	0	0	0	0	0	0	0
	setCurrentFrame	1	0	0	0	0	0	0	0	0	0	0	0
	setAnimationRate	1	0	0	0	0	0	0	0	0	0	0	0
	getAnimationRate	1	0	0	0	0	0	0	0	0	0	0	0
	setLoop	1	0	0	0	0	0	0	0	0	0	0	0
	isLoop	1	0	0	0	0	0	0	0	0	0	0	0
	isDisplayLastFrame	1	0	0	0	0	0	0	0	0	0	0	0
	setDisplayLastFrame	1	0	0	0	0	0	0	0	0	0	0	0
	playAnimation	3	0	0	0	0	0	0	0	0	0	0	0
Bell	ringBell	0	2	0	0	0	0	0	0	0	0	0	0
	setBellListener	0	1	0	0	0	0	0	0	0	0	0	0
	elevatorDeparted	0	0	0	0	0	0	0	0	0	0	0	0
	elevatorArrived	0	1	0	0	0	0	0	0	0	0	0	0
BellEvent	bellRang	0	0	0	0	0	0	0	0	0	0	0	0

ตารางที่ ง.2 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 3

		มาตรวัด *NCDM + NCDA*											
		Animated Panel	Bell	BellEvent	Button	Button Event	Door	Door Event	Elevator	Elevator Door	Elevator MoveEvent	Elevator Shaft	Elevator Simulation
Button	setButtonListener	0	0	0	1	0	0	0	0	0	0	0	0
	pressButton	0	0	0	2	0	0	0	0	0	0	0	0
	resetButton	0	0	0	2	0	0	0	0	0	0	0	0
	isButtonPressed	0	0	0	1	0	0	0	0	0	0	0	0
	elevatorDeparted	0	0	0	0	0	0	0	0	0	0	0	0
	elevatorArrived	0	0	0	1	0	0	0	0	0	0	0	0
ButtonEvent	buttonPressed	0	0	0	0	0	0	0	0	0	0	0	0
	buttonReset	0	0	0	0	0	0	0	0	0	0	0	0
Door	addDoorListener	0	0	0	0	0	2	0	0	0	0	0	0
	removeDoorListener	0	0	0	0	0	2	0	0	0	0	0	0
	openDoor	0	0	0	0	0	5	0	0	0	0	0	0
	closeDoor	0	0	0	0	0	4	0	0	0	0	0	0
	isDoorOpen	0	0	0	0	0	1	0	0	0	0	0	0
DoorEvent	doorOpened	0	0	0	0	0	0	0	0	0	0	0	0
	doorClosed	0	0	0	0	0	0	0	0	0	0	0	0
Elevator	changeFloors	0	0	0	0	0	0	0	4	0	0	0	0
	start	0	0	0	0	0	0	0	4	0	0	0	0
	stopElevator	0	0	0	0	0	0	0	1	0	0	0	0
	run	0	0	0	0	0	0	0	17	0	0	0	0

ตารางที่ ง.2 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 3

		มาตรวัด *NCDM + NCDA*											
		Animated Panel	Bell	BellEvent	Button	Button Event	Door	Door Event	Elevator	Elevator Door	Elevator MoveEvent	Elevator Shaft	Elevator Simulation
Elevator	pauseThread	0	0	0	0	0	0	0	0	0	0	0	0
	getButton	0	0	0	0	0	0	0	1	0	0	0	0
	getDoor	0	0	0	0	0	0	0	1	0	0	0	0
	setMoving	0	0	0	0	0	0	0	1	0	0	0	0
	isMoving	0	0	0	0	0	0	0	1	0	0	0	0
	isElevatorRunning	0	0	0	0	0	0	0	1	0	0	0	0
	addElevatorMoveListener	0	0	0	0	0	0	0	1	0	0	0	0
	setBellListener	0	0	0	0	0	0	0	1	0	0	0	0
	setButtonListener	0	0	0	0	0	0	0	1	0	0	0	0
	setDoorListener	0	0	0	0	0	0	0	1	0	0	0	0
	sendArrivalEvent	0	0	0	0	0	0	0	4	0	0	0	0
	sendDepartureEvent	0	0	0	0	0	0	0	2	0	0	0	0
	requestElevator	0	0	0	0	0	0	0	7	0	0	0	0
	bellRang	0	0	0	0	0	0	0	2	0	0	0	0
	getCurrentFloor	0	0	0	0	0	0	0	1	0	0	0	0
ElevatorDoor	openDoor	0	0	0	0	0	0	0	0	0	0	0	0
	closeDoor	0	0	0	0	0	0	0	0	0	0	0	0
	elevatorDeparted	0	0	0	0	0	0	0	0	0	0	0	0
	elevatorArrived	0	0	0	0	0	0	0	0	1	0	0	0

ตารางที่ ง.2 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 3

		มาตรวัด *NCDM + NCDA*											
		Animated Panel	Bell	BellEvent	Button	Button Event	Door	Door Event	Elevator	Elevator Door	Elevator MoveEvent	Elevator Shaft	Elevator Simulation
ElevatorMoveEvent	elevatorDeparted	0	0	0	0	0	0	0	0	0	0	0	0
	elevatorArrived	0	0	0	0	0	0	0	0	0	0	0	0
ElevatorShaft	getElevator	0	0	0	0	0	0	0	0	0	0	1	0
	getFirstFloorDoor	0	0	0	0	0	0	0	0	0	0	1	0
	getSecondFloorDoor	0	0	0	0	0	0	0	0	0	0	1	0
	getFirstFloorButton	0	0	0	0	0	0	0	0	0	0	1	0
	getSecondFloorButton	0	0	0	0	0	0	0	0	0	0	1	0
	getFirstFloorLight	0	0	0	0	0	0	0	0	0	0	1	0
	getSecondFloorLight	0	0	0	0	0	0	0	0	0	0	1	0
	bellRang	0	0	0	0	0	0	0	0	0	0	1	0
	lightTurnedOn	0	0	0	0	0	0	0	0	0	0	1	0
	lightTurnedOff	0	0	0	0	0	0	0	0	0	0	1	0
	elevatorDeparted	0	0	0	0	0	0	0	0	0	0	1	0
	elevatorArrived	0	0	0	0	0	0	0	0	0	0	1	0
	setDoorListener	0	0	0	0	0	0	0	0	0	0	1	0
	setButtonListener	0	0	0	0	0	0	0	0	0	0	1	0
	addElevatorMoveListener	0	0	0	0	0	0	0	0	0	0	1	0
	setLightListener	0	0	0	0	0	0	0	0	0	0	1	0
	setBellListener	0	0	0	0	0	0	0	0	0	0	1	0

ตารางที่ ง.2 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 3

		มาตรวัด *NCDM + NCDA*											
		Animated Panel	Bell	BellEvent	Button	Button Event	Door	Door Event	Elevator	Elevator Door	Elevator MoveEvent	Elevator Shaft	Elevator Simulation
ElevatorSimulation	getFloor	0	0	0	0	0	0	0	0	0	0	0	2
	addperson	0	0	0	0	0	0	0	0	0	0	0	4
	elevatorDeparted	0	0	0	0	0	0	0	0	0	0	0	1
	elevatorArrived	0	0	0	0	0	0	0	0	0	0	0	1
	sendPersonMoveEvent	0	0	0	0	0	0	0	0	0	0	0	1
	personCreated	0	0	0	0	0	0	0	0	0	0	0	1
	personArrived	0	0	0	0	0	0	0	0	0	0	0	1
	personPressedButton	0	0	0	0	0	0	0	0	0	0	0	1
	personEntered	0	0	0	0	0	0	0	0	0	0	0	1
	personDeparted	0	0	0	0	0	0	0	0	0	0	0	1
	personExited	0	0	0	0	0	0	0	0	0	0	0	1
	doorOpened	0	0	0	0	0	0	0	0	0	0	0	1
	doorClosed	0	0	0	0	0	0	0	0	0	0	0	1
	buttonPressed	0	0	0	0	0	0	0	0	0	0	0	1
	buttonReset	0	0	0	0	0	0	0	0	0	0	0	1
	bellRang	0	0	0	0	0	0	0	0	0	0	0	1
	lightTurnedOn	0	0	0	0	0	0	0	0	0	0	0	1
	lightTurnedOff	0	0	0	0	0	0	0	0	0	0	0	1
	setElevatorSimulationListener	0	0	0	0	0	0	0	0	0	0	0	6

ตารางที่ ง.2 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 3

		มาตรวัด *NCDM + NCDA*											
		Animated Panel	Bell	BellEvent	Button	Button Event	Door	Door Event	Elevator	Elevator Door	Elevator MoveEvent	Elevator Shaft	Elevator Simulation
ElevatorSimulation	addPersonMoveListener	0	0	0	0	0	0	0	0	0	0	0	1
	setDoorListener	0	0	0	0	0	0	0	0	0	0	0	1
	setButtonListener	0	0	0	0	0	0	0	0	0	0	0	1
	setElevatorMoveListener	0	0	0	0	0	0	0	0	0	0	0	1
	setLightListener	0	0	0	0	0	0	0	0	0	0	0	1
	setBellListener	0	0	0	0	0	0	0	0	0	0	0	1
ElevatorSimulationEvent	setLocation	0	0	0	0	0	0	0	0	0	0	0	0
	getLocation	0	0	0	0	0	0	0	0	0	0	0	0
	setSource	0	0	0	0	0	0	0	0	0	0	0	0
	getSource	0	0	0	0	0	0	0	0	0	0	0	0
ElevatorView	instantiatePanels	0	0	0	0	0	0	0	0	0	0	0	0
	placePanelsOnView	0	0	0	0	0	0	0	0	0	0	0	0
	initializeAudio	0	0	0	0	0	0	0	0	0	0	0	0
	startAnimation	0	0	0	0	0	0	0	0	0	0	0	0
	stopAnimation	0	0	0	0	0	0	0	0	0	0	0	0
	actionPerformed	0	0	0	0	0	0	0	0	0	0	0	0
	getPersonAnimatedPanelsIterator	0	0	0	0	0	0	0	0	0	0	0	0
	stopWalkingSound	0	0	0	0	0	0	0	0	0	0	0	0
	getPersonPanel	0	0	0	0	0	0	0	0	0	0	0	0

ตารางที่ ง.2 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 3

		มาตรวัด *NCDM + NCDA*										
		Elevator SimulationEvent	Elevator View	Floor	Image Panel	Light	Light Event	Location	Moving Panel	Person	Person MoveEvent	Sound Effects
ElevatorSimulation	getFloor	0	0	0	0	0	0	0	0	0	0	0
	addperson	0	0	0	0	0	0	0	0	0	0	0
	elevatorDeparted	0	0	0	0	0	0	0	0	0	0	0
	elevatorArrived	0	0	0	0	0	0	0	0	0	0	0
	sendPersonMoveEvent	0	0	0	0	0	0	0	0	6	0	0
	personCreated	0	0	0	0	0	0	0	0	1	0	0
	personArrived	0	0	0	0	0	0	0	0	1	0	0
	personPressedButton	0	0	0	0	0	0	0	0	1	0	0
	personEntered	0	0	0	0	0	0	0	0	1	0	0
	personDeparted	0	0	0	0	0	0	0	0	1	0	0
	personExited	0	0	0	0	0	0	0	0	1	0	0
	doorOpened	0	0	0	0	0	0	0	0	0	0	0
	doorClosed	0	0	0	0	0	0	0	0	0	0	0
	buttonPressed	0	0	0	0	0	0	0	0	0	0	0
	buttonReset	0	0	0	0	0	0	0	0	0	0	0
	bellRang	0	0	0	0	0	0	0	0	0	0	0
	lightTurnedOn	0	0	0	0	0	0	0	0	0	0	0
	lightTurnedOff	0	0	0	0	0	0	0	0	0	0	0
	setElevatorSimulationListener	0	0	0	0	0	0	0	0	0	0	0

ตารางที่ ง.2 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 3

		มาตรวัด *NCDM + NCDA*										
		Elevator SimulationEvent	Elevator View	Floor	Image Panel	Light	Light Event	Location	Moving Panel	Person	Person MoveEvent	Sound Effects
ElevatorSimulation	addPersonMoveListener	0	0	0	0	0	0	0	0	0	0	0
	setDoorListener	0	0	0	0	0	0	0	0	0	0	0
	setButtonListener	0	0	0	0	0	0	0	0	0	0	0
	setElevatorMoveListener	0	0	0	0	0	0	0	0	0	0	0
	setLightListener	0	0	0	0	0	0	0	0	0	0	0
	setBellListener	0	0	0	0	0	0	0	0	0	0	0
ElevatorSimulationEvent	setLocation	1	0	0	0	0	0	0	0	0	0	0
	getLocation	1	0	0	0	0	0	0	0	0	0	0
	setSource	1	0	0	0	0	0	0	0	0	0	0
	getSource	1	0	0	0	0	0	0	0	0	0	0
ElevatorView	instantiatePanels	0	89	0	0	0	0	0	0	0	0	0
	placePanelsOnView	0	18	0	0	0	0	0	0	0	0	0
	initializeAudio	0	14	0	0	0	0	0	0	0	0	0
	startAnimation	0	6	0	0	0	0	0	0	0	0	0
	stopAnimation	0	1	0	0	0	0	0	0	0	0	0
	actionPerformed	0	4	0	0	0	0	0	0	0	0	0
	getPersonAnimatedPanelsIterator	0	2	0	0	0	0	0	0	0	0	0
	stopWalkingSound	0	3	0	0	0	0	0	0	0	0	0
	getPersonPanel	0	1	0	0	0	0	0	0	0	0	0

ตารางที่ ง.2 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 3

		มาตรวัด *NCDM + NCDA*										
		Elevator SimulationEvent	Elevator View	Floor	Image Panel	Light	Light Event	Location	Moving Panel	Person	Person MoveEvent	Sound Effects
ElevatorView	elevatorDeparted	0	15	0	0	0	0	0	0	0	0	0
	elevatorArrived	0	9	0	0	0	0	0	0	0	0	0
	personCreated	0	12	0	0	0	0	0	0	0	0	0
	personArrived	0	3	0	0	0	0	0	0	0	0	0
	personPressedButton	0	2	0	0	0	0	0	0	0	0	0
	personEntered	0	5	0	0	0	0	0	0	0	0	0
	personDeparted	0	12	0	0	0	0	0	0	0	0	0
	personExited	0	4	0	0	0	0	0	0	0	0	0
	doorOpened	0	5	0	0	0	0	0	0	0	0	0
	doorClosed	0	5	0	0	0	0	0	0	0	0	0
	buttonPressed	0	8	0	0	0	0	0	0	0	0	0
	buttonReset	0	9	0	0	0	0	0	0	0	0	0
	bellRang	0	3	0	0	0	0	0	0	0	0	0
	lightTurnedOn	0	3	0	0	0	0	0	0	0	0	0
	lightTurnedOff	0	3	0	0	0	0	0	0	0	0	0
	getPreferredSize	0	2	0	0	0	0	0	0	0	0	0
	getMinimumSize	0	1	0	0	0	0	0	0	0	0	0
	getMaximumSize	0	1	0	0	0	0	0	0	0	0	0
Floor	getButton	0	0	2	0	0	0	0	0	0	0	0

ตารางที่ ง.2 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 3

		มาตรวัด *NCDM + NCDA*										
		Elevator SimulationEvent	Elevator View	Floor	Image Panel	Light	Light Event	Location	Moving Panel	Person	Person MoveEvent	Sound Effects
Floor	getDoor	0	0	2	0	0	0	0	0	0	0	0
	getElevatorShaft	0	0	1	0	0	0	0	0	0	0	0
	setElevatorShaft	0	0	1	0	0	0	0	0	0	0	0
ImagePanel	paintComponent	0	0	0	1	0	0	0	0	0	0	0
	add	0	0	0	1	0	0	0	0	0	0	0
	add	0	0	0	1	0	0	0	0	0	0	0
	remove	0	0	0	1	0	0	0	0	0	0	0
	setIcon	0	0	0	1	0	0	0	0	0	0	0
	setPosition	0	0	0	1	0	0	0	0	0	0	0
	getID	0	0	0	1	0	0	0	0	0	0	0
	getPosition	0	0	0	1	0	0	0	0	0	0	0
	getImageIcon	0	0	0	1	0	0	0	0	0	0	0
	getChildren	0	0	0	1	0	0	0	0	0	0	0
Light	setLightListener	0	0	0	0	1	0	0	0	0	0	0
	turnOnLight	0	0	0	0	4	0	0	0	0	0	0
	turnOffLight	0	0	0	0	3	0	0	0	0	0	0
	isLightOn	0	0	0	0	1	0	0	0	0	0	0
	elevatorDeparted	0	0	0	0	1	0	0	0	0	0	0
	elevatorArrived	0	0	0	0	1	0	0	0	0	0	0

ตารางที่ ง.2 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 3

		มาตรวัด *NCDM + NCDA*										
		Elevator SimulationEvent	Elevator View	Floor	Image Panel	Light	Light Event	Location	Moving Panel	Person	Person MoveEvent	Sound Effects
LightEvent	lightTurnedOn	0	0	0	0	0	0	0	0	0	0	0
	lightTurnedOff	0	0	0	0	0	0	0	0	0	0	0
Location	setLocationName	0	0	0	0	0	0	1	0	0	0	0
	getLocationName	0	0	0	0	0	0	1	0	0	0	0
	getButton	0	0	0	0	0	0	0	0	0	0	0
	getDoor	0	0	0	0	0	0	0	0	0	0	0
MovingPanel	animate	0	0	0	0	0	0	0	3	0	0	0
	isMoving	0	0	0	0	0	0	0	1	0	0	0
	setMoving	0	0	0	0	0	0	0	1	0	0	0
	setVelocity	0	0	0	0	0	0	0	2	0	0	0
	getXVelocity	0	0	0	0	0	0	0	1	0	0	0
	getYVelocity	0	0	0	0	0	0	0	1	0	0	0
Person	setPersonMoveListener	0	0	0	0	0	0	0	0	1	0	0
	setLocation	0	0	0	0	0	0	0	0	1	0	0
	getLocation	0	0	0	0	0	0	0	0	1	0	0
	getID	0	0	0	0	0	0	0	0	1	0	0
	setMoving	0	0	0	0	0	0	0	0	1	0	0
	isMoving	0	0	0	0	0	0	0	0	1	0	0
	run	0	0	0	0	0	0	0	0	35	0	0

ตารางที่ ง.2 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 3

		มาตรวัด *NCDM + NCDA*										
		Elevator SimulationEvent	Elevator View	Floor	Image Panel	Light	Light Event	Location	Moving Panel	Person	Person MoveEvent	Sound Effects
Person	pauseThread	0	0	0	0	0	0	0	0	0	0	0
	sendPersonMoveEvent	0	0	0	0	0	0	0	0	14	0	0
PersonMoveEvent	getID	0	0	0	0	0	0	0	0	0	1	0
	personCreated	0	0	0	0	0	0	0	0	0	0	0
	personArrived	0	0	0	0	0	0	0	0	0	0	0
	personDeparted	0	0	0	0	0	0	0	0	0	0	0
	personPressedButton	0	0	0	0	0	0	0	0	0	0	0
	personEntered	0	0	0	0	0	0	0	0	0	0	0
	personExited	0	0	0	0	0	0	0	0	0	0	0
SoundEffects	getAudioClip	0	0	0	0	0	0	0	0	0	0	1

ตารางที่ ง.2 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 3

Class	Method	*NOS*	*NOP*	*NOT*	*NOSS*
Animated Panel	animate	6	0	0	0
	determineNextFrame	**8**	0	1	0
	addFrameSequence	1	1	0	0
	isAnimating	1	0	0	0
	setAnimating	1	1	0	0
	setCurrentFrame	1	1	0	0
	setAnimationRate	1	1	0	0
	getAnimationRate	1	0	0	0
	setLoop	1	1	0	0
	isLoop	1	0	0	0
	isDisplayLastFrame	1	0	0	0
	setDisplayLastFrame	1	1	0	0
	playAnimation	3	1	0	0
Bell	ringBell	2	1	0	0
	setBellListener	1	1	0	0
	elevatorDeparted	0	1	0	0
	elevatorArrived	1	1	0	0
BellEvent	bellRang	1	1	0	0
Button	setButtonListener	1	1	0	0
	pressButton	2	1	0	0
	resetButton	2	1	0	0
	isButtonPressed	1	0	0	0
	elevatorDeparted	0	1	0	0
	elevatorArrived	1	1	0	0
ButtonEvent	buttonPressed	1	1	0	0
	buttonReset	1	1	0	0
Door	addDoorListener	1	1	0	0
	removeDoorListener	1	1	0	0
	openDoor	**8**	1	3	0
	closeDoor	5	1	2	0
	isDoorOpen	1	0	0	0
DoorEvent	doorOpened	1	1	0	0
	doorClosed	1	1	0	0
Elevator	changeFloors	2	0	1	0
	start	4	0	0	0
	stopElevator	1	0	0	0
	run	**11**	0	0	0
	pauseThread	1	**2**	0	0

ตารางที่ ง.2 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 3

Class	Method	*NOS*	*NOP*	*NOT*	*NOSS*
Elevator	getButton	1	0	0	0
	getDoor	1	0	0	0
	setMoving	1	1	0	0
	isMoving	1	0	0	0
	isElevatorRunning	1	0	0	0
	addElevatorMoveListener	1	1	0	0
	setBellListener	1	1	0	0
	setButtonListener	1	1	0	0
	setDoorListener	1	1	0	0
	sendArrivalEvent	5	1	2	0
	sendDepartureEvent	2	1	2	0
	requestElevator	6	1	0	0
	bellRang	2	1	0	0
	getCurrentFloor	1	0	3	0
ElevatorDoor	openDoor	2	1	0	0
	closeDoor	2	1	0	0
ElevatorMoveEvent	elevatorDeparted	0	1	0	0
	elevatorArrived	1	1	0	0
	elevatorDeparted	1	1	0	0
	elevatorArrived	1	1	0	0
ElevatorShaft	getElevator	1	0	0	0
	getFirstFloorDoor	1	0	0	0
	getSecondFloorDoor	1	0	0	0
	getFirstFloorButton	1	0	0	0
	getSecondFloorButton	1	0	0	0
	getFirstFloorLight	1	0	0	0
	getSecondFloorLight	1	0	0	0
	bellRang	1	1	0	0
	lightTurnedOn	1	1	0	0
	lightTurnedOff	1	1	0	0
	elevatorDeparted	2	1	2	0
	elevatorArrived	2	1	2	0
	setDoorListener	1	1	0	0
	setButtonListener	1	1	0	0
	addElevatorMoveListener	1	1	0	0
	setLightListener	1	1	0	0
	setBellListener	1	1	0	0

ตารางที่ ง.2 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 3

Class	Method	*NOS*	*NOP*	*NOT*	*NOSS*
ElevatorSimulation	getFloor	5	1	0	0
	_dperson	4	1	1	0
	elevatorDeparted	1	1	0	0
	elevatorArrived	1	1	0	0
	sendPersonMoveEvent	**22**	**2**	2	**1**
	personCreated	1	1	0	0
	personArrived	1	1	0	0
	personPressedButton	1	1	0	0
	personEntered	1	1	0	0
	personDeparted	1	1	0	0
	personExited	1	1	0	0
	doorOpened	1	1	0	0
	doorClosed	1	1	0	0
	buttonPressed	1	1	0	0
	buttonReset	1	1	0	0
	bellRang	1	1	0	0
	lightTurnedOn	1	1	0	0
	lightTurnedOff	1	1	0	0
	setElevatorSimulationListener	6	1	0	0
	addPersonMoveListener	1	1	0	0
	setDoorListener	1	1	0	0
	setButtonListener	1	1	0	0
	setElevatorMoveListener	1	1	0	0
	setLightListener	1	1	0	0
	setBellListener	1	1	0	0
ElevatorSimulationEvent	setLocation	1	1	0	0
	getLocation	1	0	0	0
	setSource	1	1	0	0
	getSource	1	0	0	0
ElevatorView	**instantiatePanels**	**48**	0	7	0
	placePanelsOnView	**14**	0	0	0
	initializeAudio	**8**	0	1	0
	startAnimation	5	0	0	0
	stopAnimation	1	0	0	0
	actionPerformed	6	1	2	0
	getPersonAnimatedPanelsIterator	1	0	0	0
	stopWalkingSound	4	0	2	0
	getPersonPanel	4	1	2	0

ตารางที่ ง.2 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 3

Class	Method	*NOS*	*NOP*	*NOT*	*NOSS*
ElevatorView	**elevatorDeparted**	**20**	1	6	0
	elevatorArrived	6	1	2	0
	personCreated	**18**	1	11	0
	personArrived	6	1	3	0
	personPressedButton	7	1	1	0
	personEntered	5	1	3	0
	personDeparted	**15**	1	6	0
	personExited	6	1	1	0
	doorOpened	5	1	1	0
	doorClosed	5	1	1	0
	buttonPressed	**11**	1	1	0
	buttonReset	**12**	1	1	0
	bellRang	3	1	0	0
	lightTurnedOn	5	1	1	0
	lightTurnedOff	5	1	1	0
	getPreferredSize	1	0	0	0
	getMinimumSize	1	0	0	0
	getMaximumSize	1	0	0	0
Floor	getButton	5	0	0	0
	getDoor	5	0	0	0
	getElevatorShaft	1	0	0	0
	setElevatorShaft	1	1	0	0
ImagePanel	paintComponent	2	1	0	0
	add	2	1	0	0
	add	2	**2**	0	0
	remove	2	1	0	0
	setIcon	1	1	0	0
	setPosition	2	**2**	0	0
	getID	1	0	0	0
	getPosition	1	0	0	0
	getImageIcon	1	0	0	0
	getChildren	1	0	0	0
Light	setLightListener	1	1	0	0
	turnOnLight	5	1	1	0
	turnOffLight	3	1	0	0
	isLightOn	1	0	0	0
	elevatorDeparted	1	1	0	0
	elevatorArrived	1	1	0	0

ตารางที่ ง.2 (ต่อ) ค่ามาตรวัดสำหรับเมทธอดของโปรแกรมที่ 3

Class	Method	*NOS*	*NOP*	*NOT*	*NOSS*
LightEvent	lightTurnedOn	1	1	0	0
	lightTurnedOff	1	1	0	0
Location	setLocationName	1	1	0	0
	getLocationName	1	0	0	0
	getButton	1	0	0	0
	getDoor	1	0	0	0
MovingPanel	animate	4	0	4	0
	isMoving	1	0	0	0
	setMoving	1	1	0	0
	setVelocity	2	**2**	0	0
	getXVelocity	1	0	0	0
	getYVelocity	1	0	0	0
Person	setPersonMoveListener	1	1	0	0
	setLocation	1	1	0	0
	getLocation	1	0	0	0
	getID	1	0	0	0
	setMoving	1	1	0	0
	isMoving	1	0	0	0
	run	**26**	**2**	5	0
	pauseThread	1	**2**	0	0
	sendPersonMoveEvent	**21**	1	1	**1**
PersonMoveEvent	getID	1	0	0	0
	personCreated	1	1	0	0
	personArrived	1	1	0	0
	personDeparted	1	1	0	0
	personPressedButton	1	1	0	0
	personEntered	1	1	0	0
	personExited	1	1	0	0
SoundEffects	**getAudioClip**	1	**2**	0	0

ตารางที่ ง.3 ค่ามาตรวัดสำหรับคุณลักษณะของโปรแกรมที่ 3

		มาตรวัด *NCRA*									
		Animated Panel	Bell	Button	Door	Elevator	ElevatorController	ElevatorShaft	Elevator Simulation	Elevator SimulationEvent	ElevatorView
Door	AUTOMATIC_ CLOSE_DELAY	0	0	0	0	0	0	0	0	0	0
Elevator	ONE_SECOND	0	0	0	0	2	0	0	0	0	0
ElevatorSimulation	bellListener	0	0	0	0	0	0	0	2	0	0
	elevatorMoveListener	0	0	0	0	0	0	0	3	0	0
	numberOfPeople	0	0	0	0	0	0	0	3	0	0
Light	AUTOMATIC_ TURNOFF_DELAY	0	0	0	0	0	0	0	0	0	0
Person	TIME_TO_WALK	0	0	0	0	0	0	0	0	0	0
	PERSON_CREATED	0	0	0	0	0	0	0	0	2	0
	PERSON_ARRIVED	0	0	0	0	0	0	0	0	2	0
	PERSON_ENTERING _ELEVATOR	0	0	0	0	0	0	0	0	2	0
	PERSON_PRESSING _BUTTON	0	0	0	0	0	0	0	0	2	0
	PERSON_EXITING _ELEVATOR	0	0	0	0	0	0	0	0	2	0
	PERSON_EXITED	0	0	0	0	0	0	0	0	2	0

ตารางที่ ง.3 (ต่อ) ค่ามาตรวัดสำหรับคุณลักษณะของโปรแกรมที่ 3

		มาตรวัด *NCRA*							
		Floor	ImagePanel	Light	Location	MovingPanel	Person	Person MoveEvent	SoundEffects
Door	AUTOMATIC_CLOSE_DELAY	0	0	0	0	0	0	0	0
Elevator	ONE_SECOND	0	0	0	0	0	0	0	0
ElevatorSimulation	bellListener	0	0	0	0	0	0	0	0
	elevatorMoveListener	0	0	0	0	0	0	0	0
	numberOfPeople	0	0	0	0	0	0	0	0
Light	AUTOMATIC_ TURNOFF_DELAY	0	0	0	0	0	0	0	0
Person	TIME_TO_WALK	0	0	0	0	0	2	0	0
	PERSON_CREATED	0	0	0	0	0	2	0	0
	PERSON_ARRIVED	0	0	0	0	0	2	0	0
	PERSON_ENTERING _ELEVATOR	0	0	0	0	0	2	0	0
	PERSON_PRESSING _BUTTON	0	0	0	0	0	3	0	0
	PERSON_EXITING _ELEVATOR	0	0	0	0	0	2	0	0
	PERSON_EXITED	0	0	0	0	0	2	0	0

ตารางที่ ง.4 ประเภทร่องรอยที่ไม่ดีที่พบในโปรแกรมที่ 3 และวิธีรีแฟคทอริงที่เสนอ

ประเภท	ร่องรอยที่ไม่ดี	วิธีรีแฟคทอริง
Long Method	เมทธอด determineNextFrame ในคลาส AnimatedPanel	Replace Temp with Query Decompose Conditional
	เมทธอด openDoor ในคลาส Door	Extract Method Replace Method with Method Object Introduce Parameter Object Preserve Whole Object Replace Temp with Query Decompose Conditional
	เมทธอด run ในคลาส Elevator	Decompose Conditional
	เมทธอด sendPersonMoveEvent ในคลาส ElevatorSimulation	Extract Method Replace Method with Method Object Introduce Parameter Object Preserve Whole Object Replace Temp with Query Decompose Conditional
	เมทธอด instantiatePanels ในคลาส ElevatorView	Replace Temp with Query
	เมทธอด placePanelsOnView ในคลาส ElevatorView	
	เมทธอด initializeAudio ในคลาส ElevatorView	Replace Temp with Query
	เมทธอด elevatorDeparted ในคลาส ElevatorView	Extract Method Replace Method with Method Object Introduce Parameter Object Preserve Whole Object Replace Temp with Query Decompose Conditional
	เมทธอด personCreated ในคลาส ElevatorView	Extract Method Replace Method with Method Object Introduce Parameter Object Preserve Whole Object Replace Temp with Query Decompose Conditional
	เมทธอด personDeparted ในคลาส ElevatorView	Extract Method Replace Method with Method Object Introduce Parameter Object Preserve Whole Object Replace Temp with Query Decompose Conditional

ตารางที่ ง.4 (ต่อ) ประเภทร่องรอยที่ไม่ดีที่พบในโปรแกรมที่ 3 และวิธีรีแฟคทอริงที่เสนอ

ประเภท	ร่องรอยที่ไม่ดี	วิธีรีแฟคทอริง
Long Method	เมทธอด buttonPressed ในคลาส ElevatorView	Extract Method Replace Method with Method Object Introduce Parameter Object Preserve Whole Object Replace Temp with Query Decompose Conditional
	เมทธอด run ในคลาส Person	Extract Method Replace Method with Method Object Introduce Parameter Object Preserve Whole Object Replace Temp with Query Decompose Conditional
	เมทธอด sendPersonMoveEvent ในคลาส Person	Extract Method Replace Method with Method Object Introduce Parameter Object Preserve Whole Object Replace Temp with Query Decompose Conditional
Long Parameter List	เมทธอด pauseThread ในคลาส Elevator	Replace Parameters with Method Preserve Whole Object Introduce Parameter Object
	เมทธอด sendPersonMoveEvent ในคลาส ElevatorSimulation	Replace Parameters with Method Preserve Whole Object Introduce Parameter Object
	เมทธอด add ในคลาส ImagePanel	Replace Parameters with Method Preserve Whole Object Introduce Parameter Object
	เมทธอด setPosition ในคลาส ImagePanel	Replace Parameters with Method Preserve Whole Object Introduce Parameter Object
	เมทธอด setVelocity ในคลาส MovingPanel	Replace Parameters with Method Preserve Whole Object Introduce Parameter Object
	เมทธอด run ในคลาส Person	Replace Parameters with Method Preserve Whole Object Introduce Parameter Object
	เมทธอด pauseThread ในคลาส Person	Replace Parameters with Method Preserve Whole Object Introduce Parameter Object

ตารางที่ ง.4 (ต่อ) ประเภทร่องรอยที่ไม่ดีที่พบในโปรแกรมที่ 3 และวิธีรีแฟคทอริงที่เสนอ

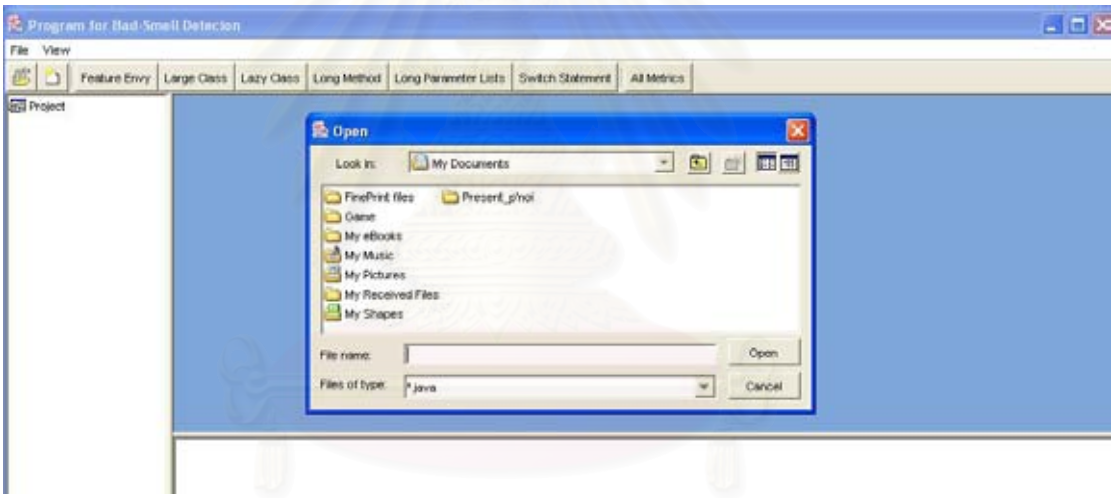
ประเภท	ร่องรอยที่ไม่ดี	วิธีรีแฟคทอริง
Long Parameter List	เมทธอด getAudioClip ในคลาส SoundEffects	Replace Parameters with Method Preserve Whole Object Introduce Parameter Object
Switch Statement	เมทธอด sendPersonMoveEven ในคลาส ElevatorSimulation	Extract Method Replace Conditional with Polymorphism Replace Parameter with Explicit Methods Introduce null object
	เมทธอด sendPersonMoveEvent ในคลาส Person	Extract Method Replace Conditional with Polymorphism Replace Parameter with Explicit Methods Introduce null object

ภาคผนวก จ
การใช้งานเครื่องมือเพื่อช่วยในการตรวจจับร่องรอยที่ไม่ดี

ในภาคผนวกนี้ จะอธิบายถึงการใช้งานเครื่องมือเพื่อช่วยในการตรวจจับร่องรอยที่ไม่ดี ซึ่งแบ่งเป็น 2 ส่วนคือ การอ่านโปรแกรมต้นฉบับ การดูค่ามาตรวัดและวิธีการรีแฟคทอริง ดังรายละเอียดต่อไปนี้

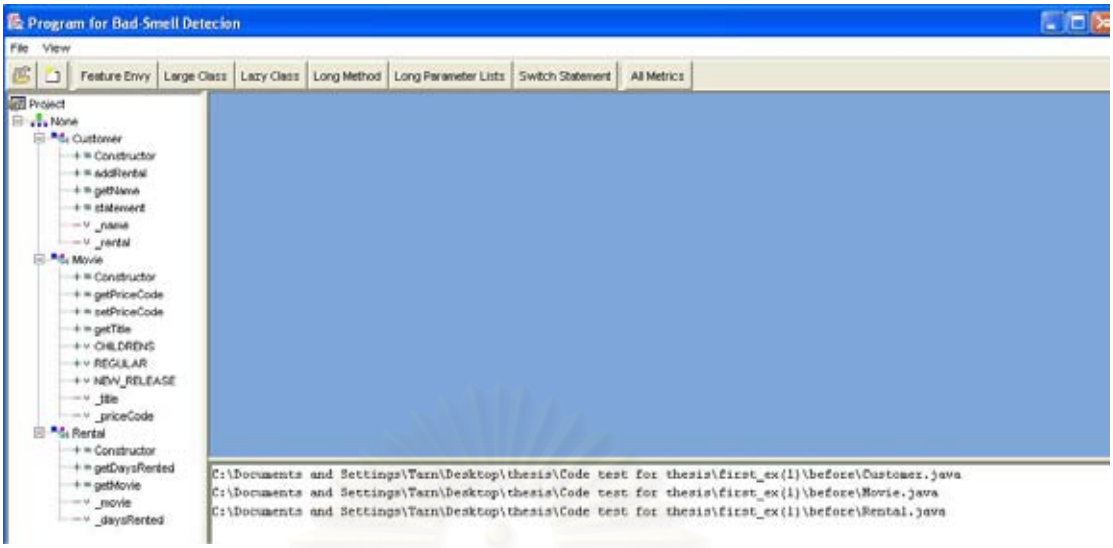
จ.1 การอ่านโปรแกรมต้นฉบับ

เมื่อผู้ใช้เข้าสู่โปรแกรม ผู้ใช้สามารถเลือกโปรแกรมต้นฉบับได้จากฟังก์ชันเพิ่มไฟล์ (File/ Add Files) เพื่อเลือกโปรแกรมต้นฉบับเข้าสู่โปรเจค ผู้ใช้สามารถเลือกทีละไฟล์ หรือหลายๆ ไฟล์ได้ ดังรูปที่ จ.1



รูปที่ จ.1 เฟรมเพื่อเลือกโปรแกรมต้นฉบับจาวา

หลังจากผู้ใช้ได้เลือกโปรแกรมต้นฉบับภาษาจาวาเข้าสู่ระบบแล้ว ระบบจะแสดงชื่อของโปรเจค แพ็กเกจ คลาส เมทธอด และตัวแปรอินสแตนท์ในรูปแบบต้นไม้ (Tree) ดังรูปที่ จ.2 เพื่อให้ผู้ใช้สามารถเลือกดูค่าตัววัดต่างๆ ตามที่ผู้ใช้ต้องการ หรือถ้าผู้ใช้ต้องการที่จะเริ่มเลือกโปรแกรมต้นฉบับเข้าสู่ต้นฉบับเข้าสู่โปรเจคใหม่ตั้งแต่ต้น ผู้ใช้สามารถใช้ฟังก์ชันเริ่มโปรเจคใหม่ (File/ New Project) หรือถ้าผู้ใช้ต้องการที่จะออกจากระบบ ผู้ใช้สามารถใช้ฟังก์ชันออกจากระบบ



รูปที่ จ.2 ทรีและโหนดต่างๆ ที่ได้จากการเลือกโปรแกรมต้นฉบับ

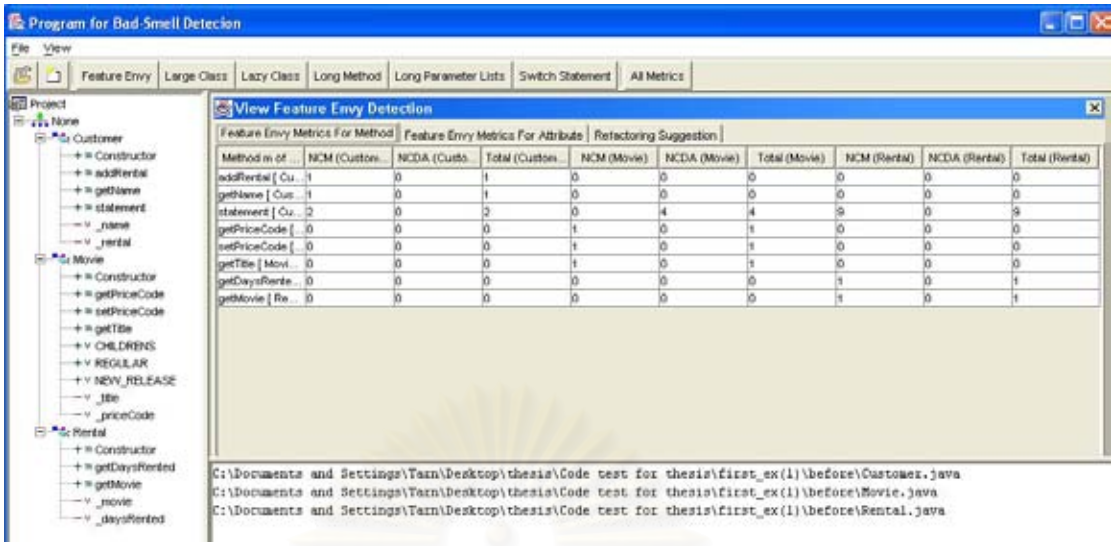
จ.2 การดูค่ามาตรวัด และวิธีการรีแฟคทอริง

เมื่อผู้ใช้ต้องการดูค่ามาตรวัด ผู้ใช้สามารถเลือกฟังก์ชันดูมาตรวัดแบ่งตามประเภทของร่องรอยที่ไม่ดีแต่ละประเภท 6 ประเภท คือ Feature Envy, Large Class, Lazy Class, Long Method, Long Parameter Lists และ Switch Statement และเลือกดูมาตรวัดทั้งหมดแยกตามคลาส เมทธอดและคุณลักษณะ ซึ่งมีรายละเอียดดังนี้

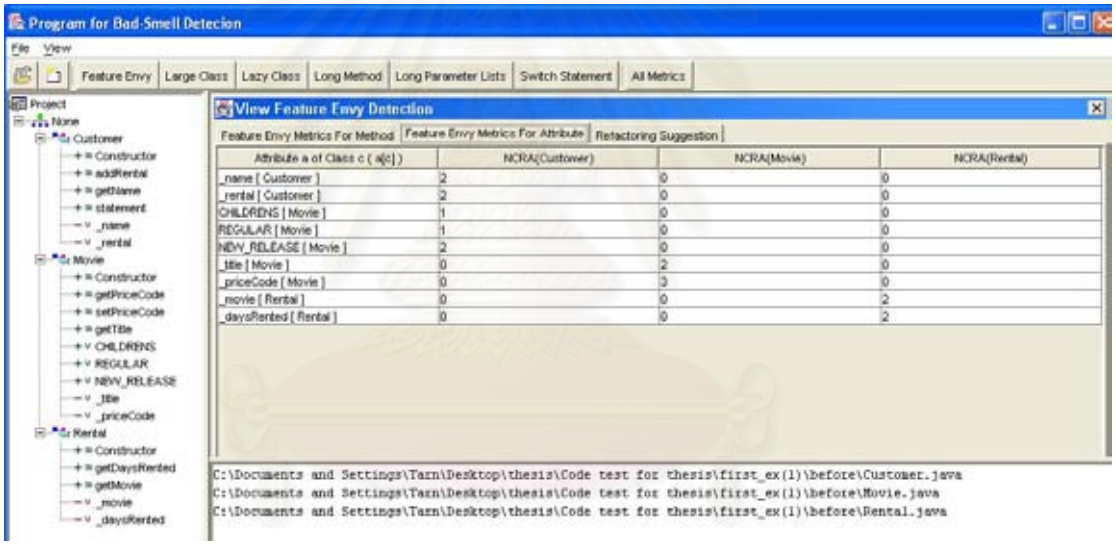
จ.2.1 การดูค่ามาตรวัด และวิธีการรีแฟคทอริงของ Feauture Envy

ผู้ใช้สามารถเลือกฟังก์ชันดูมาตรวัดร่องรอยที่ไม่ดีของ Feature Envy จาก 

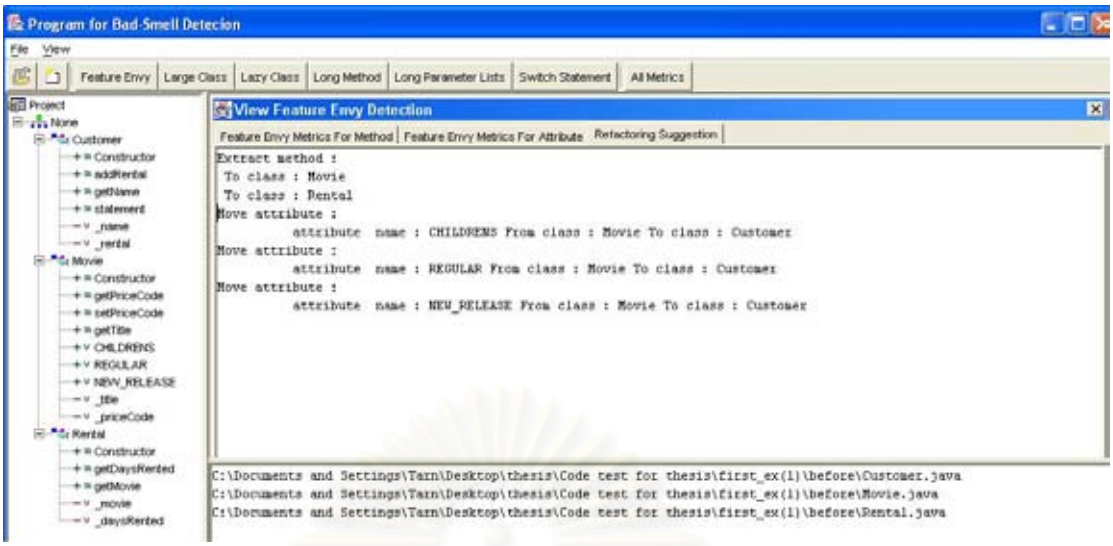
(View/ Feature Envy) ระบบก็จะทำการแสดงแฟรมที่มีแท็ปแสดงรายละเอียดค่ามาตรวัดของเมทธอด คุณลักษณะ และวิธีการรีแฟคทอริง ดังแสดงรายละเอียดในรูปที่ จ.3, จ.4 และ จ.5



รูปที่ จ.3 ค่ามาตรวัดร่องรอยที่ไม่ดีของ Feature Envy สำหรับเมทธอด



รูปที่ จ.4 ค่ามาตรวัดร่องรอยที่ไม่ดีของ Feature Envy สำหรับคุณลักษณะ

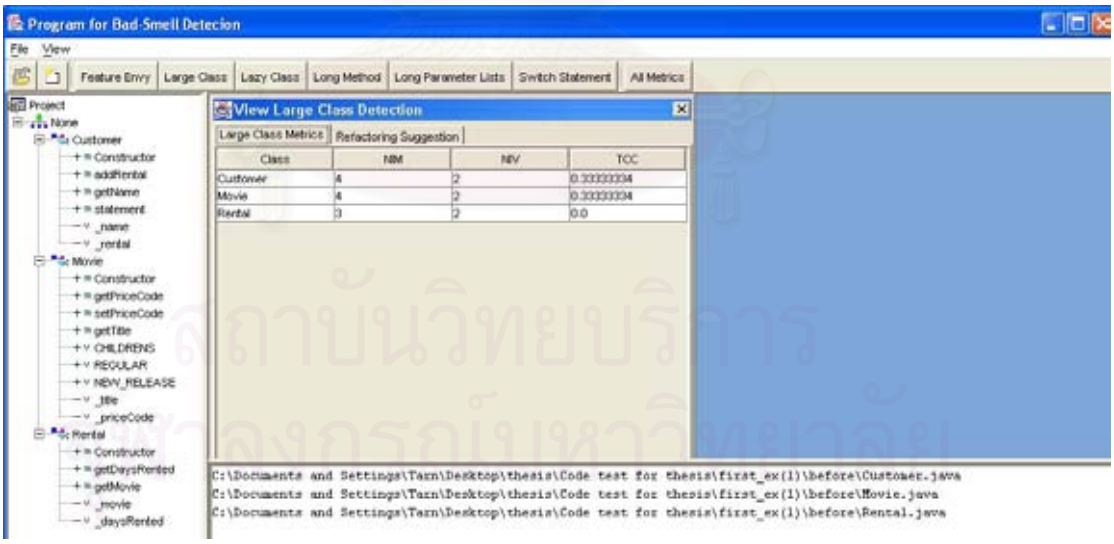


รูปที่ จ.5 ค่าวิธีรีแฟคทอริงของ Feature Envy

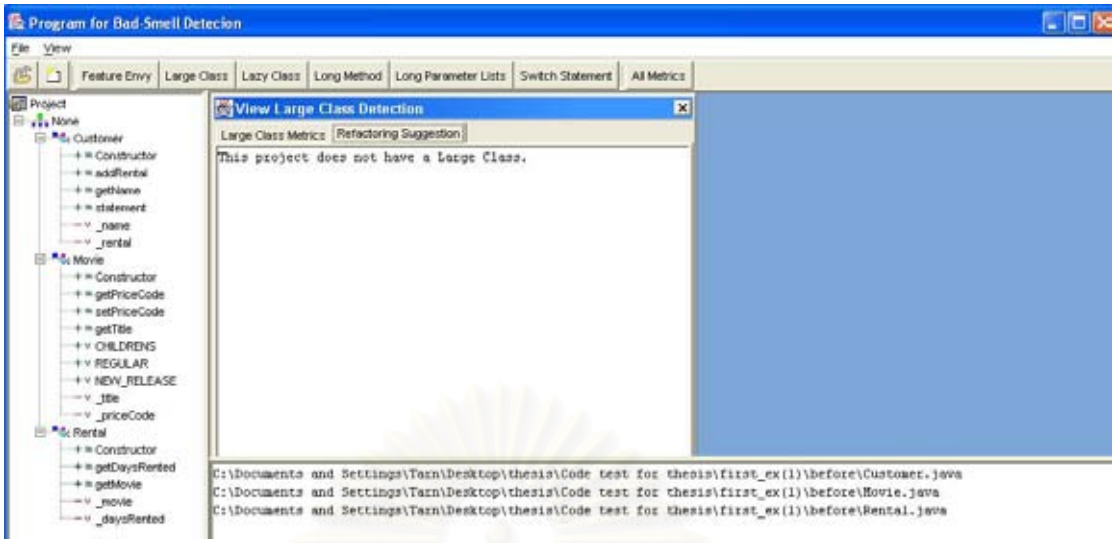
จ.2.2 การดูค่ามาตรวัด และวิธีการรีแฟคทอริงของ Large Class

ผู้ใช้สามารถเลือกฟังก์ชันดูมาตรวัดร่องรอยที่ไม่ดีของ Large Class จาก 

(View/ Large Class) ระบบก็จะทำการแสดงแฟรมที่มีแท็ปแสดงรายละเอียดค่ามาตรวัด และวิธีการรีแฟคทอริง ดังแสดงรายละเอียดในรูปที่ จ.6 และ จ.7



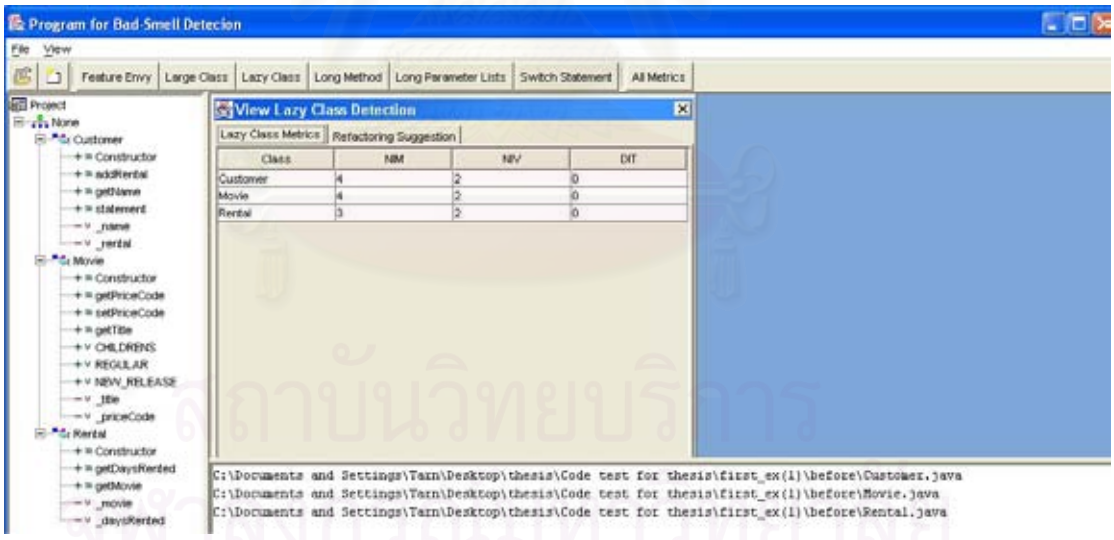
รูปที่ จ.6 ค่ามาตรวัดร่องรอยที่ไม่ดีของ Large Class



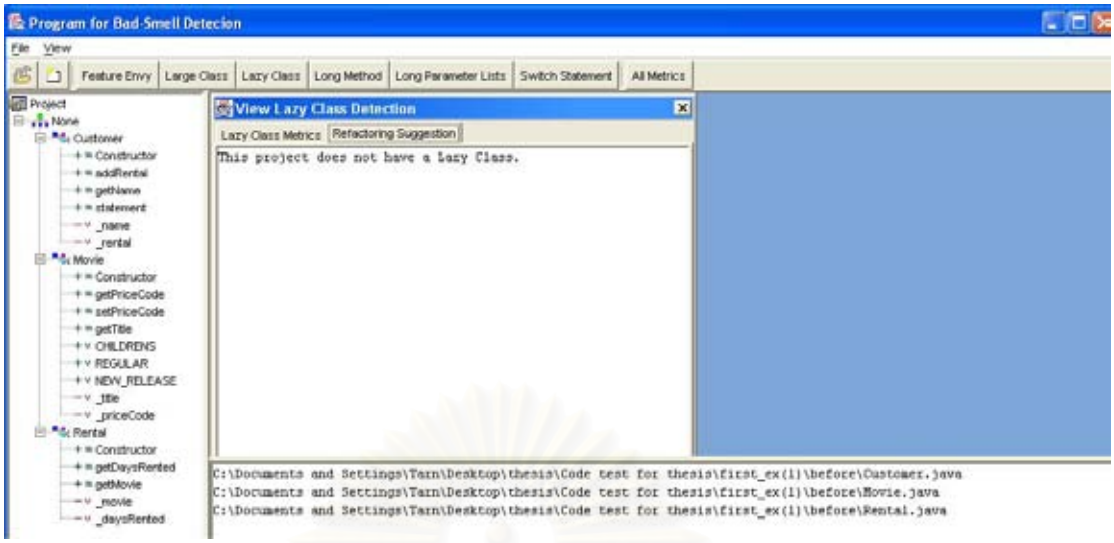
รูปที่ จ.7 วิธีรีแฟคทอริงของ Large Class

จ.2.3 การดูค่ามาตรวัด และวิธีการรีแฟคทอริงของ Lazy Class

ผู้ใช้สามารถเลือกฟังก์ชันดูมาตรวัดร่องรอยที่ไม่ดีของ Lazy Class จาก Lazy Class (View/ Lazy Class) ระบบก็จะทำการแสดงแฟรมที่มีแท็ปแสดงรายละเอียดค่ามาตรวัด และวิธีการรีแฟคทอริง ดังแสดงรายละเอียดในรูปที่ จ.8 และ จ.9



รูปที่ จ.8 ค่ามาตรวัดร่องรอยที่ไม่ดีของ Lazy Class

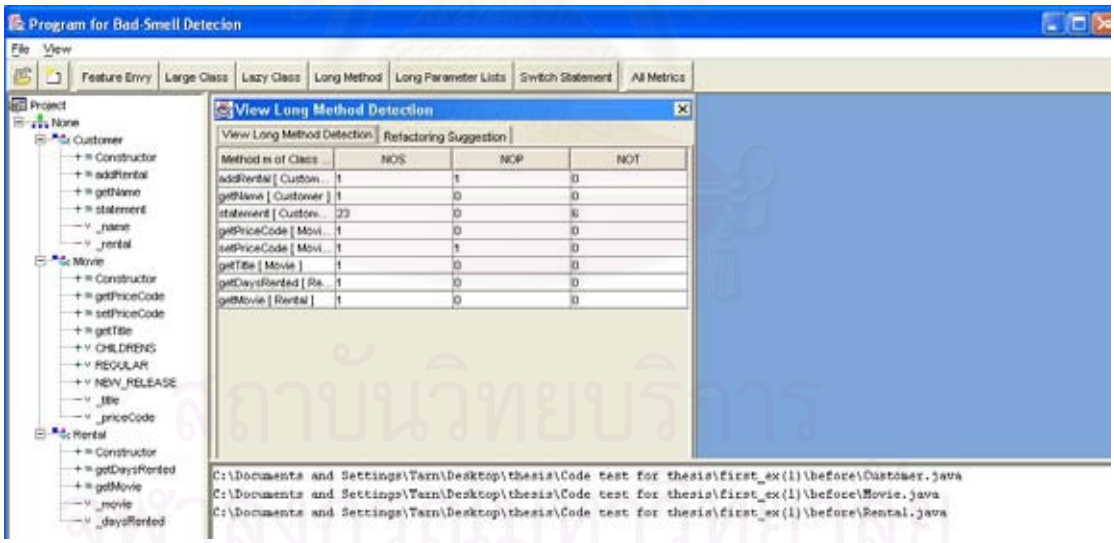


รูปที่ จ.9 วิธีรีแฟคทอริงของ Lazy Class

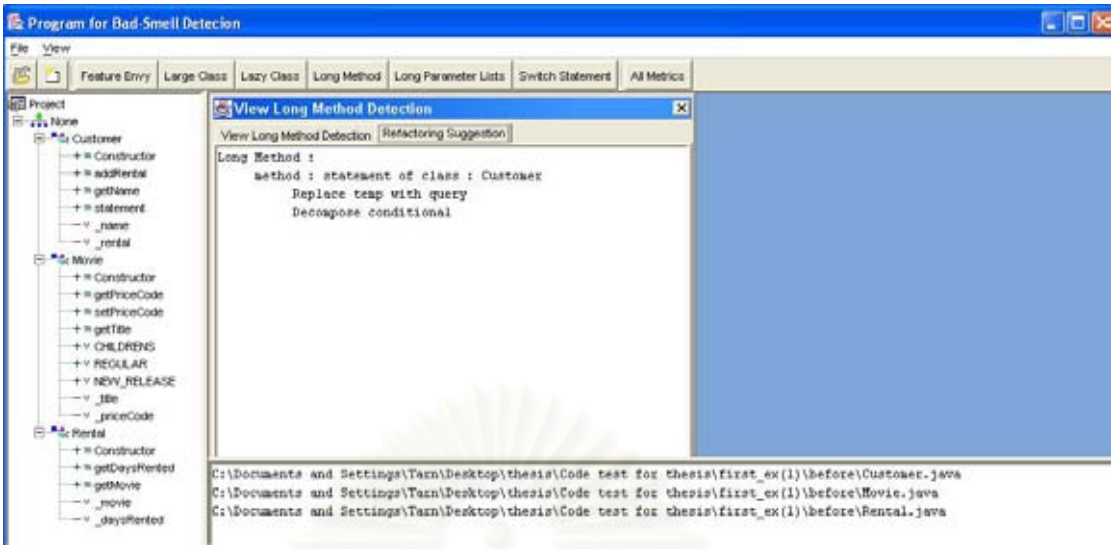
จ.2.4 การดูค่ามาตรวัด และวิธีการรีแฟคทอริงของ Long Method

ผู้ใช้สามารถเลือกฟังก์ชันดูมาตรวัดร่องรอยที่ไม่ดีของ Long Method จาก 

(View/ Long Method) ระบบก็จะทำการแสดงแฟรมที่มีแท็ปแสดงรายละเอียดค่ามาตรวัด และวิธีการรีแฟคทอริง ดังแสดงรายละเอียดในรูปที่ จ.10 และ จ.11



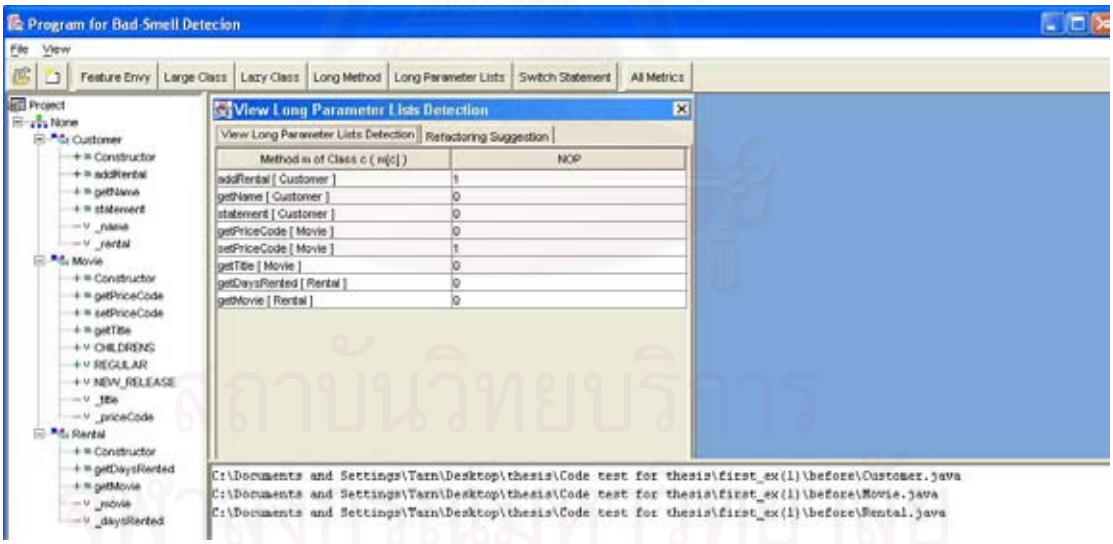
รูปที่ จ.10 ค่ามาตรวัดร่องรอยที่ไม่ดีของ Long Method



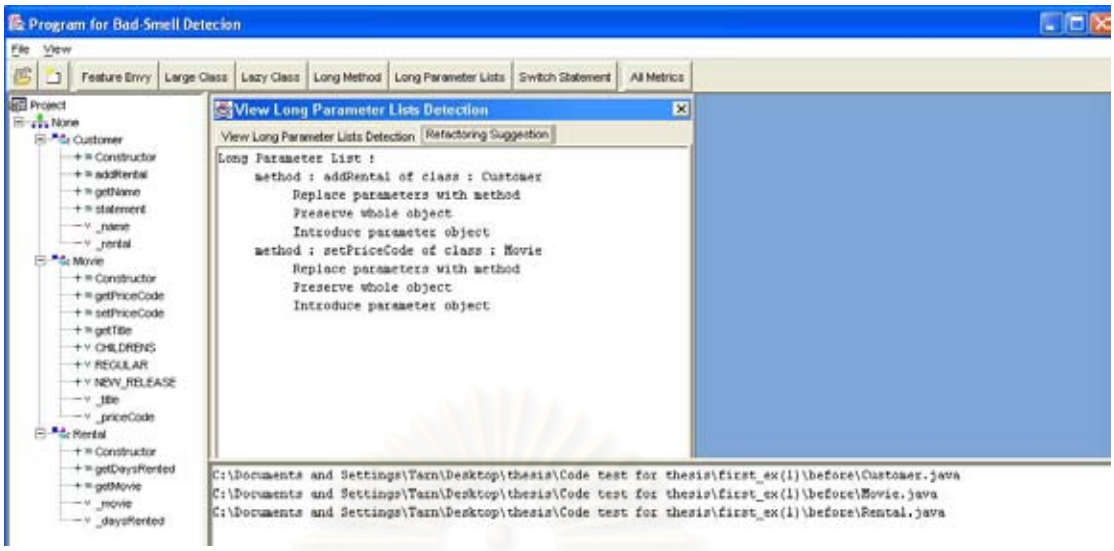
รูปที่ จ.11 วิธีรีแฟคทอริงของ Long Method

จ.2.5 การดูค่ามาตรวัด และวิธีการรีแฟคทอริงของ Long Parameter Lists

ผู้ใช้สามารถเลือกฟังก์ชันดูมาตรวัดร่องรอยที่ไม่ดีของ Long Parameter Lists จาก Long Parameter Lists (View/ Long Parameter Lists) ระบบก็จะทำการแสดงแฟรมที่มีแท็ปแสดงรายละเอียดค่ามาตรวัด และวิธีการรีแฟคทอริง ดังแสดงรายละเอียดในรูปที่ จ.12 และ จ.13



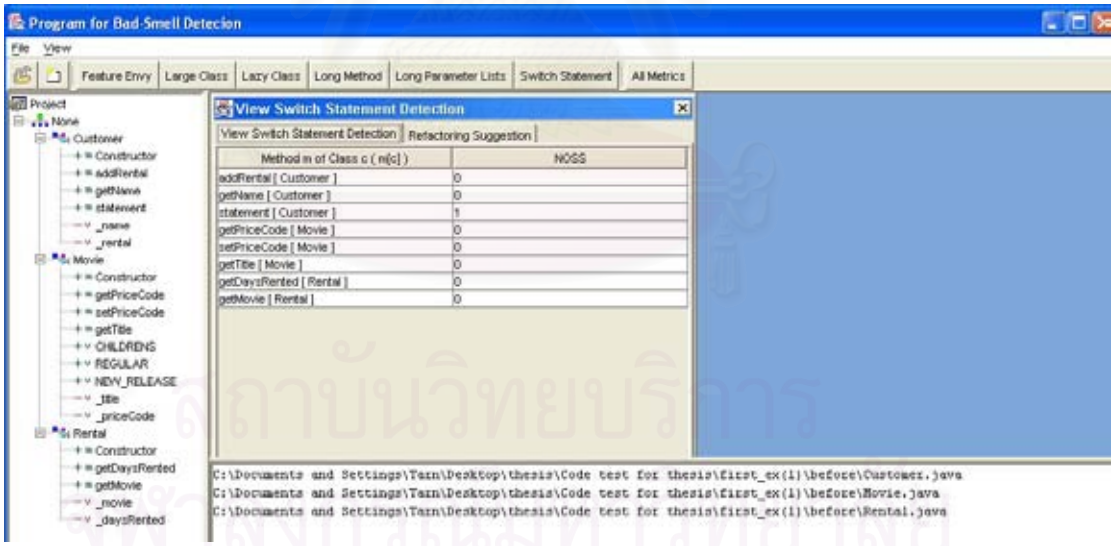
รูปที่ จ.12 ค่ามาตรวัดร่องรอยที่ไม่ดีของ Long Parameter Lists



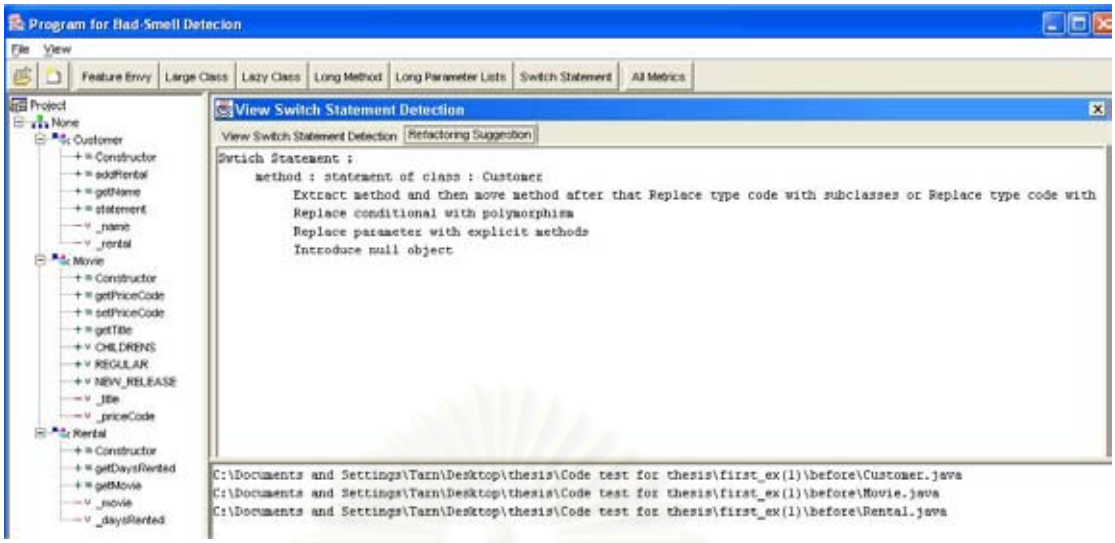
รูปที่ จ.13 วิธีรีแฟคทอริงของ Long Parameter Lists

จ.2.6 การดูค่ามาตรวัด และวิธีการรีแฟคทอริงของ Switch Statement

ผู้ใช้สามารถเลือกฟังก์ชันดูมาตรวัดร่องรอยที่ไม่ดีของ Switch Statement จาก Switch Statement (View/ Switch Statement) ระบบก็จะทำการแสดงแฟรมที่มีแท็ปแสดงรายละเอียดค่ามาตรวัด และวิธีการรีแฟคทอริง ดังแสดงรายละเอียดในรูปที่ จ.14 และ จ.15



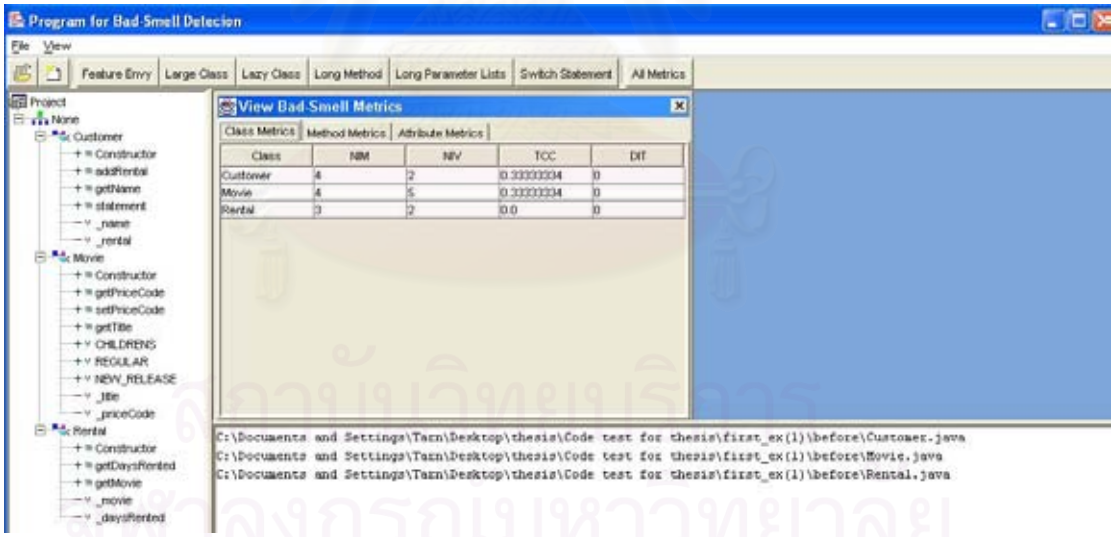
รูปที่ จ.14 ค่ามาตรวัดร่องรอยที่ไม่ดีของ Switch Statement



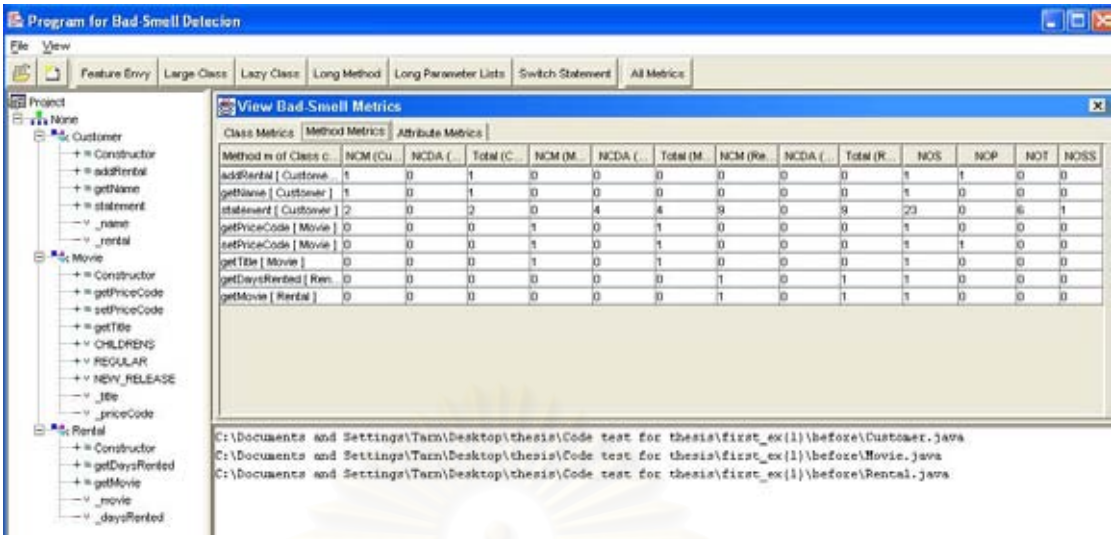
รูปที่ จ.15 วิธีรีแฟคทอริงของ Switch Statement

จ.2.7 การดูค่ามาตรวัดทั้งหมด แยกตามคลาส เมทธอด และคุณลักษณะ

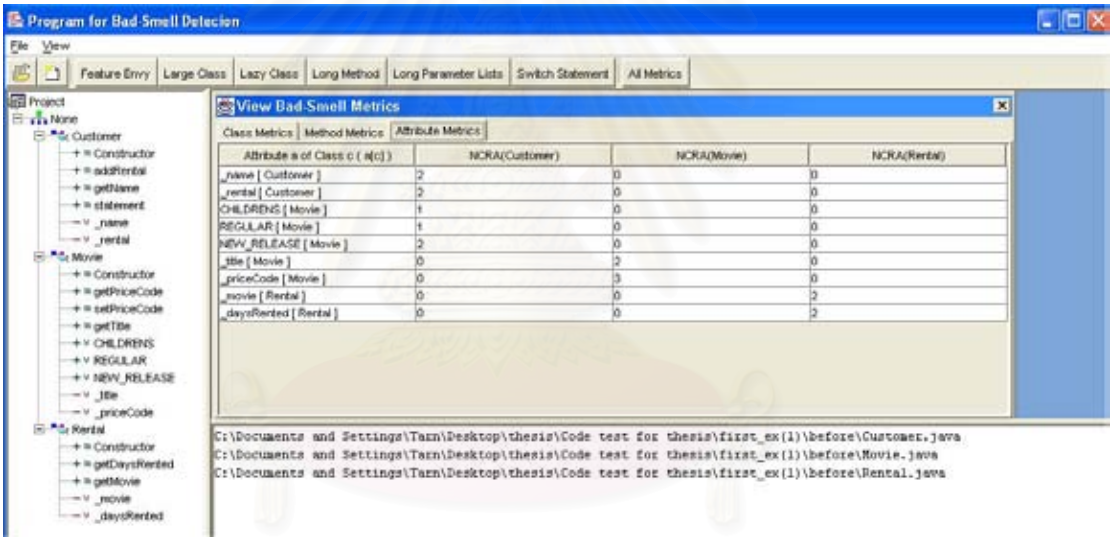
ผู้ใช้สามารถเลือกฟังก์ชันดูมาตรวัดร่องรอยทั้งหมด จาก All Metrics (View/ All Metrics) ระบบก็จะทำการแสดงแฟรมที่มีแท็ปแสดงรายละเอียดค่ามาตรวัด และวิธีการรีแฟคทอริง ดังแสดงรายละเอียดในรูปที่ จ.16, จ.17 และ จ.18



รูปที่ จ.16 ค่ามาตรวัดร่องรอยที่ไม่ดีของคลาส



รูปที่ จ.17 ค่ามาตรวัดร่องรอยที่ไม่ดีของเมทธอด



รูปที่ จ.18 ค่ามาตรวัดร่องรอยที่ไม่ดีของคุณลักษณะ

ภาคผนวก ฉ
ผลงานตีพิมพ์

ผลงานวิจัยนี้ ได้รับคัดเลือกให้ถูกตีพิมพ์ในงานสัมมนาวิชาการต่างประเทศ “Computer Science, Software Engineering, Information Technology, e-business and Application 2004 (CSITeA-04)” ซึ่งได้จัดขึ้นที่เมืองไคโร ประเทศอียิปต์ ระหว่างวันที่ 27- 29 ธันวาคม 2547

ประวัติผู้เขียนวิทยานิพนธ์

นางสาวธิษณา เพียรเลิศ เกิดวันที่ 13 มิถุนายน พ.ศ. 2523 ที่จังหวัดกรุงเทพฯ สำเร็จการศึกษาปริญญาวิศวกรรมศาสตร์บัณฑิต สาขาวิศวกรรมศาสตร์คอมพิวเตอร์ คณะวิศวกรรมศาสตร์ มหาวิทยาลัยมหิดล ในปีการศึกษา 2543 และได้เข้าศึกษาในหลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิทยาศาสตร์คอมพิวเตอร์ ณ จุฬาลงกรณ์มหาวิทยาลัย ปีการศึกษา 2545