วิธีการปรับปรุงคุณภาพโค้ดด้วยมาตรวัดซอฟต์แวร์และฟัซซี่โลจิก เพื่อเพิ่มความสามารถในการบำรุงรักษา

นายพรชัย เลิศหทัยรัตน์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2554

A METHOD FOR CODE QUALITY IMPROVEMENT USING SOFTWARE METRICS AND FUZZY LOGIC TO ENHANCE MAINTAINABILITY

Mr. Pornchai Lerthathairat

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science Program in Computer Science

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2011

หัวข้อวิทยานิพนธ์ วิธีการปรับปรุงคุณภาพโค้ดด้วยมาตรวัดซอฟต์แวร์และ
ฟัซซี่โลจิกเพื่อเพิ่มความสามารถในการบำรุงรักษา

โดย	พรชัย เลิศหทัยรัตน์
สาขาวิชา	วิทยาศาสตร์คอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	ผู้ช่วยศาสตราจารย์ นครทิพย์ พร้อมพูล

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้ เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(รองศาสตราจารย์ ดร.บุญสม เลิศหิรัญวงศ์)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(รองศาสตราจารย์ ดร. วิวัฒน์ วัฒนาวุฒิ)

..... อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(ผู้ช่วยศาสตราจารย์ นครทิพย์ พร้อมพูล)

..... กรรมการ
(รองศาสตราจารย์ ดร.ธาราทิพย์ สุวรรณศาสตร์)

..... กรรมการภายนอกมหาวิทยาลัย
(ดร.เฉลิมศักดิ์ เลิศวงศ์เสถียร)

พรชัย เลิศหทัยรัตน์ : วิธีการปรับปรุงคุณภาพโค้ดด้วยมาตรวัดซอฟต์แวร์และฟัซซี่โลจิกเพื่อเพิ่มความสามารถในการบำรุงรักษา า. (A METHOD FOR CODE QUALITY IMPROVEMENT USING SOFTWARE METRICS AND FUZZY LOGIC TO ENHANCE MAINTAINABILITY) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: ผศ. นครทิพย์ พร้อมพูล, 265 หน้า

จุดมุ่งหมายที่สำคัญของกระบวนการพัฒนาซอฟต์แวร์คือ การส่งมอบผลิตภัณฑ์ซอฟต์แวร์ที่มีคุณภาพและตอบสนองความต้องการของผู้ใช้งาน ด้วยข้อจำกัดด้านเวลาและงบประมาณทำให้ผู้พัฒนามักจะคำนึงถึงปัจจัยด้านคุณภาพน้อยกว่าปัจจัยอื่น จึงอาจเป็นสาเหตุให้เกิดร่องรอยไม่ดีปรากฏขึ้นในซอฟต์แวร์ที่มีผลต่อการทำงานของซอฟต์แวร์เป็นผลให้ไม่สามารถทำงานได้อย่างเหมาะสม ผู้พัฒนาจำเป็นต้องจัดสรรเวลาเพื่อแก้ไขให้เรียบร้อยก่อนส่งมอบ ให้ผู้ใช้งาน งานวิจัยนี้เล็งเห็นถึงความสำคัญของความสามารถในการบำรุงรักษาที่จะช่วยในการระบุตำแหน่งร่องรอยไม่ดีได้อย่างง่ายและรวดเร็วช่วยให้ใช้ระยะเวลาในการแก้ไขน้อยลง

งานวิจัยนี้นำเสนอวิธีการปรับปรุงคุณภาพโค้ดด้วยมาตรวัดซอฟต์แวร์และฟัซซี่โลจิก เพื่อเพิ่มความสามารถในการบำรุงรักษา ประกอบด้วย 3 ส่วนหลักได้แก่ ส่วนที่ 1 การจำแนกโค้ดด้วยมาตรวัดซอฟต์แวร์และฟัซซี่โลจิกออกเป็น 3 กลุ่มได้แก่ กลุ่มซับเซตคลีนโค้ด กลุ่มโค้ดที่มีความคลุมเครือ และกลุ่มร่องรอยไม่ดี โดยผลลัพธ์ที่ถูกจำแนกเป็นร่องรอยไม่ดีและความคลุมเครือจะถูกแก้ไขในส่วนที่ 2 การปรับปรุงร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือด้วยเทคนิค รีแฟคทอริ่งตามวิธีปฏิบัติที่ออกแบบไว้เพื่อให้ได้เป็นโค้ดประเภทซับเซตคลีนโค้ด และส่วนที่ 3 การวัดคุณภาพโค้ดด้วยมาตรวัดดัชนีความสามารถในการบำรุงรักษา ผลการจำแนกโค้ดมีความถูกต้องคิดเป็นร้อยละ 85 จาก 60 ตัวอย่าง และสามารถเพิ่มความสามารถในการบำรุงรักษาตามวิธีการที่ออกแบบคิดเป็นร้อยละ 62.5 จาก 60 ตัวอย่าง วิธีการปรับปรุงคุณภาพโค้ดที่สร้างขึ้นนี้ช่วยสนับสนุนการผลิตซอฟต์แวร์ให้มีคุณภาพด้วยการเพิ่ม ความสามารถในการบำรุงรักษา ให้แก่โค้ดที่ได้รับการปรับปรุง ซึ่งมีส่วนสำคัญในการผลิตซอฟต์แวร์ให้มีคุณภาพ

ภาควิชา..วิศวกรรมคอมพิวเตอร์..... ลายมือชื่อนิสิต……..…….

สาขาวิชา.วิทยาศาสตร์คอมพิวเตอร์ ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก ……..……

ปีการศึกษา……2554……

5271436221 : MAJOR SOFTWARE ENGINEERING
KEYWORDS: AMBIGUITY / CODE QUALITY IMPROVEMENT / CODE CLASSIFICATION / CLEANCODE / FUZZY SET / REFACTORING / SOFTWARE METRICS

PORNCHAI LERTHATHAIRAT: A METHOD FOR CODE QUALITY IMPROVEMENT USING SOFTWARE METRICS AND FUZZY LOGIC TO ENHANCE MAINTAINABILITY. ADVISOR: ASST.PROF. NAKORNTHIP PROMPOON, 265 pp.

The main purpose of software development is to deliver the quality software that meets the user requirements. Due to the limitation of time and budget, developers may pay less concern about quality factors than others. For this reason, bad smells code may appear in software which leads to software execution improperly. Developers have to allocate time to eliminate bad smell before delivering software to users. This research focuses on finding a method for software maintainability enhancement which helps identifying bad smell easily and reduces time of code improvement.

This research presents a method for code quality improvement using software metrics and fuzzy logic to enhance maintainability. Our approach is composed of 3 main sections. The first section is source code classification using software metrics and fuzzy logic. The result from this section can be able to classify code into Sub-set of Clean code, Ambiguous code and Bad smell. The second section is code improvement using refactoring. The third section is source code measurement using maintainability index. From our 60 examples experiment, the result of our approach can classify source code 85% accurately and can enhance maintainability to 62.5 %. The process of code improvement using our proposed method helps support the software quality development focusing on code maintainability.

Department: Computer Engineering Student's Signature: .....
Field of Study: Computer Science Advisor's Signature: .....
Academic Year: 2011

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้ได้สำเร็จลุล่วงด้วยความเมตตาอย่างยิ่งจากผู้ช่วยศาสตราจารย์ นครทิพย์ พร้อมพูล อาจารย์ที่ปรึกษาที่เสียสละเวลาช่วยให้คำปรึกษา ข้อคิดและคำแนะนำที่มีประโยชน์ต่องานวิจัย พร้อมมอบ ความเชื่อมั่นที่อาจารย์มีให้ ต่อผู้วิจัย ซึ่งเป็นกำลังใจและเป็นแรงส่งเสริมอย่างดีเยี่ยมให้กับผู้วิจัยสามารถพัฒนางานวิจัยที่มีคุณภาพและมีคุณค่า

ขอกราบขอบพระคุณคณะกรรมการสอบวิทยานิพนธ์ รองศาสตราจารย์ ดร. วิวัฒน์ วัฒนวุฒิ ประธานกรรมการสอบวิทยานิพนธ์ รองศาสตราจารย์ ดร.ธาราทิพย์ สุวรรณศาสตร์ และ ดร.เฉลิมศักดิ์ เลิศวงศ์เสถียร ที่กรุณาสละเวลาในการให้คำแนะนำเกี่ยวกับงานวิจัยและตรวจสอบความถูกต้องของวิทยานิพนธ์ฉบับนี้ให้มีคุณภาพและความสมบูรณ์ที่สุด

ขอขอบพระคุณคณาจารย์ในภาควิชาวิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัย ทุกท่านที่ประสิทธิ์ประสาทความรู้อันมีค่ายิ่งแก่ผู้วิจัย

ขอขอบคุณบุคลากรในภาควิชาวิศวกรรมคอมพิวเตอร์ จุฬาลงกรณ์มหาวิทยาลัยทุกท่าน ที่ให้ข้อมูล คำแนะนำและความช่วยเหลือในการดำเนินการทั้งในเรื่องการศึกษาและการสอบวิทยานิพนธ์ได้สำเร็จลุล่วง

สุดท้ายกราบขอบพระคุณบิดา มารดาและเพื่อนๆ ที่คอยสนับสนุนพลังในการสร้างสรรค์ผลงานและคอยให้กำลังใจมาโดยตลอดเพื่อให้ได้มาซึ่งวิทยานิพนธ์ฉบับนี้

สารบัญ

บทที่ หน้า

หน้า

สารบัญตาราง

หน้า

หน้า

หน้า

หน้า

หน้า

หน้า

สารบัญภาพ

หน้า

บทที่ 1

บทนำ

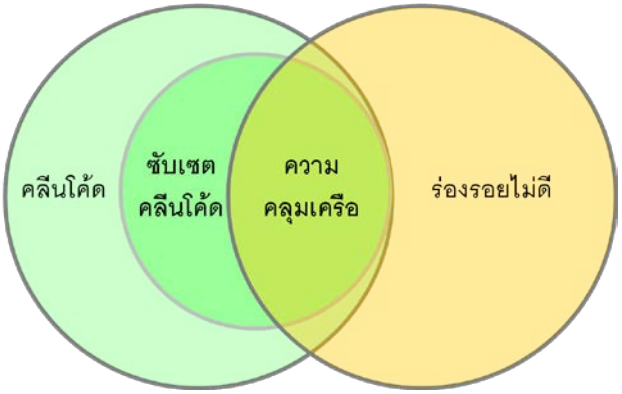
1.1 ความเป็นมาและความสำคัญของปัญหา

จุดมุ่งหมายที่สำคัญของกระบวนการพัฒนาซอฟต์แวร์ที่ดีคือ การส่งมอบผลิตภัณฑ์ซอฟต์แวร์ที่สามารถตอบสนองต่อความต้องการของผู้ใช้งาน ดังเช่นนิยามของ Phillip Crosby [1] นักวิจัยด้านคุณภาพ ที่ได้นิยามคุณภาพซอฟต์แวร์ไว้ว่า " Conformance to requirement" ซึ่งมีความหมาย คือ “ตรงตามความต้องการของผู้ใช้งาน ” นอกจากนี้มีการนิยาม คุณภาพของผลิตภัณฑ์ไว้ถึงความ สามารถประเมินและวัดคุณภาพจากจุดใดนั้น Peter Drucker [2] นักวิจัยด้านคุณภาพอีกท่านได้กล่าวว่า "ซอฟต์แวร์ที่มีคุณภาพ หมายถึง การสร้างความพึงพอใจระดับสูงให้กับผู้ใช้งาน ไม่ว่าโปรแกรมนั้นจะถูกพัฒนาด้วยเครื่องมือใดก็ตาม ฉะนั้นจึงเห็นได้ว่า ซอฟต์แวร์ที่มีคุณภาพ เกิดจาก ผู้พัฒนา มีความมุ่งมั่นที่จะผลิต ซอฟต์แวร์ คุณภาพเพื่อตอบสนองความต้องการของผู้ใช้งานเป็นสำคัญ

การผลิตซอฟต์แวร์ที่ดีและมีคุณภาพนั้นจำเป็นต้องมี กรอบแนวทางในการประเมินคุณภาพของซอฟต์แวร์ที่ผลิตด้วยการ สนับสนุนให้เกิ ดกระบวนการพัฒนาซอฟต์แวร์ที่นำไปสู่ผลผลิตที่มีคุณภาพ [3] ดังมาตรฐาน The International Organization for Standardization and The International Electrotechnical Commission: Software Quality (ISO/IEC 9126) [4] เป็นกรอบแนวทางมาตรฐานสนับสนุนวิธีการประเมินคุณภาพภายในและภายนอกของซอฟต์แวร์ที่มีคุณสมบัติ ได้แก่ การตอบสนองต่อความต้องการในการใช้งาน (Functionality) ความน่าเชื่อถือ (Reliability) การใช้งาน (Usability) ความมีประสิทธิภาพ (Efficiency) ความสามารถในการบำรุงรักษา (Maintainability) และ ความสามารถในการปรับใช้ (Portability) วิทยานิพนธ์นี้ มุ่งประเด็นความสามารถในการบำรุงรักษา ซอฟต์แวร์เป็นสำคัญ เนื่องจาก สามารถประยุกต์ใช้งานร่วมกับการพัฒนาซอฟต์แวร์ตั้งแต่เริ่มต้นกระบวนการ โดยเริ่มจากขั้นตอนการเก็บรวบรวมความต้องการของผู้ใช้งาน ขั้นตอนออกแบบและพัฒนาโปรแกรม ขั้นตอนทำการทดสอบ ขั้นตอนส่งมอบให้ผู้ใช้งาน และขั้นตอนการบำรุงรักษาหลังจากส่งมอบซอฟต์แวร์ ซึ่งคุณสมบัตินี้มีความสำคัญสำหรับการผลิตซอฟต์แวร์ที่มีคุณภาพ ดังที่ผู้พัฒนาทำเพื่อตอบสนองความต้องการของ ผู้ใช้งาน หากพิจารณาในขั้นตอนการออกแบบและพัฒนาโปรแกรม ภายในการทำงานของ โปรแกรมหนึ่งๆ จะประกอบไปด้วยโค้ดและโมดูลย่อยที่ทำการเชื่อมต่อกันและอาจ มีการทำงาน ที่ซับซ้อน แต่ควร

สามารถรองรับการเปลี่ยนแปลงได้ง่ายตามความต้องการ ฉะนั้นการบำรุงรักษาจึงมีส่วนสำคัญ หากมีการเปลี่ยนแปลงหรือตรวจพบข้อผิดพลาดที่เกิดขึ้นกับซอฟต์แวร์ใดๆ คุณสมบัตินี้มีส่วนช่วยวิเคราะห์และระบุตำแหน่งข้อผิดพลาดได้อย่างง่ายและรวดเร็วส่งผลให้ระยะเวลาในการซ่อมบำรุงน้อยลง

แต่ในทางปฏิบัติกลับพบว่า มีปัญหาหนึ่งที่เกิดขึ้นซ้ำๆ ในกระบวนการพัฒนาซอฟต์แวร์[5] คือ ผู้พัฒนาคำนึงถึงปัจจัยด้านคุณภาพน้อยกว่าปัจจัยด้านเวลาเพื่อส่งมอบงานให้ทันกำหนดและด้วยปัจจัยด้านงบประมาณที่มีอย่างจำกัด ส่งผลให้การวิเคราะห์ความต้องการของผู้ใช้ไม่รอบคอบและด้วยเหตุนี้จึงส่งผลต่อเนื่องไปยังกระบวนการออกแบบและพัฒนาโปรแกรม อาจเป็นสาเหตุให้ร่องรอยไม่ดี (Bad Smell) ปรากฏขึ้นในซอฟต์แวร์ หากลองวิเคราะห์ สาเหตุ ที่เกิดร่องรอยไม่ดี อาทิเช่น ความซับซ้อนภายในโปรแกรม การออกแบบที่ไม่ ครอบคลุมกับความต้องการ อาจเป็นสาเหตุให้เกิด การทำงานอย่างไม่พึงประสงค์ และทำให้เกิด การเปลี่ยนแปลงแก้ไขยาก ซึ่งผู้วิจัยได้นิยามร่องรอยไม่ดี ไว้ในบทที่ 2 ของงานวิทยานิพนธ์นี้ เพราะฉะนั้นการแก้ไขร่องรอยไม่ดี ถือเป็นอุปสรรคหนึ่งที่ผู้พัฒนาจำเป็นต้องจัดสรรเวลาเพื่อจัดการแก้ไขให้เรียบร้อยก่อนส่งมอบให้ผู้ใช้งาน มิเช่นนั้นอุปสรรคนี้อาจเป็นเหตุทำให้โครงการ ล่าช้าและเสียค่าใช้จ่ายในการบำรุงรักษาเพิ่มขึ้น สำหรับการแก้ไขร่องรอยไม่ดีนั้นในปัจจุบันมีเทคนิครีแฟคทอริ่ง (Refactoring Technique) ที่ Martin Fowler และคณะ เป็นผู้คิดค้นขึ้น [6] แม้ว่าเทคนิครีแฟคทอริ่งจะเข้ามามีส่วนช่วยให้การแก้ไขสะดวกขึ้น แต่เทคนิครีแฟคทอริ่งมีข้อจำกัดหลายประการที่ไม่สามารถทำให้ซอฟต์แวร์มีคุณภาพได้ [7] เนื่องจากแนวทางการทำรีแฟคทอริ่งนั้นเป็นการแก้ไขโค้ดที่ไม่ได้พิจารณาปัจจัยด้านคุณภาพเป็นสำคัญ ส่งผลให้เกิดโค้ดที่มีคุณภาพแย่ลง (Code Degradation) [5] นอกจากนี้ผลลัพธ์ที่ได้จากการแก้ไขร่องรอยไม่ดีอาจเกิดความคลุมเครือ (Ambiguity) ซึ่งเป็นโค้ดที่ไม่ปรากฏร่องรอยไม่ดีแต่ขาดคุณสมบัติคลีนโค้ดเป็นเหตุให้ต้องออกแบบวิธีการแก้ไขปัญหาในส่วนนี้เพื่อปรับปรุงให้มีความคุณสมบัติเป็นคลีนโค้ด ทั้งนี้ความคลุมเครือที่พบเกิดขึ้นได้ดังภาพที่ 1.1 สำหรับวิทยานิพนธ์นี้ได้ออกแบบวิธีการจำแนกและปรับปรุงโค้ดไว้ โดยกำหนดขอบเขตเฉพาะส่วนย่อยของคลีนโค้ดเท่านั้น ซึ่งผู้วิจัยขอนิยามส่วนย่อยของคลีนโค้ดนี้ว่า ซับเซตคลีนโค้ด



ภาพที่ 1.1 ความคลุมเครือเกิดขึ้นระหว่างซับเซตคลีนโค้ดและร่องรอยไม่ดี

ดังนั้นวิทยานิพนธ์นี้ขอนำเสนอ วิธี การปรับปรุงโค้ดให้มีคุณภาพเพื่อแก้ไขปัญหาร่องรอยไม่ดีและความคลุมเครือ โดยมี 3 ขั้นตอนหลักได้แก่ ขั้นตอนการสร้างวิธีการจำแนก โดยใช้มาตรวัดซอฟต์แวร์และฟัซซี่โลจิก ขั้นตอนการนำวิธีการปฏิบัติไปแก้ไขร่องรอยไม่ดีและความคลุมเครือเพื่อปรับปรุงโค้ดให้มีซับเซตคลีนโค้ดตามเกณฑ์ และ ขั้นตอนการประเมินความสามารถในการบำรุงรักษาโดยใช้มาตรวัดซอฟต์แวร์ดัชนีความสามารถในการบำรุงรักษาทำการประเมินค่าดัชนีความสามารถในการบำรุงรักษาโค้ดก่อนและหลังการปรับปรุงเพื่อเป็นผลสรุปของวิธีการที่ออกแบบ สามารถแก้ไขและปรับปรุงคุณภาพโค้ดได้จริง

ขั้นตอนการสร้างวิธีการจำแนกโดยใช้มาตรวัดซอฟต์แวร์และฟัซซี่โลจิก ขั้นตอนที่ 1 เริ่มจากการนำโค้ดตัวอย่างจำนวน 60 ชุดตัวอย่างประกอบด้วย กลุ่มซับเซต คลีนโค้ด 20 ชุดตัวอย่าง กลุ่มร่องรอยไม่ดี 20 ชุดตัวอย่าง และกลุ่มโค้ด ที่มีความ คลุมเครือ 20 ชุดตัวอย่าง มาสร้างเกณฑ์การวัดโดยคัดเลือกมาตรวัดซอฟต์แวร์พื้นฐาน เพื่ออธิบายกลุ่มของคลีนโค้ด และความคลุมเครืออาทิ เช่น คัปปลิง(Coupling) โคฮีชัน(Cohesion) โฮลสเตท(Halstead) เป็นต้น และกลุ่มของร่องรอยไม่ดีได้นำงานวิจัยของ Mika Mäntylä [8] ที่เสนอการจัดกลุ่มร่องรอยไม่ดีพร้อม กับมาตรวัดซอฟต์แวร์มาใช้ในวิทยานิพนธ์นี้ เพื่อให้สามารถจำแนกซับเซตคลีนโค้ดและ ความคลุมเครือตามรูปที่ 1.1 จากนั้นนำเทคนิคฟัซซี่โลจิก (Fuzzy Logic) เข้ามาช่วยจำแนกและระบุความเป็นคลีนโค้ดให้ชัดเจนขึ้น ซึ่งเทคนิคนี้สามารถระบุตำแหน่งที่เกิดร่องรอยไม่ดีและความคลุมเครือที่เกิดขึ้น โดยจะนำไปประกอบการตัดสินใจสำหรับผู้พัฒนาในขั้นตอนการปรับปรุงตามวิธีการปฏิบัติที่ได้ออกแบบไว้ จากนั้นจึงทำการทวนสอบ ด้วยการรวบรวมโค้ดจาก 3 กลุ่มตัวอย่างคละกันจำนวน 60 ตัวอย่าง สำหรับการทวนสอบเพื่อประเมินความถูกต้องของวิธีการจำแนกโค้ด ขั้นตอนที่ 2 การนำวิธีการปฏิบัติไปแก้ไขร่องรอยไม่ดีและความคลุมเครือ โดยเลือกใช้เทคนิค

รีแฟคทอริ่งเพื่อให้โค้ดที่ได้รับการแก้ไขมีคุณภาพตามเกณฑ์ของคลีนโค้ดที่ Robert C. Martin [9] ได้รวบรวมและระบุ แนวคิด แบบรูปของคลีนโค้ด ไว้ ขั้นตอนที่ 3 คือ การประเมินด้วยมาตรวัดความสามารถในการบำรุงรักษา [10, 11] จะเป็น ตรวจสอบค่าดัชนีความสามารถในการบำรุงรักษาเพิ่มขึ้นหลังจากทำการปรับปรุง โค้ด ให้มีคุณสมบัติ เป็นซับเซตคลีนโค้ดซึ่งเป็นการยกระดับความสามารถในการบำรุงรักษาให้ดีขึ้นตามที่วิทยานิพนธ์นี้ได้ออกแบบไว้

วิทยานิพนธ์นี้ ต้องการสร้างวิธีการที่สนับสนุนการผลิตซอฟต์แวร์ให้มีคุณภาพ อีกทั้งยังสามารถนำวิธีการที่สร้างขึ้นไปประยุกต์ใช้ในการแก้ไขผลผลิตซอฟต์แวร์ให้กลับมีคุณภาพได้ ดังนั้นความสามารถในการบำรุงรักษา จึงมีส่วนสำคัญในการลดระยะเวลาในแก้ไขปรับปรุงผลผลิตซอฟต์แวร์ทำให้มีคุณภาพและตอบสนองความต้องการของผู้ใช้งานได้ทันเวลา

1.2 วัตถุประสงค์ของการวิจัย

1) ออกแบบและสร้างวิธีการจำแนกโค้ดเป็นซับเซตคลีนโค้ด ร่องรอยไม่ดีและความคลุมเครือโดยใช้มาตรวัดซอฟต์แวร์และฟัซซี่โลจิก

2) ออกแบบวิธีการปฏิบัติเพื่อใช้ปรับปรุงโค้ดกลุ่มร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือโดยใช้เทคนิครีแฟคทอริ่งในการแก้ไขปัญหาเพื่อให้โค้ดมีคุณสมบัติเป็นซับเซตคลีนโค้ดและเพิ่มความสามารถในการบำรุงรักษาให้แก่โค้ดได้

1.3 ขอบเขตการวิจัย

1) งานวิทยานิพนธ์นี้ได้จำแนกโค้ดออกเป็น 3 กลุ่ม ได้แก่ กลุ่มซับเซตคลีนโค้ด กลุ่มความคลุมเครือ และกลุ่มร่องรอยไม่ดี

2) กลุ่มซับเซตคลีนโค้ดที่ใช้ในงานวิทยานิพนธ์มีดังต่อไปนี้

(1) Good Comment

(2) Function ประกอบด้วย Small Functions, Do One Thing, Nesting Depth, Function Arguments, Structured Programming

(3) Class ประกอบด้วย Class Organization, Encapsulation, Classes Should Be Small, Maintaining Cohesion, Organizing for Change, The Law of Demeter, Clean Boundary

3) กลุ่มร่องรอยไม่ดีที่ใช้ในงานวิทยานิพนธ์มีดังต่อไปนี้

(1) Long Method

(2) Large Class

(3) Long Parameter List

(4) Temporary Field

(5) Feature Envy

(6) Inappropriate Intimacy

(7) Lazy Class

(8) Bad Comment

4) กลุ่มโค้ดตัวอย่างที่ใช้ในงานวิ ทยานิพนธ์นี้ เป็นชุดโค้ดภาษา C# ซึ่งกลุ่มโค้ดตัวอย่าง 1 ชุด จะประกอบไปด้วยระบบย่อย (Sub-system) ที่มีคลาสเดี่ยวหรือหลายคลาสที่สัมพันธ์กัน และเป็นโค้ดที่แปลโปรแกรม (Compile) ผ่านแล้วทั้งสิ้น กลุ่มโค้ดตัวอย่างถูกแบ่งออกเป็น 2 ชุด หลักคือ ชุดกลุ่มโค้ดเริ่มต้น และชุดกลุ่มโค้ดทวนสอบ

ชุดกลุ่มโค้ดเริ่มต้นที่ใช้ในการออกแบบและสร้างวิธีการจำแนกโค้ดตามงานวิทยานิพนธ์นี้ ประกอบด้วยกลุ่มโค้ด 3 ประเภท ได้แก่ ชุดซับเซตคลีนโค้ด ชุดโค้ดที่มีความคลุมเครือ และชุดโค้ดร่องรอยที่ไม่ดี อย่างละ 20 ชุด รวมทั้งหมด 60 ชุด และนำชุดกลุ่มโค้ดทวนสอบประกอบด้วยกลุ่มโค้ด 3 ประเภทเช่นเดียวกันกับชุดโค้ดเริ่มต้นคละกันทั้งหมด 60 ชุดตัวอย่าง เพื่อใช้ตรวจสอบความถูกต้องของวิธีการที่ออกแบบไว้รวมกลุ่มโค้ดตัวอย่างที่ใช้ในงานวิทยานิพนธ์นี้ทั้งสิ้น 120 ชุด

5) ออกแบบ วิธีการ ปฏิบัติ เพื่อ ปรับปรุงโค้ดสำหรับผู้เชี่ยวชาญด้านโปรแกรม โดยเลือกใช้วิธีการปรับปรุงโค้ดใช้วิธีการรีแฟคทอริ่ง โดย Martin Fowler 6 วิธีการ ได้แก่

(1) การย้ายคุณลักษณะระหว่างวัตถุ (Moving Features between Objects)

(2) การลดความยุ่งยากของนิพจน์ที่มีเงื่อนไข (Simplifying Conditional Expression)

(3) การสร้างคุณสมบัติในการใช้งานร่วมกัน (Dealing with Generalization)

(4) การ ประกอบการทำงานของเมท็อด (Composing Method)

(5) การจัดการ โครงสร้างข้อมูล (Organizing Data)

(6) การทำให้เมท็อดเรียกใช้งานได้ง่าย (Making Method Calls Simpler)

6) มาตรวัตรซอฟต์แวร์ที่ใช้สำหรับการจำแนกกลุ่มโค้ด ยกตัวอย่างเช่น Percentage of Comment (PCC), Number Line of Codes (NLOC), Number InLine Instruction (NILI), Cyclomatic Complexity (CC), Number of interface (NOI) เป็นต้น

1.4 ขั้นตอนการวิจัย

1) การคัดเลือก มาตรวัดซอฟต์แวร์ เพื่อใช้วัดค่าของ ซับเซตคลีนโค้ด โค้ดที่มีความคลุมเครือ และร่องรอยไม่ดี

2) การสร้างกฎการจำแนกด้วยฟัซซี่โลจิกเพื่อนำค่าที่วัดจากมาตรวัดซอฟต์แวร์มาทำการแปลความเพื่อจำแนกโค้ดเป็น ซับเซตคลีนโค้ด โค้ดที่มีความคลุมเครือ และร่องรอยไม่ดี

3) การรวบรวมชุดตัวอย่างโค้ดเพื่อใช้สร้างวิธีการจำแนกด้วยกลุ่มชุดเริ่มต้น และกลุ่มชุดทวนสอบเพื่อใช้วัดความถูกต้องของวิธีการที่สร้างขึ้น

4) ออกแบบวิธีการปฏิบัติด้วยเทคนิครีแฟคทอริ่งเพื่อนำมาใช้ปรับปรุงโค้ดในกลุ่มร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือที่ผ่านการจำแนกเพื่อให้โค้ดที่ได้รับการแก้ไขมีความคุณสมบัติเป็นซับเซตคลีนโค้ด

5) การ ประเมิน ค่าดัชนีความสามารถในการบำรุงรักษากับกลุ่ม โค้ดเริ่มต้นทั้ง 3 ประเภทก่อนและหลัง การปรับปรุง เพื่อเปรียบเทียบค่าดัชนี ความสามารถในการบำรุงรักษา และนำผลลัพธ์ที่ได้มาสรุปผลต่อไป

6) การทวนสอบโดยนำกลุ่มโค้ดที่รวบรวมมา ทวนสอบวิธีการจำแนกที่สร้างขึ้นเพื่อประเมินความถูกต้องของแนวทางที่สร้างและจึงทำการประเมินค่าดัชนีความสามารถในการบำรุงรักษากับกลุ่มโค้ดที่ผ่านการทวนสอบ

7) สรุปผลการทดลอง

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1) วิธีการ จำแนกและระบุประเภทของโค้ด ที่ออกแบบนี้สามารถแยกแยะ กลุ่มซับเซตคลีนโค้ด กลุ่มร่องรอยไม่ดี และกลุ่ม โค้ดที่มีความ คลุมเครือ ทั้งนี้สามารถระบุตำแหน่งที่เกิดของร่องรอยไม่ดีและความคลุมเครือที่เกิดขึ้นในซอฟต์แวร์เพื่อช่วยสนับสนุนผู้พัฒนาในการตัดสินใจในการแก้ไขได้

2) วิธีการปฏิบัติด้วยเทคนิครีแฟคทอริ่ง สามารถปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี ให้หมดไป และสามารถนำมาประยุกต์ใช้ในการแก้ไขกลุ่มโค้ดที่มีความคลุมเครือ ให้กลับเป็นซับเซตคลีนโค้ดได้ ซึ่งวิธีการปฏิบัติที่ออกแบบไว้ถือเป็นหนึ่งในวิธีการที่จะทำให้ซอฟต์แวร์มีคุณภาพมากขึ้น

3) ในขั้นตอนการประเมินคุณภาพด้วยดัชนีความสามารถ ในการบำรุงรักษา สามารถนำมาตรวัดซอฟต์แวร์อื่นมาปรับใช้ในการวัดคุณภาพตามมาตรฐานด้านอื่นๆ ได้

1.6 บทความวิชาการที่ได้รับการตีพิมพ์

ในการทำวิทยานิพนธ์ นี้ ผู้วิจัยมีผลงานทางวิชาการร่วมกับคณะผู้วิจัย ซึ่งเป็นบทความวิชาการระดับชาติและนานาชาติ รวมเป็น 2 บทความ ดังนี้

1) บทความวิชาการเรื่อง An approach for Source Code Classification Using Software Metrics and Fuzzy Logic to improve Code Quality with Refactoring techniques ซึ่งได้รับการคัดเลือกเพื่อนำเสนอและตีพิมพ์ในงาน " The 2nd International Conference on Software Engineering and Computer Systems(ICSECS2011)" ระหว่างวันที่ 27 - 29 มิถุนายน 2554 ณ มหาวิทยาลัยมาเลเซียปะหัง รัฐปะหัง ประเทศมาเลเซีย

2) บทความวิชาการเรื่อง “An approach for source code classification to enhance maintainability” ซึ่งได้รับการคัดเลือกเพื่อนำเสนอและตีพิมพ์ในงาน “การประชุมวิชาการร่วมสาขาวิทยาการคอมพิวเตอร์และวิศวกรรมซอฟต์แวร์ ครั้งที่ 8 (The 8th International Joint Conference on Computer Science and Software Engineering: JCSSE 2011)” ระหว่างวันที่ 11 - 13 พฤษภาคม 2554 ณ คณะเทคโนโลยีสารสนเทศและการสื่อสาร มหาวิทยาลัยมหิดล วิทยาเขตศาลายา นครปฐม ประเทศไทย

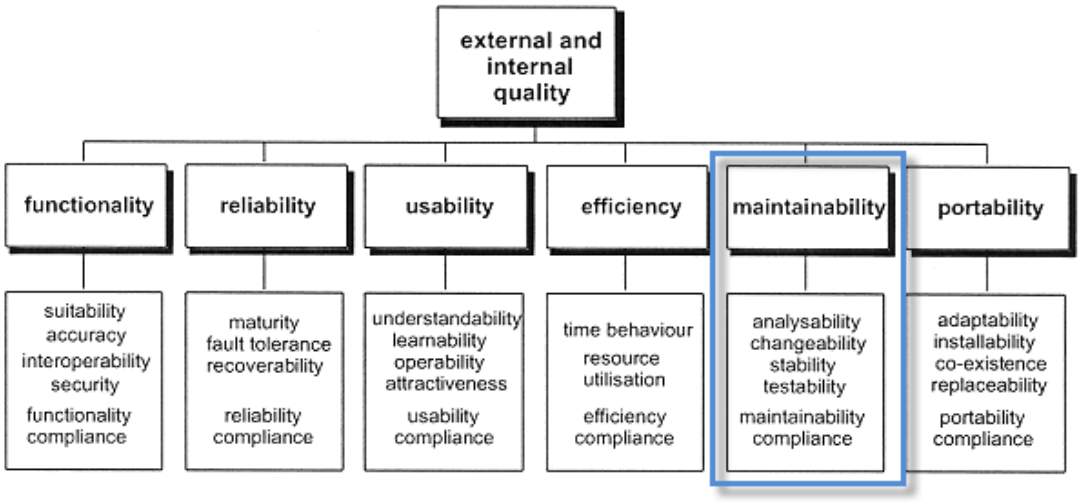
บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ปัจจัยสำคัญสำหรับงานวิทยานิพนธ์คือ การศึกษาและนำทฤษฎี งานวิจัยที่เกี่ยวข้องมาเป็นแนวทางในการออกแบบแนวทาง วิธีปรับปรุงคุณภาพโค้ด ซึ่งในงานวิทยานิพนธ์นี้นำข้อมูลสำคัญดังต่อไปนี้มาช่วยเรื่องมาตรฐานคุณภาพซอฟต์แวร์ด้านความสามารถในการบำรุงรักษา หลักการออกแบบและการเขียนโปรแกรมเชิงวัตถุ คลีนโค้ด ร่องรอยไม่ดี รีแฟคทอริ่ง มาตรวัดซอฟต์แวร์ และฟัซซี่โลจิก โดยมีรายละเอียดดังต่อไปนี้

2.1 ทฤษฎีที่เกี่ยวข้อง

2.1.1 The International Organization for Standardization and The International Electrotechnical Commission: Software Quality (ISO/IEC 9126)

จากการศึกษารายละเอียดมาตรฐาน ISO/IEC 9126 ซึ่งเป็นกรอบแนวทางในการประเมินคุณภาพของผลิตภัณฑ์ซอฟต์แวร์ทั้งภายนอกและภายใน โดย แบ่งออกเป็น 6 หัวข้อดังภาพที่ 2.1 เพื่อให้การออกแบบและพัฒนาซอฟต์แวร์มีคุณภาพตามกรอบมาตรฐาน ISO/IEC 9126



ภาพที่ 2.1 คุณสมบัติคุณภาพภายนอกและภายในตามมาตรฐาน ISO/IEC 9126 [4]

ซึ่งผู้วิจัยคำนึงถึงคุณภาพของซอฟต์แวร์ในขั้นตอนก่อนและหลังการส่งมอบงานให้กับผู้ใช้ ผู้วิจัยเห็นว่า ผู้พัฒนาซอฟต์แวร์อาจพบข้อผิดพลาดหรืออาจต้องแก้ไขปรับให้เข้ากับสภาพแวดล้อมที่มีการเปลี่ยนแปลงและจำเป็นต้องตรวจสอบข้อผิดพลาดและแก้ไขอย่างรวดเร็ว เพื่อให้ทันกำหนดการส่งมอบงาน เมื่อพิจารณาตามรายละเอียดของคุณสมบัติในแต่ละหัวข้อของ

ISO/IEC 9126 แล้ว ผู้วิจัยเห็นว่า คุณภาพด้านความสามารถในการบำรุงรักษามีส่วนเกี่ยวข้องในเรื่องการตรวจสอบข้อผิดพลาดและปรับปรุงซอฟต์แวร์ เนื่องจากคุณสมบัติย่อยเพื่อการประเมินคุณภาพด้านความสามารถในการบำรุงรักษานั้นประกอบไปด้วย

1) ความสามารถในการวิเคราะห์ (Analyzability) ระดับความยากหรือง่ายที่จะสามารถวิเคราะห์ระบุหรือระบุว่าองค์ประกอบใดต้องการปรับปรุง

2) ความสามารถในการปรับเปลี่ยน (Changeability) ระดับความยากหรือง่ายที่สามารถปรับให้เข้ากับระบบ หากมีการเปลี่ยนแปลง

3) ความมีเสถียรภาพ (Stability) ระดับความยากหรือง่ายที่สามารถทำให้ระบบคงที่ในระหว่างการซ่อมแซมปรับปรุง

4) ความสามารถทดสอบได้ (Testability) ระดับความยากหรือง่ายที่สามารถทดสอบได้หลังจากปรับปรุงระบบ

5) ความสอดคล้องในเรื่องการบำรุงรักษา (Maintainability Conformance) ระดับความยากหรือง่ายที่สามารถทำให้ระบบสอดคล้องกับมาตรฐานหรือการคำนึงถึงเรื่องความสามารถในการบำรุงรักษา

งานวิทยานิพนธ์นี้ จึง เลือกให้ความสำคัญกับความสามารถด้านการบำรุงรักษา (Maintainability) ซึ่งเป็นความสามารถที่ทำให้ซอฟต์แวร์ปรับเปลี่ยนแก้ไขปัญหาต่างๆได้อย่างรวดเร็วตามมาตรฐาน ISO/IEC 9126

2.1.2 หลักการออกแบบและการเขียนโปรแกรมเชิงวัตถุ

จากคำกล่าวของเอลัน เคย์ (Alan Kay) [12, 13, 14] นักคอมพิวเตอร์ด้าน ออกแบบเขียนโปรแกรมเชิงวัตถุและการออกแบบส่วนต่อประสานกราฟิกกับผู้ใช้ Graphical User Interface (GUI) ว่า "ทุกๆ สิ่งที่เป็นวัตถุหรือโปรแกรม คือ กลุ่มของวัตถุที่ส่งข้อมูลข่าวสารโต้ตอบระหว่างกันเพื่อทำงานร่วมกัน " ดังนั้น การเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming: OOP) คือหนึ่งในรูปแบบการพัฒนาโปรแกรมคอมพิวเตอร์ที่ให้ความสำคัญกับการทำงานร่วมกันของวัตถุ โดยสามารถนำวัตถุมาประกอบกันเพื่อแลกเปลี่ยนข่าวสารแล ะประมวลผลกับวัตถุต่างๆ ที่เกี่ยวข้อง

อย่างไรก็ตามจุดประสงค์หลักของการเขียนโปรแกรมเชิงวัตถุก็คือ การนำส่วนประกอบต่างๆ ของวัตถุที่มี นำกลับมาใช้ใหม่ (Reuse) ซึ่งวัตถุที่สร้างขึ้นมาอยู่ก่อนนั้นมีความยืดหยุ่น (Flexible) นำมาประกอบกันเป็นวัตถุอีกชนิดหนึ่งเพื่อให้มีความสามารถ (Capability) มากกว่า

เดิม ทำให้ลดระยะเวลาในการเขียนโปรแกรม อีกทั้งส่งผลให้โปรแกรมใช้งานได้ตรงตามความต้องการอย่างมีประสิทธิภาพ เพื่อให้การเขียนโปรแกรมเชิงวัตถุเป็นไปตามรูปแบบที่มีคุณภาพ จำเป็นต้องอาศัยหลักการออกแบบและเขียนโปรแกรมเชิงวัตถุ 5 หลักการ [9] ได้แก่

1) Single Responsibility Principle (SRP)

การเขียนโปรแกรมเชิงวัตถุทั่วไป กำหนดให้คลาสหนึ่งๆ ควรมีหน้าที่รับผิดชอบเพียงหนึ่งอย่าง หรือมีการตอบสนองต่อการเปลี่ยนแปลงเมื่อเพียงหนึ่งอย่างเท่านั้น โดยหลักการทำงาน ของโมดูลและคลาสต้องทำงานแยกกันซึ่งถือว่าเป็นหลักการทั่วไปแต่ทำให้ถูกต้องนั้นทำได้ยาก การนำงานมารวมให้เป็นกลุ่มเดียวกันสามารถทำได้ง่าย จึงเป็นเหตุให้ที่ผู้ออก แบบซอฟต์แวร์ควรระมัดระวัง

2) Open/Closed Principle (OCP)

ซอฟต์แวร์ที่มีองค์ประกอบย่อยๆได้แก่ คลาส (Class), โมดูล (Module), ฟังก์ชัน (Function) และอื่นๆ จำเป็นต้องมีคุณลักษณะการทำงานที่สำคัญ 2 ส่วน ได้แก่

(1) การอนุญาตให้ส่วนหนึ่งของโมดูลสามารถขยายได้ (Extension) คือ เป็นหลักการที่กำหนดให้ส่วนหนึ่งส่วนใดภายในคลาสหนึ่งๆ สามารถขยายและเพิ่มเติมได้ ซึ่งเป็นการตอบสนองต่อการเปลี่ยนแปลงตามความต้องการในการใช้งาน

(2) การไม่อนุญาตให้ส่วนหนึ่งของโมดูลได้รับการแก้ไขต่อเติม (Modification) เป็นการป้องกันการขยายโมดูลที่ก่อให้เกิดผลกระทบถึงโมดูลในส่วนอื่นๆ โดยหลักการ Open/Closed ถือเป็นหัวใจของการเขียนโปรแกรมเชิงวัตถุ ซึ่งทำให้เกิดผลลัพธ์ด้านความยืดหยุ่น (Flexibility) ความสามารถนำกลับมาใช้งาน(Reusability)และความสามารถในการบำรุงรักษา (Maintainability) ซอฟต์แวร์ให้ใช้งานได้นาน

3) Liskov Substitution Principle (LSP)

หลักการ Liskov เป็นหลักการที่สำคัญและเป็นส่วนส่งเสริมหลักการของ Open/Closed ที่อนุญาตให้สามารถทำการแก้ไขพฤติกรรมการทำงานของคลาสได้โดยไม่ส่งผลกระทบทำให้ชนิดข้อมูลพื้นฐานของคลาสนั้นถูกเปลี่ยนแปลงไปจากเดิม หรือ ยังคงคุณสมบัติพื้นฐานเดิมของชนิดข้อมูลไว้ ซึ่งสามารถจัดการได้ โดยปรับเปลี่ยนลำดับที่อยู่ของคลาสในแผนภูมิต้นไม้ให้มีความเหมาะสมสำหรับการสืบทอดพฤติกรรมและการทำงานของแต่ละคลาส ในการปรับเปลี่ยนลำดับตำแหน่งนี้ ผู้พัฒนาซอฟต์แวร์ จำเป็นต้องเข้าใจพื้นฐานของโค้ดเป็นอย่างดี เพื่อกำหนด แบบชนิดย่อย(Subtype) ไม่ให้กว้างเกินไป โดยการดึงนิยามของวัตถุนั้นๆ ให้แสดงพฤติกรรมได้อย่างชัดเจน

4) Dependency Inversion Principle (DIP)

หลักการนี้เป็นการอ้างถึงรูปแบบเฉพาะของ Decoupling โดยแสดงความสัมพันธ์พื้นฐานของโมดูลระดับสูง ในการกำหนดการตั้งค่าในแต่ละโมดูล วัตถุประสงค์นี้เพื่อแสดงระดับสูง-ต่ำ และความเป็นอิสระต่อกันของโมดูล มีเกณฑ์ดังนี้

(1) โมดูลระดับสูงไม่ควรทำงานขึ้นกับโมดูลระดับต่ำ โดยทั้งสองต้องขึ้นกับการกำหนดสาระสำคัญ (Abstractions)

(2) การกำหนดสาระสำคัญไม่ควรทำงานขึ้นอยู่กับรายละเอียด แต่รายละเอียดต้องเป็นส่วนที่ขึ้นกับการกำหนดสาระสำคัญสำหรับหลักการนี้สามารถช่วยให้โค้ดและขอบเขตของงานสามารถนำกลับไปใช้ได้อีก ง่ายต่อการแก้ไขปรับปรุง

5) Interface Segregation Principle (ISP)

เป็นหลักการที่ช่วยแก้ปัญหาจำนวนของส่วนต่อประสาน(Interface) จำนวนมากที่ประกาศไว้ในคลาสเป็นเหตุที่ทำให้เกิดปัญหาโคฮีชันขึ้น เมื่อมีการเรียกใช้งานส่วนต่อประสานที่ประกาศไว้เป็นจำนวนมาก อาจมีบางส่วนต่อประสานที่มีพฤติกรรมขัดแย้งกันเองเนื่องจากการเรียกใช้งานของส่วนประสานที่มีจำนวนมาก ดังนั้นควรทำการตัดแบ่งส่วนประสานให้มีขนาดเล็กลงหรือแบ่งกลุ่มส่วนประสานตามชนิดหรือประเภทของการใช้งาน เพื่อแก้ไขปัญหาตามหลักการนี้

จากแนวคิดของผู้เชี่ยวชาญในอุตสาหกรรมซอฟต์แวร์และหลักการออกแบบและเขียนโปรแกรมเชิงวัตถุนี้ถูกนำมาปรับใช้เพื่อเป็นเกณฑ์ในการจำแนกและระบุลักษณะของคลีนโค้ดซึ่งมีคุณสมบัติตรงตามหลักการออกแบบที่ดี และเป็นแนวทางสำหรับการสร้างกฎการแปลความด้วยฟัซซี่โลจิก สำหรับการจำแนกโค้ดที่มีลักษณะในทางตรงกันข้ามคลีนโค้ดนั้น ได้รวบรวมจากความรู้จากผู้เชี่ยวชาญเพื่อสร้างเกณฑ์สำหรับการเลือกร่องรอยไม่ดีและความคลุมเครือที่มีลักษณะไม่เป็นตามหลักการออกแบบที่ดี

2.1.3 **คลีนโค้ด** (Clean Code)

ลักษณะของโค้ดที่ดีและตรงตามแบบแผนคือ มีการออกแบบและเขียนอย่างเป็นระบบ มีระเบียบในการจัดองค์ประกอบอย่างดี ทำให้เข้าใจและสามารถปรับเปลี่ยนโค้ดได้ง่าย จากแนวคิดคลีนโค้ดของ Robert C. Martin [9] ได้รวบรวมความคิดจากผู้เชี่ยวชาญการพัฒนาซอฟต์แวร์ชั้นนำ โดยมีรายละเอียดดังต่อไปนี้

1) มโนภาพ (idea) ของคลีนโค้ด

(1) ความระมัดระวัง, การเอาใจใส่ (Care)

Michael Feathers, ผู้เขียนหนังสือ Working Effectively with Legacy กล่าวไว้ว่า ความเอาใจใส่ในการเขียนโค้ดนั้นคือ จำเป็นต้องใช้เวลาเพื่อที่จะให้โค้ดมีลักษณะที่เรียบง่าย มีรายละเอียดที่เหมาะสมตรงตามวัตถุประสงค์ของการพัฒนาซอฟต์แวร์ในแต่ละโครงการ

(2) ความเรียบง่ายผสมผสานกับความสวยงาม (Elegant)

Bjarne Stroustrup ผู้ให้กำเนิด ภาษา C++ และผู้เขียนหนังสือ The C++ Programming Language แสดงความคิดเห็นเกี่ยวกับคลีนโค้ดไว้ว่า เป็นโค้ดที่มีรายละเอียด เน้นความเรียบง่าย อ่านเข้าใจง่าย

(3) สะดวกต่อการปรับปรุงให้ดีขึ้น (Easy to enhance)

Dave Thomas, ผู้ก่อตั้ง Object Technology International Inc. (OTI) ผู้ริเริ่มกลยุทธ Eclipse ได้ให้ความเห็นว่า คลีนโค้ดควรเป็นโค้ดที่ง่ายต่อการนำไปปรับปรุง ลักษณะโค้ดเช่นนี้จะทำให้เห็นถึงความแตกต่างระหว่างโค้ดที่อ่านง่ายและโค้ดที่สามารถปรับเปลี่ยนได้ง่าย นอกจากนี้ไม่ว่าโค้ดนั้นจะมีความสวยงาม อ่านเข้าใจ หรือสามารถเข้าถึงได้ง่าย ถ้ายังไม่ผ่านการทดสอบ จะถือว่ายังเป็นโค้ดที่ไม่สมบูรณ์

(4) มีประสิทธิภาพ (Efficient)

Bjarne Stroustrup กล่าวว่า นอกจากโค้ดที่มีลักษณะเรียบง่ายและอ่านเข้าใจแล้ว Bjarne ให้ความสำคัญกับเรื่องประสิทธิภาพ เพราะตรรกะ (Logic) ที่ใช้ในการพัฒนาซอฟต์แวร์จะต้องมีกระบวนการพุ่งเป้าไปที่การจัดการ จุดบกพร่อง (Bug) ความสัมพันธ์ของกระบวนการทำงานที่ขนาดเล็กจะช่วยลดภาระเรื่องการบำรุงรักษา การควบคุมข้อผิดพลาดที่เกิดขึ้นระหว่างการทำงานจะสำเร็จด้วยดี ขึ้นอยู่กับกลยุทธที่วางแผนรองรับ แต่การปฏิบัติ ที่ดีที่สุดคือ การป้องกันไม่ให้ทีมงานเข้าไปทำความเสียหายให้กับโค้ด ในแต่ละฟังก์ชัน, คลาส, และโมดูล

(5) ง่ายต่อการอ่าน (Readability)

ความเห็นของ Grady Booch, ผู้เขียนหนังสือ Object Oriented Analysis and Design with Applications กล่าวว่า คลีนโค้ดควรเป็นโค้ดที่อ่านเข้าใจ ซึ่งไม่ควรปิดกั้นแนวคิดการสร้างสรรค์งานของผู้พัฒนาซอฟต์แวร์แต่ควรที่จะมีกรอบและการควบคุมที่ชัดเจน เพราะฉะนั้นคลีนโค้ดควรเป็นโค้ดที่เข้ามาช่วยแก้ไขปัญหาข้อผิดพลาดที่เกิดขึ้น

(6) **ความสละสลวยของรายละเอียด (**Like a Well-Written Prose)

Dave Thomas ผู้ก่อตั้ง OTI และผู้นำแนวคิดและเผยแพร่กล ยุทธ์ Eclipse ได้แสดงความคิดเห็นเกี่ยวกับคลีนโค้ดว่า โค้ดควรจะมีขนาดที่เล็กมากกว่าขนาดใหญ่ และการเขียนโค้ดก็เหมือนการเรียบเรียงเขียนหนังสือ ซึ่งการเขียนโค้ดควรทำให้ผู้อ่านสามารถเข้าใจได้ง่าย

(7) **ไม่มีการทำซ้ำหรือการทำให้ซับซ้อน** (No Duplications)

Ron Jefferies ผู้เขียนหนังสือ Extreme Programming Installed and Extreme Programming Adventures in C# แสดงความคิดเกี่ยวกับคลีนโค้ดว่า เป็นโค้ดที่สามารถทำการทดสอบได้, มีรายละเอียดที่ไม่ซับซ้อน อีกทั้งยังแสดงโครงสร้างของระบบได้อย่างชัดเจน มีจำนวนเอนทิตี คลาส เมท็อด ฟังก์ชัน ที่ไม่มากเกินไป

(8) **เรียบง่ายและเป็นระบบ** (Simple and Direct)

Michael Feathers นอกจากได้กล่าวถึงการใส่ใจในการเขียนโค้ด ยังเสนอข้อแนะนำอื่นว่า ลักษณะของโค้ดนั้นควรจะมีความเรียบง่ายและเป็นระบบ

(9) **สามารถมองเห็นและรับรู้ปัญหา** (Makes the language look like it was made for the problem)

Ward Cunningham ผู้สร้าง Wiki, eXtreme Programming และยังเป็นผู้นำแนวคิดการออกแบบและการเขียนโปรแกรมเชิงวัตถุ ได้ให้ความเห็นเกี่ยวกับ คลีนโค้ด ว่า การเขียนโค้ดที่เน้นความสวยงาม อาจจะทำให้เกิดปัญหาได้ ดังนั้นผู้พัฒนาบางคนมองว่า โค้ดที่มีความสวยงาม (Beautiful Code) ก็มักจะมองข้ามผ่านไป ซึ่งในความเป็นจริงโค้ดนั้นควรมีลักษณะที่เรียบง่าย

2) **แบบรูปคลีนโค้ด** (Clean code Patterns)

(1) Meaningful Names

เป็นการตั้งชื่อให้กับทุกอย่างที่อยู่ภายในโปรแกรมโดยชื่อนั้นต้องสื่อความหมายให้ผู้อ่านเข้าใจว่า ส่วนนั้นหรือวัตถุนั้นมีหน้าที่งานอะไร ทำงานอย่างไร และเชื่อมโยงไปส่วนใดในโปรแกรมบ้าง โดยลักษณะของชื่อที่ดีนั้น ควรมีลักษณะที่เป็นคำที่มีความหมาย, ไม่เป็นอักษรย่อ, มีขนาดไม่ยาวจนเกินไป, ไม่ต้องผ่านการตีความเข้ารหัส (Encode) และคำนั้นต้องไม่มีความหมายที่ตีความได้หลายความหมาย

(2) Comments

การอธิบายรายละเอียดในแต่ละส่วนของโปรแกรมเพื่อบอกวัตถุประสงค์แก่ผู้อ่านให้เข้าใจว่า โค้ด ตรงตำแหน่งนี้เป็นอย่างไร ทำงานอย่างไร มีข้อควรระวังอะไรบ้าง หรือได้ผ่านการแก้ไข แต่เนื้อหาไม่ควรมีความยาวเกินไปและไม่ควรเขียนอธิบายทุกส่วนในโปรแกรม

(3) Formatting

การเขียนโค้ดที่ดีนั้นควรมีรูปแบบ รายละเอียด แบบแผนที่มีการวางลำดับก่อนหลัง เพื่อให้คนอ่านโค้ดได้ง่ายและเข้าใจ เหมือนกับการอ่านบทความในหนังสือ

(4) Error Handling

เมื่อเกิดข้อผิดพลาด ควรใช้วิธี Exception เพื่อดักจับและอธิบายรายละเอียดว่าเป็นข้อผิดพลาดประเภทใด และเชื่อมโยงไปยังจุดใดในโปรแกรม ถ้าหากใช้วิธีหาข้อผิดพลาด (Error Code) หรือ ส่งค่ากลับ (Return Code) จะไม่สามารถระบุข้อผิดพลาดที่เกิดขึ้นได้ เพราะฉะนั้นควรใช้ Exception มากกว่าเพื่อลดข้อ ผิดพลาดในการทำงาน ซึ่งในการเขียนโปรแกรมทางปฏิบัติที่ดีนั้นควรเขียนแบบ Try-catch เพื่อระบุรายละเอียด ข้อควรทำเมื่อเกิดข้อผิดพลาด และแนะนำวิธีการแก้ไข การทำงานบางครั้งหรือบางส่วนไม่มีความจำเป็นที่ต้องตรวจสอบข้อผิดพลาด เนื่องจากต้องการให้ตัวแปรทำงานตามวิธีที่มีการพ้องรูป (Polymorphism) เมื่อเกิดข้อผิดพลาด ไม่ว่าจะเกิดตรงจุดใด เมื่อไหร่ให้มีคำอธิบายกำกับไว้ เพื่อคนอ่านทำความเข้าใจโค้ด ได้ทราบข้อผิดพลาดนี้ ว่าเชื่อมโยงไปยังจุดใดในโปรแกรมบ้าง นอกจากนี้ ถ้าหากเกิดข้อผิดพลาด ไม่ควรส่งผลกลับด้วยค่าว่าง (NULL) เพราะไม่สามารถตีความได้ว่า เป็นข้อผิดพลาดประเภทใด และไม่ควรส่งค่าว่าง ผ่านไปยังส่วนอื่นๆ เพราะค่าว่างนั้นเป็นเพียงชนิดของข้อมูล

(5) Functions

(5.1) Small Function

ฟังก์ชันควรมีขนาดเล็ก กระชับเพื่อความยืดหยุ่นโดยจะส่งผลให้เกิดการทำงานมีลักษณะหนึ่งต่อหนึ่ง (1: 1)

(5.2) Do One Thing

หนึ่งฟังก์ชันควรมีความรับผิดชอบทำงานเพียงหนึ่งหน้าที่ โดยระบุหน้าที่ชัดเจนเพื่อไม่ให้เกิดการทำงานที่ซ้ำซ้อนกัน

(5.3) One Level of Abstraction per Function

ฟังก์ชันที่มีการถูกเรียกใช้งานจากวัตถุใดๆ ที่เกี่ยวข้อง ถ้าวัตถุนั้นมีระดับขอ ง Abstraction มากเกินไป ทำให้การใช้งานมีช่วงคำสั่งเรียกใช้ที่ยาวขึ้น ทำให้เวลาการทำงานมากขึ้น

(5.4) Nesting Depth

การทำงานของ Nesting Depth ควรเป็นแบบเงื่อนไข (Condition) ที่สร้างขึ้นเพื่อตัดสินใจและเลือกทำงานอย่างอิสระ โดยไม่ซ้ำซ้อนในส่วนอื่นๆ

(5.5) Function Arguments

การที่พารามิเตอร์ในแต่ละฟังก์ชันไม่ควรมีความยาวมากกว่า 3 ตัว เพราะจะทำให้การทำงานของคลาสแยกตัวกระจายหรือเพิ่มความซับซ้อนให้กับการทำงานมากขึ้น

(5.6) Have No Side Effects

ฟังก์ชันแต่ละส่วนควรมีอิสระต่อกัน ไม่ควรมีผลกระทบต่อฟังก์ชันอื่นๆ รวมไปถึงเมื่อเกิดการแก้ไขจะไม่ส่งผลกระทบต่อส่วนอื่นๆ

(5.7) Structured Programming

การเขียนโปรแกรมต้องมีการทำงานในรูปแบบ Sequential State เพื่อป้องกันไม่ให้เกิดการทำงานที่ซ้ำซ้อนกัน

(6) Objects and Data Structures

(6.1) Data Abstraction

การทำงานของแต่ละวัตถุจำเป็นต้องมีการปกปิดข้อมูลหรือป้องกันข้อมูลที่สำคัญสำหรับการใช้งาน แต่สามารถเข้าใจได้ว่า ข้อมูลนั้นมีไว้เพื่อวัตถุประสงค์ใด และทำงานอย่างไร

(6.2) Data/Object Anti-Symmetry

การกำหนดและแยกให้ชัดเจนระหว่างข้อมูลและวัตถุภายในโปรแกรม เพื่อประโยชน์ในการจัดเก็บข้อมูลและยังสามารถระบุ ข้อมูลตรงส่วนใดสามารถแสดงผลได้และข้อมูลส่วนใดควรซ่อนไว้

(6.3) Data Transfer Objects

ข้อมูลส่งผ่านกันระหว่างแต่ละวัตถุ ต้องมีรูปแบบที่ชัดเจน โดยสามารถระบุถึงหน้าที่ว่าใครทำหน้าที่ส่งข้อมูล และใครทำหน้าที่จัดเก็บข้อมูล แต่ในการจัดเก็บข้อมูลนั้นเมื่อข้อมูลถูกปรับเปลี่ยนจำเป็นต้องเปลี่ยนข้อความสั่ง (Statement) ตามด้วย

(7) Classes

(7.1) Class Organization

คลาสที่ดีนั้นต้องมีรูปแบบ เรียบเรียงตัวแปร เมท็อด เพื่อแสดงถึงเมท็อดใดถูกสืบทอดหรือถูกปรับการทำงานใหม่(Override) มาจากที่ใด

(7.2) Encapsulation

ข้อมูลของคลาส วัตถุในคลาสจำเป็นต้องมีการปกปิดข้อมูล (Information Hiding)

(7.3) Classes Should Be Small!

คลาสควรมีขนาดเล็กและกระชับ เพื่อให้การทำงานมีลักษณะเป็นหนึ่งต่อหนึ่ง (1: 1) และมีหน้าที่งานชัดเจน เพื่อลดความซ้ำซ้อนของโปรแกรม

(7.4) Maintaining Cohesion

คลาสหนึ่งๆ ภายในประกอบไปด้วยตัวแปร เมท็อด ต่างๆ จำเป็นต้องมีความสัมพันธ์แบบสมานฉันท์คือ ฟังก์ชัน หรือ เมท็อด ควรมีการเรียกใช้งานตัวแปรที่อยู่ภายในคลาสนั้น

(7.5) Organizing for Change

มีการวางแผนรองรับ เมื่อเกิดการเปลี่ยนแปลง โดยสามารถระบุได้ว่าการเปลี่ยนแปลงจะส่งผลกระทบต่อจุดใดบ้างภายในโปรแกรม

(7.6) The Law of Demeter

วัตถุใดๆ ที่สามารถทดแทนได้ โดยยังคงนิยามของวัตถุนั้นไว้ เมื่อมีการเรียกใช้งานก็สามารถเข้าใจได้ว่า วัตถุนั้นทำงานอย่างไร

(7.7) Clean Boundaries

การทำงานของแต่ละคลาส ภายในโปรแกรมหนึ่งๆ ต้องมีรายละเอียดขอบเขตที่ชัดเจนว่า ส่วนใดมีหน้าที่ทำอะไร และเชื่อมโยงไปถึงส่วนใดในโปรแกรม และเมื่อมีกา ร เปลี่ยนแปลงเกิดขึ้น ทุกๆ ส่วนจำเป็นต้องรู้สถานะ(State) ของตนเอง

(8) Emergence

(8.1) Simple

ลักษณะของโปรแกรมควรมีความเรียบง่าย เข้าใจง่าย ไม่ซับซ้อน เมื่อมีการเปลี่ยนแปลงหรือเมื่อมีการแก้ไข จะสามารถทำได้ง่าย โดยไม่ส่งผลกระทบหรือทำให้เกิดข้อผิดพลาดต่อส่วนอื่นๆ ในโปรแกรม

(8.2) No Duplication

ไม่มีส่วนหนึ่ง ส่วนใดภายในโปรแกรมที่ทำงานซับซ้อน

(8.3) Minimal Classes and Methods

การเขียนโปรแกรมให้แต่ละส่วนต้องมีความกระชับ เพื่อตอบสนองการทำงานแบบ Single Responsibility ในอนาคต ทำให้สามารถรองรับการเปลี่ยนแปลงได้อย่างคล่องตัว

(9) Boundaries

(9.1) Using Third-Party Code

การกำหนดขอบเขตของส่วนต่อประสานระหว่างผู้ดำเนินการจัดหาและผู้ใช้งาน ซึ่งโค้ดหรือระบบที่นำมาใช้งานจะพัฒนาขึ้นเองหรือใช้บริการจากภายนอกก็ได้ แต่จำเป็นต้องเชื่อมโยงกันและเข้าใจการทำงานของกันและกันชัดเจน

(9.2) Exploring and Learning Boundaries

การพัฒนาโปรแกรมโดยใช้บริการจากภายนอก (Outsource) นั้น ควรศึกษารายละเอียดของโค้ดและควรทดสอบโปรแกรมหรือระบบก่อนนำไปใช้งานจริง ถ้าหากมีข้อบกพร่องจะได้แก้ไขได้ทันที

(9.3) Learning log4j

การศึกษา log4j เพื่อนำไปใช้งานในส่วนต่อประสาน ซึ่งจะสามารถ encapsulate คลาส และง่ายต่อการทดสอบโค้ดที่พัฒนาขึ้น

(9.4) Learning Tests Are Better Than Free

ควรให้ความสำคัญกับการทดสอบทั้งโปรแกรมที่พัฒนาขึ้นเอง ยิ่งโดยเฉพาะโปรแกรมที่ใช้บริการจากภายนอก เพื่อเรียนรู้ข้อแตกต่างและหากพบจุดบกพร่องควรศึกษาหาวิธีการแก้ไข

(9.5) Using Code That Does Not Yet Exist

การกำหนดขอบเขตของการเขียนโค้ดมีหลายรูปแบบ โดยเฉพาะการนำส่วนต่อประสานมาช่วยควบคุม เพื่อสะดวกแก่การทดสอบและง่ายต่อการทำความเข้าใจในตัวโค้ดแต่ละส่วนที่ประกอบเข้าด้วยกัน

(9.6) Clean Boundaries

การเขียนโค้ดที่มีการกำหนดขอบเขตไว้ จะทำให้ง่ายต่อการควบคุม แต่ถ้าต้องใช้บริการจากภายนอก ควรแยกส่วนกับโค้ดที่พัฒนาขึ้นเอง เมื่อต้องการเรียกใช้งานควรใช้ส่วนต่อประสานเชื่อมต่อ เพราะถ้าหากมีข้อผิดพลาดเกิดขึ้นกับโค้ดที่มาจากภายนอกและต้องการแก้ไข ซึ่งถ้าแยกส่วนไว้จะทำให้การบำรุงรักษาง่ายขึ้น

จากแนวคิด มโนภาพ และแบบรูป คลีนโค้ดตามที่ Robert C. Martin ถือว่าเป็นโค้ดที่สามารถทำงานตามหลักการออกแบบโปรแกรมเชิงวัตถุ ผู้วิจัยได้วิเคราะห์จนสามารถแบ่งแยกแบบรูปออกมากลุ่มหนึ่ง จึงขอเรียกกลุ่มคลีนโค้ดนี้ว่า ซับเซตคลีนโค้ด (Subset Clean code) ซึ่งเป็นแบบรูปที่มีผลกระทบต่อคุณภาพด้านความสามารถในการบำรุงรักษา ผู้วิจัยจึง เกิดแรงผลักดันในการค้นหาและสร้างเครื่องมือเพื่อจำแนกและระบุโค้ด โดยนำคุณสมบัติและลักษณะของคลีนโค้ดมาใช้สร้างเกณฑ์ การจำแนก ให้กับกลุ่มโค้ด ตัวอย่าง ที่จะนำมาทดสอบ ในงานวิทยานิพนธ์นี้

2.1.4 ร่องรอยไม่ดี

จากแนวความคิดของ Kent Beck และ Martin Fowler [14] นิยามร่องรอยไม่ดีไว้ว่า “โค้ดที่เป็นส่วนที่แสดงและบ่งชี้ถึงปัญหาที่จะเกิดขึ้นกับระบบ หรือเป็นการแสดงอาการปัญหาของโปรแกรมที่เกิดขึ้นกับโค้ดที่จะนำไปสู่ปัญหาที่ยากต่อการแก้ไข ” แม้ว่าจากคำนิยามจะสรุปได้ว่าร่องรอยไม่ดีเป็นโค้ดที่นักพัฒนาระบบต้องให้ความใส่ใจ ตรวจสอบ และแก้ไข เนื่องจากส่งผลถึงระบบหรือโปรแกรม เพราะลักษณะของโค้ดที่มีความซับซ้อน เข้าใจยาก และถูกพัฒนาขึ้นมาอย่างไม่เป็นระบบ ส่งผลให้เกิดปัญหาต่อการออกแบบโครงสร้างและการเขียนโค้ด [6, 8, 15] ร่องรอยไม่ดีของโค้ดสามารถแบ่งกลุ่มได้ตามรายละเอียด ดังนี้

1) The Bloaters

ร่องรอยไม่ดีชนิดนี้ที่ถูกจัดอยู่ในกลุ่มที่มีลักษณะขนาดใหญ่ ซับซ้อนและทำให้โปรแกรมมีการทำงานไม่พึงประสงค์ ร่องรอยไม่ดีที่ถูกจัดอยู่ในกลุ่มนี้ได้แก่ Long Method Large Class Primitive Obsession Long Parameter List และ Data Clumps

2) The Object-Orientation Abusers

ร่องรอยไม่ดีประเภทหนึ่งที่มีสาเหตุปัญหามาจากการออกแบบ ส่งผลให้ส่วนต่างๆ ภายในโปรแกรมทำงานผิดพลาด ร่องรอยไม่ดีถูกจัดอยู่ในกลุ่มนี้ได้แก่ Switch Statements Temporary Field Refused Bequest Alternative Classes with Different Interfaces และ Parallel Inheritance Hierarchies

3) The Change Preventers

ร่องรอยไม่ดีปรับเปลี่ยนแก้ไขได้ยากและข้อผิดพลาดนี้ส่งผลกระทบต่อหลายส่วนในโปรแกรมร่องรอยไม่ดีถูกจัดอยู่ในกลุ่มนี้ได้แก่ Divergent Change และ Shotgun Surgery

4) The Dispensables

ร่องรอยไม่ดีที่มีลักษณะทำงานซ้ำซ้อน หรือเป็นส่วนที่สร้างขึ้นมาแต่ไม่ถูกใช้งาน ซึ่งควรจะลบออกไปจากโปรแกรม ร่องรอยไม่ดีจัดอยู่ในกลุ่มนี้ได้แก่ Lazy class Data class Duplicate Code และ Speculative Generality

5) The Encapsulators

ลักษณะร่องรอยไม่ดี ประเภทนี้จัดเป็นส่วนในการเชื่อมโยงข้อมูลต่างๆ ภายในโปรแกรม เช่น หากมีการเรียกใช้งานในจุด A ก็ต้องเชื่อมข้อมูลมาจากจุด B ซึ่งถ้าพบข้อผิดพลาดจะต้องพิจารณาแก้ไขทุกๆ ส่วนที่เชื่อมโยงกัน ร่องรอยไม่ดีที่ถูกจัดอยู่ในกลุ่มนี้ ได้แก่ Message Chains และ Middle Man

6) The Couplers

ร่องรอยไม่ดีสามารถตรวจพบได้จากโปรแกรมมีค่าคัปปลิงสูง ซึ่งผิดหลักการออกแบบโปรแกรม ร่องรอยไม่ดีที่จัดอยู่ในกลุ่มนี้ได้แก่ Feature Envy และ Inappropriate Intimacy

7) Others

ร่องรอยไม่ดีอื่นๆ เนื่องจากมีคุณสมบัติที่ซึ่งไม่สามารถจัดเข้ากลุ่มได้ ได้แก่ Incomplete Library Class และ Comments

ทั้งนี้ร่องรอยไม่ดีที่เกิดขึ้นมาจำนวนมาก และยากต่อการทำความเข้าใจทำให้ Mika Mäntylä [8] ได้เสนองานวิจัยเพื่อจัดกลุ่ม และแยกแยะสาเหตุที่มาของร่องรอยไม่ดี โดยพิจารณาวิเคราะห์ข้อมูลที่นำมาจาก Martin Fowler โดยสรุปเป็นตารางความเป็นไปได้ของในการแก้ไขร่องรอยไม่ดีโดยเรียงลำดับความยากจาก 0 จนถึง 5 พร้อมทั้งแนะนำมาตรวัดซอฟต์แวร์ในที่ใช้ในการระบุร่องรอยไม่ดีในบางประเภท ซึ่งสามารถสรุปได้ดังตาราง ที่ 2.1 ดังต่อไปนี้

ตารางที่ 2.1 ระดับความเป็นไปได้ในการแก้ไขร่องรอยไม่ดีตามแนวคิดของ Mika Mäntylä [8]

ระดับ	รายการร่องรอยไม่ดี	ชื่อกลุ่ม
0	1. Primitive Obsession	The Bloaters
	2. Data Clumps	The Bloaters
	3. Refused Bequest	The Object-Orientation Abusers
	4. Alternative Classes with Different Interfaces	The Object-Orientation Abusers
	5. Parallel Inheritance Hierarchies	The Object-Orientation Abusers
	6. Divergent Change	The Change Preventers
	7. Shotgun Surgery	The Change Preventers
	8. Incomplete Library Class	Others
1	1. Bad Comment	Others
2	1. Middle Man	The Encapsulators
3	1. Temporary Field	The Object-Orientation Abusers
	2. Switch Statements	The Object-Orientation Abusers

ตารางที่ 2.1 ระดับความเป็นได้ในการแก้ไขร่องรอยไม่ดีตามแนวคิดของ Mika Mäntylä [8] (ต่อ)

ระดับ	รายการร่องรอยไม่ดี	ชื่อกลุ่ม
3	3. Speculative Generality	The Dispensables
	4. Message Chains	The Encapsulators
4	1. Large Class	The Bloaters
	2. Lazy class	The Dispensables
	3. Data class	The Dispensables
	4. Duplicate Code	The Dispensables
	5. Feature Envy	The Couplers
	6. Inappropriate Intimacy	The Couplers
5	1. Long Method	The Bloaters
	2. Long Parameter List	The Bloaters

ลักษณะของร่องรอยไม่ดีทั้ง 22 รายการ แสดงไว้ในภาคผนวก ก. แสดงถึงคุณสมบัติของโค้ดที่มีปัญหาและแนวทางแก้ไขว่า ควรจัดการในจุดใด ทำให้สามารถสร้าง วิธีการแก้ไขโค้ดให้เป็นไปอย่างถูกต้องตามหลักการออกแบบและเขียนโปรแกรมเชิงวัตถุ

2.1.5 **รีแฟคทอริ่ง** (Refactoring)

กระบวนการทำรีแฟคทอริ่ง [6] เกิดขึ้นเพื่อให้สามารถปรับปรุงโปรแกรมที่มีความซับซ้อนให้เข้าใจได้ง่ายขึ้น รวมทั้งสามารถจัดการได้ง่ายขึ้น และเป็นการเตรียมรับมือกับการเปลี่ยนแปลงของการออกแบบได้ดีขึ้น อีกทั้งเป็นการลดความผิดพลาดของโปรแกรมที่เกิดจากความซับซ้อนของโครงสร้างโปรแกรม ทำให้เสียค่าใช้จ่ายในการบำรุงรักษาโปรแกรมน้อยลง ขั้นตอน นรีแฟคทอริ่งประกอบไปด้วย4 ขั้นตอนหลักได้แก่

1) การค้นหาและระบุตำแหน่งของร่องรอยไม่ดี

กระบวนการค้นหาเริ่มจากการระบุว่าปัญหาคอขวด (Bottleneck) ที่น่าจะเป็นจุดสังเกตุในการค้นหาตำแหน่งที่มาของร่องรอยไม่ดีที่เกิดขึ้น บางครั้งต้องอาศัยประสบการณ์ความเชี่ยวชาญของผู้พัฒนาในการตรวจสอบ เมท็อด แอตทริบิวต์ และคลาส ที่ทำงานในโปรแกรมเพื่อที่จะมุ่งแก้ไขได้ถูกจุดเพื่อให้ตรงตามวัตถุประสงค์ ทำให้เกิดประโยชน์สูงสุด

2) การวิเคราะห์สาเหตุที่เกิดพร้อมทั้งผลกระทบในการปรับปรุง

ก่อนที่จะทำ การแก้ไขปรับปรุง จำเป็นต้อง ตรวจสอบวิเคราะห์ ให้แน่ใจว่า ส่วนที่ทำการแก้ไขส่วนนั้นจะไม่ส่งผลกระทบต่อส่วนอื่นๆ ในภายหลัง จำเป็นต้อง ตรวจสอบ ทั้งก่อนแก้ไขและหลังแก้ไข โดยการทดสอบที่ดี มีขั้นตอนเป็นกระบวนการที่เรียกว่า Test-Driven Development ซึ่งเทคนิคที่ใช้ทดสอบและทำควบคู่ไปกับกระบวนการปรับปรุงโครงสร้าง ดังนั้นควรจะมีการออกแบบการทดสอบ เพื่อตรวจสอบผลหลังจากการปรับปรุงและเพื่อให้มั่นใจในวิธีการแก้ไขจะไม่ก่อให้เกิดความผิดพลาดของโปรแกรมเกิดขึ้น

3) ทำการปรับปรุงด้วยรีแฟคทอริ่งเทคนิค

เทคนิคในการปรับปรุงนั้นมีหลากหลายวิธี โดยแต่ละวิธีจะมีขั้นตอนง่ายหรือซับซ้อนแตกต่างกันไป แต่มีจุดมุ่งหมายเดียวกันคือทำการแก้ไขโค้ดตามวิธีการที่เลือกไว้ ในปัจจุบันสิ่งแวดล้อมสำหรับการพัฒนาแบบเบ็ดเสร็จ(Integrated Development Environment) หรือ โปรแกรมการแก้ไขโค้ด (Source Code Editor) โดยหลักการทั่วไปนั้นสนับสนุนเทคนิคการปรับปรุงโครงสร้างในตัว ซึ่งทำให้ปรับปรุงง่ายขึ้น แต่ปัจจุบันภาษาที่สนับสนุนยังมีไม่มาก เช่น จาวา(Java) ซีชาร์ป(C#) เป็นต้น

4) การทดสอบ

หลังจากการแก้ไขปรับปรุงแล้วจำเป็นต้องทำการ การทดสอบระบบทุกครั้งนั้น เพื่อให้เกิดความมั่นใจ ให้กับโปรแกรม และตรวจสอบข้อผิดพลาดหลังจากที่ได้ทำการการปรับปรุงไปแล้ว หากระบบนั้นมีขนาดใหญ่มาก มีโค้ดที่ซับซ้อนหลายบรรทัด ควรจะต้องมีการออกแบบและการทดสอบโปรแกรมให้สอดคล้องกับลักษณะโครงการ โดยกรณีนี้ควรมีการทดสอบด้วยระบบอัตโนมัติทุกๆ คืน (Nightly Build and Test)

เทคนิควิธีแก้ไขโดยนำแนวคิดมาจาก Kent Beck และ Martin Fowler [6] มาประกอบกัน สามารถแบ่งกลุ่มออกเป็น 6 วิธี ดังภาพที่ 2.2

ภาพที่ 2.2 วิธีการรีแฟคทอริ่ง [16]

กระบวนการรีแฟคทอริ่ง ประกอบด้วยวิธีการจัดการแก้ไขย่อยหลายวิธีแต่เมื่อนำมาจัดกลุ่มตามวิธีการแก้ไขสามารถแบ่งออกได้เป็น 6 ส่วนได้แก่

1) การย้ายคุณลักษณะระหว่างวัตถุ คือ เมื่อมีการออกแบบที่ไม่ดีจะทำให้เกิดความไม่สมดุลและนำพาปัญหาให้แต่ละวัตถุมีหน้าที่รับผิดชอบที่มากเกินไปหรือคุณลักษณะที่ไม่เหมาะสม วิธีการจัดการกับปัญหานี้คือ การเปลี่ยนโยกย้ายลักษณะจากวัตถุหนึ่งไปยังอีกวัตถุเพื่อทำให้เกิดความสมดุล ไม่ให้วัตถุหนึ่งมีหน้ารับผิดชอบมากจนเกินไป

2) การลดความยุ่งยากของ นิพจน์ที่มีเงื่อนไข คือ ภายใต้การทำงานของโปรแกรมหนึ่งจะประกอบด้วย เงื่อนไขการทำงานต่างๆ เพื่อให้ทำโปรแกรมทำงานตามวัตถุประสงค์ แต่บางเงื่อนไขถูกสร้างขึ้นอย่างไม่เหมาะสมทำให้มีความซับซ้อน ดังนั้น วิธีการนี้คือการจัดการลดความยุ่งยากของเงื่อนไขต่างๆ โดยทำการแยกย่อยเงื่อนไขออกเป็นส่วนเล็กเพื่อให้มีความง่ายต่อความเข้าใจ

3) การสร้างคุณสมบัติในการใช้งานร่วมกัน คือ การสืบทอดคุณสมบัติเป็นหลักการทั่วไปของการพัฒนาซอฟต์แวร์เชิงวัตถุ แต่บางครั้งการสืบทอดพฤติกรรมหรือการเติมคุณสมบัตินั้นทำได้

ยากเนื่องจากความลึกหรือลำดับของการสืบทอดเอง ดังนั้นวิธีการจัดการนี้ คือการย้ายลำดับของคลาสหรือการแยกพฤติกรรมการตอบสนองออกให้เป็นอิสระ

4) การ ประกอบการ ทำงานของเมท็อด คือ การทำงานแต่ในละเมท็อดอาจจะมีความคล้ายคลึงกัน หรือ แตกต่างกันบ้างแล้วแต่การทำงาน แต่เมื่อเมท็อดหรือคลาสมีขนาดใหญ่ขึ้นเป็นสาเหตุให้เกิดความซับซ้อนและยุ่งยากในการจัดการ ดังนั้นวิธีนี้จะทำการปรับแก้การทำงานเมท็อดให้มีขนาดเล็กและเหมาะสมต่อการใช้งาน

5) การจัดโครงสร้างข้อมูล คือ ข้อมูลจะถูกเก็บไว้ภายในวัตถุเพื่อใช้ในการติดต่อระหว่างกัน เพื่อทำการตอบสนองต่อการทำงานให้เป็นไปตามวัตถุประสงค์ แต่การจัดเก็บข้อมูลนั้นอาจจัดเก็บอย่างไม่เหมาะสม อาทิเช่น ชนิดของข้อมูลไม่เหมาะสม มีความซ้ำซ้อน รวมถึงปัญหาการปกปิดข้อมูล เป็นต้น ดังนั้นวิธีการนี้จะเป็นการจัดการกับข้อมูลภายในวัตถุ เพื่อให้เป็นตามวัตถุประสงค์ของการเขียนโปรแกรมเชิงวัตถุที่ต้องการให้ข้อมูลภายในวัตถุถูกปกปิดไว้และแก้ไขปัญหาความซ้ำซ้อนและไม่เหมาะสมของการเก็บข้อมูล

6) การทำให้เมท็อดเรียกใช้งานได้ง่าย คือ การพัฒนาโปรแกรมเชิงวัตถุที่ดีควรให้วัตถุสามารถเรียกใช้งานได้ และง่ายต่อการพัฒนา โดยวิธีการนี้ เป็นการแก้ไขเมท็อดต่างๆให้ง่ายต่อการพัฒนา เช่น การแก้ไขให้ชื่อของตัวแปรและเมท็อดต่างๆ ให้ง่ายต่อความเข้าใจ อีกทั้งการแก้ไขพารามิเตอร์ให้มีจำนวนเหมาะสมต่อการทำงานและการแก้ไขการดักจับข้อผิดพลาดของการทำงานให้เหมาะสม เป็นต้น

ในการแก้ไขปัญหาร่องรอยไม่ดีและการปรับปรุงคุณภาพโค้ด จำเป็นต้องเข้าใจถึงกระบวนการรีแฟคทอริ่ง เนื่องจากกระบวนการรีแฟคทอริ่งมีการระบุวิธีแก้ไขร่องรอยไม่ดีในแต่ละประเภทไว้ อย่างไรก็ดีวิธีการแก้ไขมีทั้งข้อดีและข้อเสีย งานวิทยานิพนธ์นี้จึงคัดกรองเลือกวิธีรีแฟคทอริ่งเพื่อแก้ไขร่องรอยไม่ดีและความคลุมเครือที่เหมาะสมในแต่ละรายการ เพื่อให้สามารถปรับปรุงโค้ดให้เป็นไปตามเกณฑ์ของคลีนโค้ดที่กำหนดไว้

2.1.6 **มาตรวัดซอฟต์แวร์** (Software Metrics)

มาตรวัดซอฟต์แวร์ถือได้ว่าเป็นเครื่องมือวัดสิ่งใดสิ่งหนึ่งของซอฟต์แวร์ โดยสามารถวัดได้ในเชิงปริมาณ ด้วยหลักการทางคณิตศาสตร์และวิทยาศาสตร์ [16] แนวคิดของ Fenton [17] กล่าวถึงหลักการวัดไว้ว่า การวัด (Measurement) นั้น เป็นกระบวนการที่ใช้ตัวเลขหรือสัญลักษณ์แทนค่าวัตถุหรือสิ่งต่างๆ โดยการสร้างเอนทิตี (Entity) และ แอตทริบิวต์ (Attribute) ขึ้นมาคำนวณหาค่าแสดงปริมาณหรือมูลค่าของวัตถุนั้นๆ แล้วทำการสรุปผลออกมาเป็นข้อมูล จากนั้น

นำไปจัดลำดับ แยกหมวดหมู่ หรือเปรียบเทียบ นำไปสู่กระบวนการควบคุมและพัฒนางานต่อไปได้ แต่การที่จะได้ผลลัพธ์ที่แม่นยำนั้นขึ้นอยู่กับตัวแปรและเครื่องมือที่นำมาใช้

หลักการออกแบบซอฟต์แวร์เชิงวัตถุจะแตกต่างจากวิธีการออกแบบโครงสร้างคือ วิธีการเชิงวัตถุจะพิจารณาถึงความสำคัญของข้อมูล (Data Attribute)และกระบวนการฟังก์ชันและเมท็อด โดยพยายามจัดรวมทั้งสองส่วนด้วยการห่อหุ้ม (Encapsulate) ไว้ในวัตถุ (Object) หนึ่งๆ ซึ่งวัตถุนั้นอาจเป็น สิ่งของ สถานที่ ระบบงาน ภาระงาน อุปกรณ์ ฯลฯ โดยวัตถุในระบบงานที่ออกแบบหรือพัฒนาซอฟต์แวร์นั้น อาจมีความสัมพันธ์กันในลักษณะใดลักษณะหนึ่ง เช่น ความซับซ้อนของโปรแกรม (Complexity) ของวิธีการเชิงวัตถุ และลักษณะการทำงานของโค้ดที่เป็นอิสระต่อกันในเชิงฟังก์ชัน (Functional Independence) เนื่องจากความซับซ้อนของซอฟต์แวร์ ทำให้สามารถใช้มาตรวัดคุณภาพได้อย่างเป็นอิสระ ดังนั้นขั้นตอนแรกของการวัดซอฟต์แวร์ จึงใช้แนวคิดและเทคนิคของ McCabe, Kamerer และ Halstead [3, 18, 19, 20, 21] ซึ่งเป็นเทคนิคพื้นฐานของการวัด ได้แก่

1) คัปปลิง (Coupling)

การวัดความสัมพันธ์ระหว่างโมดูลสองโมดูล [12,17] ว่า มีความซับซ้อนหรือระดับการขึ้นต่อกันระหว่างโมดูลมากน้อยเพียงใดนั้น สามารถวัดได้ 2 วิธีคือ แน่นหนา (Tight) และหลวม (Loose Coupling) แต่ Tight Coupling นั้นแสดงให้เห็นถึงศักยภาพและประสิทธิภาพสูง เมื่อค่าคัปปลิงเพิ่ม ทำให้เกิดข้อมูลแลกเปลี่ยนภายในส่วนโปรแกรม (Component) ที่มีขนาดใหญ่ขึ้น ซับซ้อนมากขึ้น โดยค่าคัปปลิงจะมีทิศทางตรงข้ามกับค่าโคฮีชัน Low Coupling จะมี High Cohesion จากมาตรวัดนี้ Larry Constantine ได้คิดค้นเพิ่มเติมว่า Low coupling เป็นสัญลักษณ์ของระบบโครงสร้างคอมพิวเตอร์ที่ดี มีการออกแบบดี แล้วเมื่อรวมกับ High Cohesion ถือว่า เป็นระบบที่สามารถอ่านเข้าใจได้ง่ายและสามารถบำรุงรักษาไว้ได้

2) โคฮีชัน (Cohesion)

การวัดระดับความสัมพันธ์ของกิจกรรมภายในโมดูลหนึ่งๆ [22] โดยแสดงการเกาะกันของหน้าที่หรือกิจกรรมภายในโมดูล ในการประมวลข้อมูลเพื่อให้ผลลัพธ์เป็นไปตามความต้องการ ซึ่งลักษณะโครงสร้างที่ดีนั้นต้องมีระดับการยึดเกาะกันของหน้าที่ในโมดูลสูง ซึ่งสามารถวัดค่าโคฮีชันแตกต่างจากมาตรวัดเชิงคุณภาพได้โดยการวิเคราะห์ประกอบกับหลักเกณฑ์แบบ Hermeneutics เพื่อตรวจสอบลักษณะของโค้ด โดยวัดค่าออกมาเป็นตัวเลข โดยทั่วไปของการวัดแบบโคฮีชันนี้ จะมีค่าที่ได้ ถูกแบ่งออกเป็น 2 แบบคือ High Cohesion กับ Low Cohesion โมดูลที่มีค่า High Cohesion นั้นแสดงถึงค่าความดีและแสดงให้เห็นถึงความสมบูรณ์ ความน่าเชื่อถือ

การนำมาใช้ใหม่ และเป็นที่ยอมรับ แต่ Low Cohesion จึงเป็นการแสดงผลในทางตรงกันข้ามคือ ไม่สามารถทดสอบ ไม่สามารถนำกลับมาใช้ และไม่สามารถเข้าใจยอมรับได้

3) โฮลสเตท (Halstead)

มาตรวัดโฮลสเตท [18] ใช้เพื่อแสดงลักษณะโค้ดและนับจำนวนของ ตัวดำเนินการ (Operator) และตัวถูกดำเนินการ (Operand) โดยนับจำนวนกลุ่มโค้ดโดยใช้มาตรวัดปริมาตร (Volume) ดังสมการที่ 2.1

$$V = N \times \log_2(n) \tag{2.1}$$

โดยที่ V คือ จำนวนปริมาณที่วัดข้อมูลในโปรแกรม โดยวัดค่าออกมาหน่วยความจำที่ถูกใช้งาน

ค่า N และ n ในสมการที่ 2.2 คำนวณได้จากสมการที่ 2.3

$$N = N1 + N2 \tag{2.2}$$

ค่า $N1$ คือ จำนวนของตัวดำเนินการที่ไม่ซ้ำกัน และ $N2$ คือ จำนวนของตัวถูกดำเนินการที่ไม่ซ้ำกัน

$$n = n1 + n2 \tag{2.3}$$

ค่า $n1$ คือ จำนวนของตัวดำเนินการ และ $n2$ คือ จำนวนของตัวถูกดำเนินการ

มาตรวัดซอฟต์แวร์ดังที่ได้กล่าวไว้ มีส่วนช่วยให้เห็นคุณลักษณะของซับเซตคลีนโค้ดในทางคณิตศาสตร์เพื่อ ให้ง่ายต่อการทดสอบและระบุค่าของโค้ดทั้งภายในและภายนอก ผลลัพธ์ที่ได้จากการระบุค่านี้จะนำไปจำแนก ในขั้นตอนของวิธีการต่อไป

2.1.7 **ฟัซซี่โลจิก** (Fuzzy Logic)

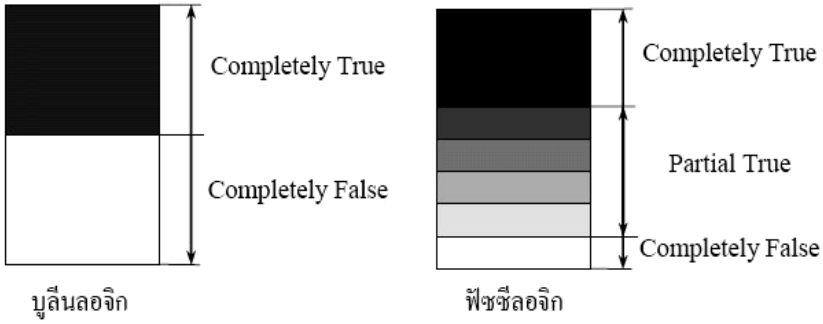
ตรรกศาสตร์คลุมเครือ หรือ ฟัซซี่ลอจิก (Fuzzy Logic) [23, 24, 25] พัฒนาจากทฤษฎีฟัซซี่เซต (Fuzzy Set) โดยเป็นการใช้เหตุผลแบบประมาณ ซึ่งแตกต่างจากการใช้เหตุผลแบบ

เด็ดขาดในลักษณะ ถูก หรือ ผิด ใช่ หรือ ไม่ใช่ ของตรรกศาสตร์แบบฉบับ (Classical Logic) ตรรกศาสตร์คลุมเครือนั้นสามารถถือเป็นการประยุกต์ใช้งาน ฟัซซี่เซต เพื่อจำลองการตัดสินใจของผู้เชี่ยวชาญต่อปัญหาที่ซับซ้อนค่าระดับความจริง ในตรรก ศาสตร์คลุมเครือนั้น มักจะสับสนกับค่าความน่าจะเป็น ซึ่งมีแนวความคิดที่แตกต่างกัน ค่าระดับความจริงคลุมเครือใช้ในการระบุค่าความเป็นสมาชิกของ เซต แต่ค่าความน่าจะเป็นระบุความเป็นไปได้ของสภาพการณ์แต่ละรูปแบบ ที่อาจจะเกิดขึ้น

ตรรกศาสตร์คลุมเครือสามารถระบุค่าความเป็นสมาชิกของเซต (Set Membership Values) ด้วยค่าระหว่าง 0 และ 1 ทำให้เกิดระดับกึ่งในลักษณะของ สีเทา นอกจาก ขาว และดำ ซึ่งมีประโยชน์ในการจำลองระดับซึ่งสามารถระบุด้วยคำพูด "เล็กน้อย" "ค่อนข้าง" "มาก" โดยใช้ค่าความเป็นสมาชิกของเซตบางส่วน ตรรกศาสตร์คลุมเครือมีความสัมพันธ์กับฟัซซี่เซตและทฤษฎีความเป็นไปได้ (Possibility Theory) ซึ่งคิดค้นขึ้นในปี ค.ศ.1965 โดยศาสตราจารย์ลอตฟี ซาเดห์ แห่งมหาวิทยาลัยแห่งรัฐแคลิฟอร์เนีย เบิร์กลีย์

1) พื้นฐานแนวคิดแบบฟัซซี่

ตรรกะแบบฟัซซี่[23, 24, 25] เป็นเครื่องมือที่ช่วยในการตัดสินใจภายในใต้ความไม่แน่นอนของข้อมูลโดยยอมให้มีความยืดหยุ่นได้ โดยใช้หลักเหตุผลที่คล้ายการเลียนแบบวิธีความคิดที่ซับซ้อนของมนุษย์ ฟัซซี่ลอจิกมีลักษณะที่พิเศษกว่าตรรกะแบบ บูลีน (Boolean Logic) เป็นแนวคิดที่มีการต่อขยายในส่วนของความจริง(Partial True) โดยค่าความจริงจะอยู่ในช่วงระหว่างจริง (Completely True) กับเท็จ (Completely False) ส่วนตรรกศาสตร์เดิมจะมีค่าเป็นจริงกับเท็จเท่านั้น แสดงดังภาพที่ 2.3



ภาพที่ 2.3 ตรรกะแบบบูลีนกับตรรกะแบบฟัซซี่ [23]

ความเป็นฟัซซี่ (Fuzziness) มีชื่อเรียกว่า มัลติวาลานซ์ (Multivalance) ซึ่งมีค่าที่ความเป็นสมาชิกมากกว่า 2 ค่า และแตกต่างกับไบวาลานซ์ (Bivalance) ที่มีความเป็นสมาชิกเพียง2 ค่า ฟัซซี่เซต (Fuzzy Set) [26] เป็นเครื่องมือทางคณิตศาสตร์ที่สื่อถึงความไม่แน่นอน (Uncertainty)

ภาพที่ 2.4 ความไม่แน่นอน (Uncertainty) และความคลุมเครือ (Ambiguity) [23]

จากภาพที่ 2.4 เป็นการแสดงให้เห็นว่าแนวทางในการตัดสินใจของปัญหาทั้งหมดมีเพียงส่วนน้อยที่เป็นสิ่งที่แน่นอน (Certainty) ที่เหลือคือสิ่งที่ไม่แน่นอนซึ่งประกอบด้วยความไม่แน่นอนที่มีลักษณะแบบสุ่ม และความไม่แน่นอนที่มีลักษณะเป็นฟัซซี่ หรือความคลุมเครือ ซึ่งมีมากกว่าร้อยละ 40 เพราะปัญหาส่วนมากเกี่ยวข้องกับการตัดสินใจของมนุษย์ซึ่งจะตัดสินใจตามพื้นฐานความคิดของตนเป็นหลัก

แนวคิดฟัซซี่จะสร้างวิธีทางคณิตศาสตร์ที่แสดงถึงความคลุมเครือความไม่แน่นอนของระบบที่เกี่ยวข้องกับความคิด ความรู้สึกของมนุษย์ เมื่อพิจารณาส่วนประกอบต่าง ๆ ในความไม่แน่นอนเพื่อกำหนดเงื่อนไขในการตัดสินใจ (Decision Making) โดยอาศัยเซตของความเป็นสมาชิก (Set Membership)

2) ฟัซซี่เซต (Fuzzy set)

ข้อสรุปของฟัซซี่เซต ดร.พยุง มีสัจ [23] ได้รวบรวมรายอย่างละเอียดไว้ดังนี้ ฟัซซี่เซต (Fuzzy Set) เป็นเซตที่มีขอบเขตที่ราบเรียบ ทฤษฎีฟัซซี่เซตจะครอบคลุมทฤษฎีเซตแบบฉบับ โดยฟัซซี่เซตยอมให้มีค่าความเป็นสมาชิกของเซตระหว่าง 0 และ 1 ในโลกแห่งความเป็นจริงเซตไม่ใช่มีเฉพาะเซตแบบฉบับเท่านั้น จะมีเซตแบบฟัซซี่ด้วย ฟัซซี่เซต จะมีขอบเขตแบบฟัซซี่ไม่ใช่เปลี่ยนแปลงทันทีทันใดจากขาวเป็นดำ

(1) เซตแบบฉบับ

ในเซตแบบฉบับ (Classical Set) หรือเซตทวินัย (Crisp Set) เป็นเซตที่มีค่าความเป็นสมาชิกเป็น 0 หรือ 1 {0, 1} เท่านั้น เซตในทฤษฎีเซตแบบฉบับจะมีขอบเขตแบบแข็ง (Sharp

Boundary) ซึ่งเป็นขอบเขตที่ตัดขาดจากกันแบบทันทีทันใด เซตแบบฉบับมีการกำหนดค่าความเป็นสมาชิกตามแนวคิดเลขฐานสอง โดยที่ตัวแปรหนึ่ง ๆ จะมีค่าความเป็นสมาชิกเพียงสองค่า คือ 0 ไม่เป็นสมาชิก และ 1 เป็นสมาชิก

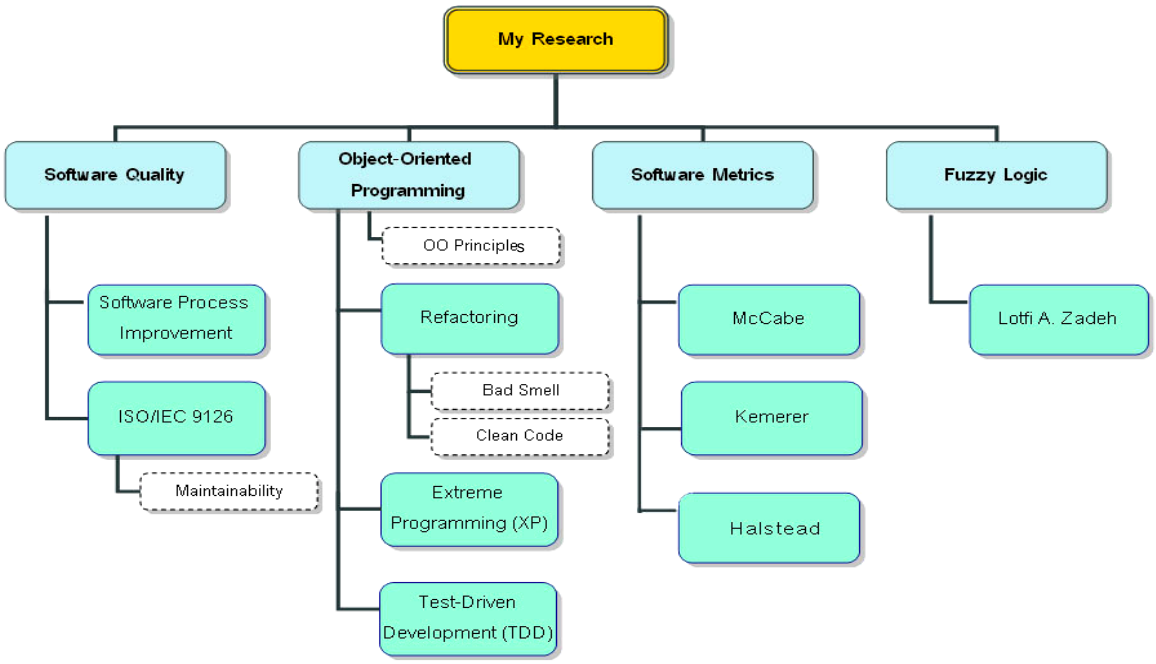
(2) นิยามของฟัซซี่เซต

กำหนดให้ *X* เป็นเซตที่ไม่ว่าง ฟัซซี่เซต *A* สามารถแสดงลักษณะเฉพาะได้จากฟังก์ชันความเป็นสมาชิก $\mu_{\underline{A}}(x) : X \rightarrow [0,1]$ เมื่อสามารถตีความเป็นค่าของความเป็นสมาชิกภาพของตัวประกอบ *x* ในฟัซซี่เซต $\underline{A}$ สำหรับแต่ละ (อ่านว่า "*x* เป็นสมาชิกของ *X*") ฟัซซี่เซตสามารถเขียนเป็นเซตของคู่ลำดับ (Tuples) ทฤษฎีฟัซซี่เซตสามารถแก้ปัญญาข้อจำกัดของเซตแบบดั้งเดิมได้ โดยฟัซซี่เซตยอมให้มีค่าหรือดีกรีของความเป็นสมาชิก (Degree of Membership) ซึ่งแสดงด้วยค่าตัวเลขระหว่าง 0 และ 1 หรือเขียนเป็นสัญลักษณ์ [0, 1] โดย 0 หมายถึง ไม่เป็นสมาชิกในเซต 1 หมายถึง เป็นสมาชิกในเซต และค่าระหว่าง 0 กับ 1 เป็นสมาชิกบางส่วนในเซต การทำเช่นนี้ทำให้เกิดความราบเรียบในการเปลี่ยนจากพื้นที่นอกเซตไปอยู่ในเซตของสมาชิกต่างๆ โดยมีฟังก์ชันสมาชิก (Membership Function) เป็นฟังก์ชันจัดเทียบ (Mapping Function) วัตถุในโดเมนใดๆ ให้เป็นค่าความเป็นสมาชิกในฟัซซี่เซต

การใช้ความเชี่ยวชาญและค่าความเป็นสมาชิกนั้นถือเป็นข้อได้เปรียบที่เป็นประโยชน์ต่อการจำแนกความคลุมเครือที่มีอยู่ในโค้ด ดังนั้นจึงนำเทคนิคฟัซซี่เซตเข้ามาประกอบเป็นขั้นตอนหนึ่งในวิธีการทดสอบของวิทยานิพนธ์นี้

2.2 งานวิจัยที่เกี่ยวข้อง

งานวิจัยที่เกี่ยวข้องกับงานวิทยานิพนธ์นี้ประกอบด้วย 4 ส่วนหลัก คือ คุณภาพซอฟต์แวร์ (Software Quality) การออกแบบและเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming) มาตรวัดซอฟต์แวร์และฟัซซี่โลจิกโดยแสดงความสัมพันธ์ดัง ภาพที่ 2.5



ภาพที่ 2.5 แผนภาพงานวิจัยที่เกี่ยวข้อง (Related Work Diagram)

2.2.1 งานวิจัยที่เกี่ยวข้องกับซอฟต์แวร์คุณภาพและความสามารถในการบำรุงรักษา

นอกจากหลักการสำหรับเขียนโปรแกรมเพื่อให้ได้โค้ดที่ดีและมีคุณภาพ มาตรฐานมีส่วนกำหนดในเรื่องการประเมินคุณภาพของซอฟต์แวร์ก่อนส่งมอบให้กับผู้ใช้งาน งานวิทยานิพนธ์นี้สนใจเรื่อง ความสามารถในการบำรุงรักษา (Maintainability) ที่มีความเชื่อมโยงกับโค้ด เนื่องจากโค้ดเป็นองค์ประกอบภายในสำคัญของซอฟต์แวร์ที่นักพัฒนาควรให้ความใส่ใจ ดังเช่น งานวิจัยของ Don Coleman Paul Oman และ Jack Hagermeister [27,28] ได้สร้างวิธีการเพื่อหาค่าความสามารถในการบำรุงรักษาด้วยมาตรวัดซอฟต์แวร์เพื่อประเมินคุณภาพ โดยพิจารณาซอฟต์แวร์ออกเป็นลำดับชั้น (Hierarchy) ได้แก่ ด้านการจัดการ (Management) ด้านปฏิบัติการ (Operation Environment) และองค์ประกอบโดยรวมของระบบ (Target Software System) ผลจากการสร้างวิธีการนี้สามารถคาดการณ์และวัดค่าความสามารถในการบำรุงรักษาได้สำเร็จ นอกจากนี้ Paul ได้ร่วมงานวิจัย กับ Troy Pearse [29] เพื่อศึกษาความสามารถในการบำรุงรักษาของโค้ด ที่ผ่านวิธีการและกิจกรรมต่างๆ โดยมีการประเมินค่าความเปลี่ยนแปลงทั้งก่อนและหลังกิจกรรม งานวิจัยทั้งสองฉบับนี้ชี้ให้เห็นถึงการเชื่อมโยงของโค้ดที่มีส่งผลถึงค่าความสามารถในการบำรุงรักษา โดยนำนิยามของ Software Engineering Institute: SEI [11] ได้กล่าวไว้ว่า

“ความพยายามแก้ไของค์ประกอบให้เป็นไปตามข้อกำหนดหรือตามความต้องการเพื่อให้ดำเนินงานได้ตามวัตถุประสงค์” โดยวิทยานิพนธ์นี้ได้นำนิยามและดัชนีวัดค่าความสามารถในการบำรุงรักษาตามมาตรฐาน SEI เพื่อประเมินคุณภาพและความสามารถในการบำรุงรักษาให้กับโค้ดที่ผ่านขั้นตอนการจำแนกและปรับปรุงคุณภาพตามเกณฑ์ของซับเซตคลีนโค้ด โดยใช้สมการ Maintainability Index [11, 28] คือ

$$MI = MIwoc - MIwc \tag{2.4}$$

MI หมายถึง ค่าดัชนีความสามารถในการบำรุงรักษาที่สามารถคำนวณจากทุกฟังก์ชัน คลาสและโครงสร้างในทุกไฟล์โดยที่ ***MIwoc*** คำนวณจากสมการที่ 2.5 และ ***MIwc*** คำนวณจากสมการที่ 2.6 มีรายละเอียดดังต่อไปนี้

$$MIwoc = 171 - 5.2 * \ln(aveV) - 0.23 * aveG - 16.2 * \ln(aveLOC) \tag{2.5}$$

MIwoc หมายถึง ค่าดัชนีความสามารถในการบำรุงรักษาที่ไม่นับรวมกับคำอธิบายภายในโปรแกรม โดยที่ aveV คือ ค่าเฉลี่ยของ Halstead Volume ***aveG*** คือ ค่าเฉลี่ยของ Cyclomatic Complexity และ ***aveLOC*** คือค่าเฉลี่ยของจำนวนบรรทัดของโค้ดภายในโปรแกรมที่ทดสอบ

$$MIwc = 50 * \sin(\sqrt{2.4 * perCM}) \tag{2.6}$$

MIwc หมายถึง ค่าความสามารถในการบำรุงรักษาที่มีการนับรวมคำอธิบายภายในโปรแกรม โดยที่

perCM คือ อัตราร้อยละของจำนวนคำอธิบายภายในโปรแกรมที่ใช้ในการทดสอบ

ผลลัพธ์ที่ได้จากการประเมินด้วยสมการนี้ จะนำไปประเมินจากดัชนีชี้วัดค่าความสามารถในการบำรุงรักษาได้ตามตารางที่2.2

ตารางที่ 2.2 ค่าดัชนีชี้วัดความสามารถในการบำรุงรักษา [11, 28]

ค่าดัชนี	**รายละเอียด**
มากกว่า 85	ระดับสูง - มีความสามารถในการบำรุงรักษาที่ดี

ตารางที่ 2.2 ค่าดัชนีชี้วัดความสามารถในการบำรุงรักษา [11, 28] (ต่อ)

ค่าดัชนี	รายละเอียด
65 – 85	ระดับปานกลาง - ยากต่อการจัดการบำรุงรักษา
น้อยกว่า 65	ระดับต่ำ - ไม่สามารถจัดการบำรุงรักษาระบบหรือซอฟต์แวร์ได้

จากคุณสมบัติด้านความสามารถในการบำรุงรักษาส่งผลให้ผู้พัฒนาซอฟต์แวร์จำเป็นต้องใส่ใจในทุกๆ รายละเอียด ทุกๆ ขั้นตอนของกระบวนการพัฒนาซอฟต์แวร์ โดยเฉพาะอย่างยิ่งในส่วนของขั้นตอนการออกแบบและการเขียนโปรแกรมที่มีความสำคัญไม่น้อยไปกว่าขั้นตอนอื่นๆ ซึ่งจะถูกนำไปประเมินซับเซตคลีนโค้ดว่า มีคุณสมบัติความสามารถด้านการบำรุงรักษาหรือไม่

วิทยานิพนธ์นี้ศึกษาความสามารถในการบำรุงรักษาเพื่อเป็นเกณฑ์ชี้วัดคุณภาพความเป็นซับเซตคลีนโค้ดและนำสมการของ Maintainability Index มาประเมินค่าความสามารถในการบำรุงรักษาของซับเซตคลีนโค้ดที่ผ่านขั้นตอนการจำแนกและระบุ ค่าที่ได้รับจากการวัดนี้จะแสดงให้เห็นถึงคุณสมบัติของซอฟต์แวร์คุณภาพตามมาตรฐาน

2.2.2 งานวิจัยที่เกี่ยวข้องกับหลักการออกแบบและเขียนโปรแกรมเชิงวัตถุ

การพัฒนาซอฟต์แวร์โดยยึดหลักการออกแบบและเขียนโปรแกรมเชิงวัตถุนั้น เริ่มต้นพัฒนาในปี 2493 โดย Alan Kay [12, 13, 14] ซึ่งเป็นหนึ่งในผู้ผลักดันให้เกิดการขยายตัวของอุตสาหกรรมซอฟต์แวร์ที่มีการแข่งขันกันอย่างสูงในด้านเทคนิคการตอบสนองความต้องการของผู้ใช้งาน จึงเกิดความคิดริเริ่มในผู้พัฒนาซอฟต์แวร์ทำการพัฒนาปรับปรุงเทคนิคการออกแบบและการเขียนโปรแกรมเชิงวัตถุอย่างต่อเนื่อง เป็นผลให้ลดทอนเวลาในการผลิตพร้อมทั้งลดข้อผิดพลาดที่เคยเกิดขึ้น อย่างไรตามจนถึงวันนี้ประเด็นเรื่อง ความผิดพลาดก็ยังคงเกิดขึ้นและยังเป็นปัญหาที่ต้องการการแก้ไข เมื่อย้อนมองดูรายละเอียดของข้อผิดพลาดพบว่า สาเหตุส่วนหนึ่งเกิดจากการเขียนโค้ดที่ขาดคุณภาพ เป็นเหตุให้ผู้เชี่ยวชาญด้านการออกแบบและการเขียนโปรแกรม Martin F. Fowler และคณะ [6] ได้รวบรวมข้อผิดพลาดของโค้ดหรือร่องรอยไม่ดี มาสร้างแนวทางแก้ไขซึ่งเรียกว่า “เทคนิคการทำรีแฟคทอริ่ง(Refactoring)” โดยเทคนิคการทำรีแฟคทอริ่งถือเป็นเทคนิคที่ผู้พัฒนาซอฟต์แวร์ได้ให้ความสนใจและทำการวิจัยปรับปรุงเทคนิคนี้ให้มีประสิทธิภาพมากยิ่งขึ้น อาทิ เช่น Mika Mäntylä [8] ที่ทำการจัดแยกประเภทของสาเหตุความผิดพลาดที่เกิดจากร่องรอยไม่ดีเพื่อสร้างแนวทางและวิธีการแก้ไข ซึ่งโค้ดที่ผ่านกระบวนการทดลองของ Mäntylä แสดงให้

เห็นว่า โค้ดเหล่านั้นสามารถบำรุงรักษาได้ง่ายขึ้น ส่วนงานวิจัยของ Tom Mens และ Tourwé [30] เป็นการวิเคราะห์และการปรับกระบวนการทำรีแฟคทอริ่งให้ขั้นตอนชัดเจนมากยิ่งขึ้น ถึงแม้ว่ากระบวนการทำรีแฟคทอริ่งจะช่วยลดข้อผิดพลาดและส่งผลให้โค้ดมีคุณภาพหรือไม่ก็ตาม แต่งานของ Thomas Ruhroth และคณะ [31] ได้สร้างแบบจำลองยูเอ็มแอล (UML) เพื่อทดสอบคุณภาพของโค้ดที่ได้หลังจากผ่านกระบวนการรีแฟคทอริ่ง เป็นผลลัพธ์ที่ได้ทำให้โครงสร้างของโค้ด (Well-Structuredness) อีกทั้งโค้ดถูกเรียบเรียงให้อ่านเข้าใจง่ายขึ้น (Understandability) และโค้ดบางส่วนสามารถนำกลับมาใช้งานได้อีกครั้ง (Reusability) เป็นเหตุให้เทคนิคการทำรีแฟคทอริ่งเป็นเทคนิคที่สามารถช่วยลดข้อผิดพลาดจากการเขียนโค้ดได้ในระดับหนึ่ง

อย่างไรก็ดีเทคนิคนี้ยังไม่สามารถครอบคลุมถึงคุณสมบัติของโค้ดที่ดี ตามที่ระบุไว้ในนิยามของซอฟต์แวร์คุณภาพ (Software Quality) [32] โดยโค้ดที่ดีนั้นจำเป็นต้องประกอบไปด้วย

1) อ่านเข้าใจง่าย (Readability)

2) ง่ายต่อการบำรุงรักษา (Maintainability), การทดสอบ (Testing), การแก้จุดบกพร่อง (Debug), การแก้ไขเพิ่มเติม (Modification) และมีความยืดหยุ่น (Portability)

3) ความซับซ้อนต่ำ (Low complexity)

4) ใช้ทรัพยากรเพื่อสนับสนุนการทำงานของโค้ดต่ำ (Low resource consumption)

5) มีความทนทาน (Robustness)

อย่างไรก็ตามการจะออกแบบและเขียนโปรแกรมให้ได้โค้ดที่มีคุณภาพนั้น จำเป็นต้องเข้าใจหลักการออกแบบและการเขียนโปรแกรม อีกทั้งต้องเข้าใจลักษณะของโค้ดที่มีคุณภาพให้มากขึ้น ดังที่ Robert C. Martin [9] ได้ทำการรวบรวมประเด็นหลักทางด้านการออกแบบและเขียนโปรแกรม โดยเสนอกิจกรรมที่จะนำไปสู่ร่องรอยไม่ดี และคุณสมบัติของโค้ดที่มีคุณภาพไว้ในหนังสือ Clean Code: A Handbook of Agile Software Craftsmanship โดยแนวคิดของ Robert ถือเป็นวิธีการที่ช่วยลดจำนวนร่องรอยไม่ดีให้น้อยลงได้ งานวิทยานิพนธ์นี้จึงนำจุดเด่นมาวิเคราะห์ คิดค้นและสร้างวิธีการเพื่อจำแนก

วิทยานิพนธ์นี้มุ่งเน้นแนวทางการปรับปรุงโค้ดที่มีร่องรอยไม่ดีให้กลับเป็นโค้ดที่มีคุณลักษณะที่ดีตามหลักการออกแบบและการเขียนโปรแกรมเชิงวัตถุ ทั้งนี้การแก้ไขปัญหาร่องรอยไม่ดีจะยึดตามแนวทางที่ได้ศึกษาไว้โดยเลือกวิธีการรีแฟคทอริ่งให้เหมาะสมกับร่องรอยไม่ดีนั้น เพื่อให้โค้ดผ่านกระบวนการปรับปรุงมีคุณลักษณะที่ดีตามเกณฑ์ และไม่มีร่องรอยไม่ดีปรากฏให้เห็น

2.2.3 งานวิจัยที่เกี่ยวข้องกับมาตรวัดซอฟต์แวร์

ดังที่ Tom DeMarco [16] ผู้เชี่ยวชาญทางด้านมาตรวัดซอฟต์แวร์ได้กล่าวไว้ว่า "You cannot control what you cannot measure" ฉะนั้นสิ่งที่จะทำให้เครื่องมือจำแนกและระบุโค้ดสำเร็จได้นั้น คือการนำแนวคิดด้านมาตรวัดซอฟต์แวร์เข้ามาสนับสนุนการทดลอง เนื่องด้วยเหตุดังต่อไปนี้

1) มาตรวัดซอฟต์แวร์เป็นการนำเอาหลักคณิตศาสตร์เข้ามาช่วยในด้านการวัดค่า (Measurement) เพื่อทำการจำแนกและการระบุค่า ทั้งนี้จากหลักทั่วไปที่ Chris F. Kemerer และคณะ [19, 20] ได้นำเสนอเทคนิคคัปปลิง (Coupling) และโคฮีชัน (Cohesion) เพื่อทำการจำแนกและแยกแยะโค้ด

2) มาตรวัดซอฟต์แวร์สามารถทำให้ส่วนประกอบของซอฟต์แวร์ ที่ประกอบด้วยวัตถุต่างๆ ที่แสดงอยู่ในรูปของนามธรรมสามารถแสดงอยู่ในรูปธรรมได้อย่างชัดเจนขึ้น [33]

3) มาตรวัดซอฟต์แวร์เป็นเครื่องมือพิสูจน์และการประเมินผลการทดลองที่เกิดจากเครื่องมือจำแนกและระบุคุณภาพของโค้ดว่าถูกต้อง พร้อมทั้งมีประสิทธิภาพหรือไม่

มาตรวัดซอฟต์แวร์ที่ใช้ในงานวิทยานิพนธ์นี้ ประกอบด้วย

1) Afferent Coupling at Method Level (ACML) คือ การวัดการพึ่งพาการทำงานของเมท็อดที่พิจารณาภายในคลาส

2) Afferent Coupling at Public Field Level (AC-NPF) คือ การวัดการพึ่งพาการทำงานของแอตทริบิวต์ที่สามารถเรียกใช้งานได้จากภายนอก

3) Coupling between Object Classes (CBO) คือ การวัดความสัมพันธ์ในการทำงานของวัตถุว่าจำเป็นต้องพึ่งพาการเรียกใช้งานแอตทริบิวต์หรือเมท็อดของวัตถุอื่นในการทำงาน

4) Cyclomatic Complexity (CC) คือ การนับจำนวนของเส้นทางทั้งหมดที่สามารถเกิดได้ในการทำงานภายในฟังก์ชันหรือเมท็อดที่พิจารณา

5) Depth of Inheritance Tree (DIT) คือ การนับจำนวนของระดับชั้นของการสืบทอดคุณสมบัติจากคลาสที่พิจารณา

6) Distance from the Main Sequence (DMS) คือ การวัดระยะห่างของการทำงานที่เกิดขึ้นเทียบกับส่วนการทำงานหลักของโปรแกรม

7) Efferent Coupling at Method Level (ECML) คือ การวัดการเรียกใช้งานของเมท็อดที่พิจารณาภายในคลาส

8) Inline of Cyclomatic Complexity (ILCC) คือ การนับจำนวนเส้นทางทั้งหมดที่สามารถเป็นไปได้ของชุดคำสั่งในการทำงานภายในฟังก์ชันหรือเมท็อดที่พิจารณา

9) Inline of Nesting Depth (ILND) คือ การนับระดับความลึกของชุดคำสั่งที่เรียกซ้อนกันภายในเมท็อดที่ถูกพิจารณา

10) Is a Structure (ISS) คือ การวัดลักษณะของการทำงานของชุดคำสั่งเป็นรูปแบบโครงสร้างในการทำงานของฟังก์ชันที่พิจารณา

11) Lack of Cohesion Of Methods (LCOM1) คือ การวัดระดับความสัมพันธ์ของการทำงานร่วมระหว่างเมท็อดหรือตัวแปรภายในคลาส โดยมีค่าระหว่าง 0-1

12) Lack of Cohesion Of Methods for Henderson-Sellers (LCOM3) คือ การวัดระดับความสัมพันธ์ของการทำงานร่วมระหว่างเมท็อดหรือตัวแปรภายในคลาส เพื่อป้องกันปัญหาการหาค่าโคฮีชันผิดพลาดจากสูตรแรก โดยมีค่าระหว่าง 0-2

13) Lazy between Method and Field in Class level (LMFC) คือ การวัดปริมาณของเมท็อดและแอตทริบิวต์ภายในคลาสที่ประกาศให้ถูกเรียกใช้งาน

14) Number of Fields (NFD) คือ การนับจำนวนของแอตทริบิวต์ที่ถูกประกาศไว้ในคลาสที่ถูกพิจารณา

15) Number Inline of Instructions (NILI) คือ การนับจำนวนของชุดคำสั่งที่ถูกเรียกใช้งานในฟังก์ชันหรือเมท็อดที่ถูกพิจารณา

16) Number Lines of Code (NLOC) คือ การนับจำนวนของบรรทัดของโค้ดที่ประกาศไว้ในการทำงานของฟังก์ชันหรือเมท็อดที่ถูกพิจารณา

17) Number of Children (NOC) คือ การนับจำนวนคลาสลูกทั้งหมดที่ทำการสืบทอดคุณสมบัติจากคลาสที่ถูกพิจารณา

18) Number of Interfaces (NOI) คือ การนับจำนวน ส่วนต่อประสาน ที่ถูกประกาศในคลาสที่ถูกพิจารณา

19) Number of Class not Inheritent much be Seal or Static (NOIM) คือ การนับจำนวนของคลาสที่ไม่ได้ถูกนำไปใช้ในการสืบทอดคุณสมบัติต้องมีการปกปิดเมท็อดหรือข้อมูลไว้

20) Number of Methods (NOM) คือ การนับจำนวนของเมท็อดที่ถูกประกาศไว้เพื่อเรียกใช้งานภายในหรือภายนอกคลาสที่ใช้พิจารณา

21) Number of Overloads (NOO) คือ การนับจำนวนของเมท็อดที่ถูกโอเวอร์โหลดในคลาสที่ถูกพิจารณา

22) **Number of Parameters (NOP)** คือ การนับจำนวนของพารามิเตอร์ที่ถูกส่งผ่านมาในฟังก์ชันหรือเมท็อดในการทำงานไม่ว่าจะเป็นพารามิเตอร์ที่ถูกอ้างถึงหรือส่งออกไป

23) **Number of Variables (NOV)** คือ การนับจำนวนของตัวแปรที่ถูกประกาศไว้ในฟังก์ชันหรือเมท็อดที่ถูกพิจารณา

24) **Number of Public Fields (NPF)** คือ การนับจำนวนของแอตทริบิวต์ที่ถูกประกาศไว้ในคลาสที่สามารถเรียกใช้งานได้จากภายนอก

25) **Percentage of Comment (PCC)** คือ การวัดอัตราร้อยละของคำอธิบายเพื่อบรรยายการทำงานภายในคลาสที่ถูกพิจารณา

26) **Relational Cohesion (RC)** คือ การวัดความสัมพันธ์การทำงานร่วมกันกับข้อมูลภายในฟังก์ชันหรือเมท็อดที่ถูกพิจารณา

27) **Strange of Afferent Coupling-Outer method and field level (SAC-OMF)** คือ การวัดความผิดปกติของการพึ่งพาการทำงานจากการเรียกใช้งานเมท็อดและแอตทริบิวต์จากภายนอก

28) **Strange of Efferent Coupling-Inner method level (SEC-IM)** คือ การวัดความผิดปกติของการเรียกใช้งานเมท็อดที่อยู่ภายในคลาส

มาตรวัดที่คัดเลือกมาใช้ในวิทยานิพนธ์นี้เพื่อใช้ในการอธิบายร่องรอยไม่ดีและคุณลักษณะความเป็นซับเซตคลีนโค้ดที่ถูกเลือกใช้ในการทดลอง ทั้งนี้นอกจากนำข้อมูลการวัดที่ได้จากชุดโค้ดสำหรับทดสอบจะถูกไปประมวลผลให้กับกฎการจำแนกที่ใช้สำหรับการจำแนกร่องรอยไม่ดี ความคลุมเครือ และซับเซตคลีนโค้ด ออกจากกัน มาตรวัดยังถูกนำไปใช้เป็นเกณฑ์ประกอบค่าการชี้วัดด้านการบำรุงรักษาอีกด้วย ทั้งนี้ค่าพิจารณา (Specification of Outlier) ที่ใช้จำแนกคลีนโค้ด ศึกษาและวิเคราะห์จาก Robert C.Martin [9] โดยมีรายละเอียดดังตารางที่ 2.3 ซึ่งในตารางเกณฑ์การพิจารณาประกอบด้วย ลำดับที่เพื่อแสดงถึงส่วนย่อยของ รายการซับเซตคลีนโค้ด ที่อยู่ในกลุ่มฟังก์ชันและคลาส พร้อมรายการซับเซตคลีนโค้ดพร้อมมาตรวัดที่ใช้ ส่วนเกณฑ์การพิจารณาร่องรอยไม่ดีนั้น ศึกษาและวิเคราะห์จาก Mika Mäntylä [8] โดยมีรายละเอียดดังตารางที่ 2.4

ตารางที่ 2.3 เกณฑ์การพิจารณาซับเซตคลีนโค้ด

ลำดับที่	รายการ	มาตรวัดซอฟต์แวร์	เกณฑ์พิจารณา
1	Good Comment	PCC	x>=20
2.1	Small Functions	NLOC	x<=22
		NILI	x<=50
		CC	x<=10
		ILCC	x<=20
2.2	Do One Thing	RC	1.5<=x<=3.5
2.3	Nesting Depth	ILND	x<=4
2.4	Function Arguments	NOP	x<=5
2.5	Structured Programming	ISS	x=1
3.1	Class Organization	DIT	x<6
		NOC	x<6
		NOI	x<20
		NOIM	x<1
3.2	Encapsulation	NPF	x<1
3.3	Classes Should Be Small	NOP	x<=5
		NOV	x<=8
		NOO	x<=6
		NOM	x<=15
		NFD	x<=10
3.4	Maintaining Cohesion	LCOM1	x<=0.5
		LCOM3	x<=0.8
3.5	Organizing for Change	ACML	x>0
		ECML	x<=50
3.6	The Law of Demeter	DMS	x<=0.5
3.7	Clean Boundary	CBO	x<=6

ตารางที่ 2.4 เกณฑ์การพิจารณาโค้ดร่องรอยไม่ดี

ลำดับที่	รายการ	มาตรวัดซอฟต์แวร์	เกณฑ์พิจารณา
1	Long Method	NLOC	x>=60
		NILI	x>=200
		CC	x>=30
		ILCC	x>=60
2	Large Class	NOM	x>20
		NFD	x>20
		LCOM1	x>0.8
		LCOM3	x>1.0
3	Long Parameter List	NOP	x>7
4	Temporary Field	AC-NPF	x<=3
5	Feature Envy	SAC-OMF	x>0
6	Inappropriate Intimacy	SEC-IM	x>5
7	Lazy Class	LMFC	x>0
8	Bad Comment	PCC	X=0

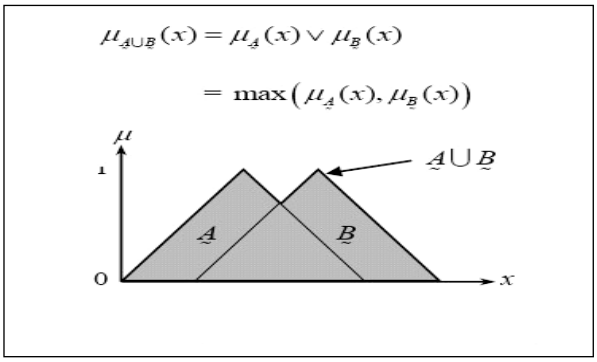
2.2.4 งานวิจัยที่เกี่ยวข้องกับฟัซซี่โลจิก

หลังจากการใช้มาตรวัดทางซอฟต์แวร์ ผลลัพธ์ที่เกิดขึ้นบางส่วนสามารถจำแนกและระบุคุณสมบัติของโค้ดได้ แต่ยังคงพบว่ามีผลลัพธ์บางส่วนเกิดความคลุมเครือขึ้น เนื่องจากความซับซ้อนของโค้ด ทำให้มาตรวัดซอฟต์แวร์ไม่สามารถวัดผลได้อย่างเต็มศักยภาพ ดังนั้นงานวิทยานิพนธ์นี้จึงได้นำเทคนิคด้านปัญญาประดิษฐ์ (Artificial Intelligence) โดยการนำเสนอทฤษฎีฟัซซี่โลจิกโดยใช้เทคนิคฟัซซี่เซตเข้ามาช่วยเพื่อให้เกิดรายละเอียดและเกิดความชัดเจนในกระบวนการจำแนกและระบุโค้ด ข้อดีของการนำฟัซซี่เซตเข้ามาสนับสนุนเครื่องมือช่วยในการจำแนกและระบุโค้ดคือ ช่วยให้การตัดสินใจภายใต้เงื่อนไขที่ซับซ้อนของโค้ดโดยใช้เหตุผลเชิงตรรกะ ทำให้เข้าใจง่ายขึ้น และสามารถตีความให้อยู่ในรูป If-Then มีความสอดคล้องกับตรรกะความคิดของมนุษย์ การตัดสินใจด้วยฟัซซี่เซตไม่ได้มีคำตอบเพียงแค่ถูกหรือผิดเท่านั้น แต่จะแบ่งระดับค่าความถูกหรือผิดให้เห็นเป็นภาพข้อมูลที่เชื่อมโยงกัน ดังเช่น งานวิจัยของ Lotfi A. Zadeh

[26, 34, 35,36] ที่ศึกษาเกี่ยวกับทฤษฎีการรับรู้ของความเป็นไปได้ด้วยเหตุผล เพื่อช่วยในการวิเคราะห์และตัดสินใจในประเด็นที่มีปัญหาเรื่องความคลุมเครือและซับซ้อน โดยใช้หลักคณิตศาสตร์ โดยงานวิจัยของ Lotfi [24,36] นั้นสามารถสรุปได้ไว้ว่าทฤษฎีความน่าจะเป็นและฟัซซี่เซตสามารถนำมาใช้เป็นเทคนิคในการช่วยตัดสินใจคำถามที่เกิดขึ้นโดยมีพื้นฐานข้อมูลหรือการรับรู้ ทั้งนี้ Lotfi ได้แสดงความคิดเห็นว่า สิ่งที่ต้องการศึกษาต่อไป คือ พัฒนาทฤษฎีขึ้นอย่างเต็มที่ ทั้งนี้เพื่อต้องการพัฒนาความสามารถในการประมวลผลข้อมูลการรับรู้ที่ใช้เป็นพื้นฐานสำหรับการตัดสินใจในเรื่องความไม่แน่นอนอย่างมีเหตุผล งานวิทยานิพนธ์นี้เล็งเห็นประโยชน์ของฟัซซี่เซต จึงนำมาสนับสนุนกระบวนการจำแนกและระบุโค้ดให้ได้ผลลัพธ์แม่นยำ

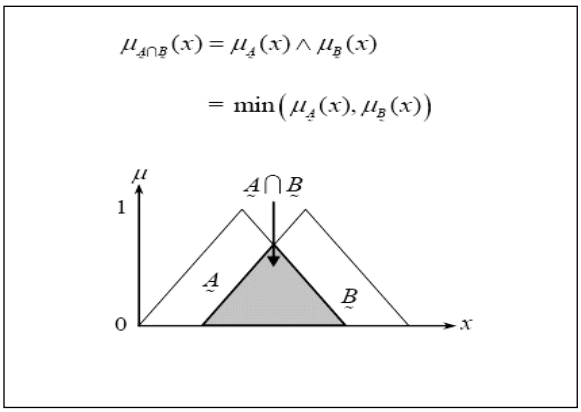
งานวิทยานิพนธ์นี้นำเทคนิคฟัซซี่เซตมาใช้ ซึ่งมีคุณสมบัติเหมือนกับเซตโดยทั่วไป มีการดำเนินการ(Operation) คือ Union และ Intersection

1) ยูเนียน (Union) ของฟัซซี่เซต จะเป็น OR การดำเนินการ ในสมการ ดังภาพที่ 2.6



ภาพที่ 2.6 ยูเนียนของฟัซซี่เซต A และ B [23]

2) อินเตอร์เซกชัน (Intersection) ของฟัซซี่เซต จะเป็น AND การดำเนินการ ในสมการดังภาพที่ 2.7



ภาพที่ 2.7 Intersection ของฟัซซี่เซต A และ B [23]

กฎการ แปล ความฟัซซี่นำมาใช้ในวิทยานิพนธ์นี้เพื่อการจำแนกร่องรอยไม่ดี ความคลุมเครือและ ซับเซตคลีนโค้ด โดยการกฎการตีความตามหลักของฟัซซี่มีรูปแบบที่สามารถอธิบายให้เข้าใจได้ง่ายใกล้เคียงกับการตัดสินใจของมนุษย์ โดยนำข้อมูลที่ได้จากการวัดมาใช้ในการประมวลผลตามกฎการตีความที่สร้างขึ้น แล้วผลลัพธ์ที่ได้นั้นจะจำแนกว่า โค้ดที่ถูกจำแนกนั้นอยู่ในกลุ่มใดในสามกลุ่ม เพื่อกำหนดวิธีการแก้ไขร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือต่อไป

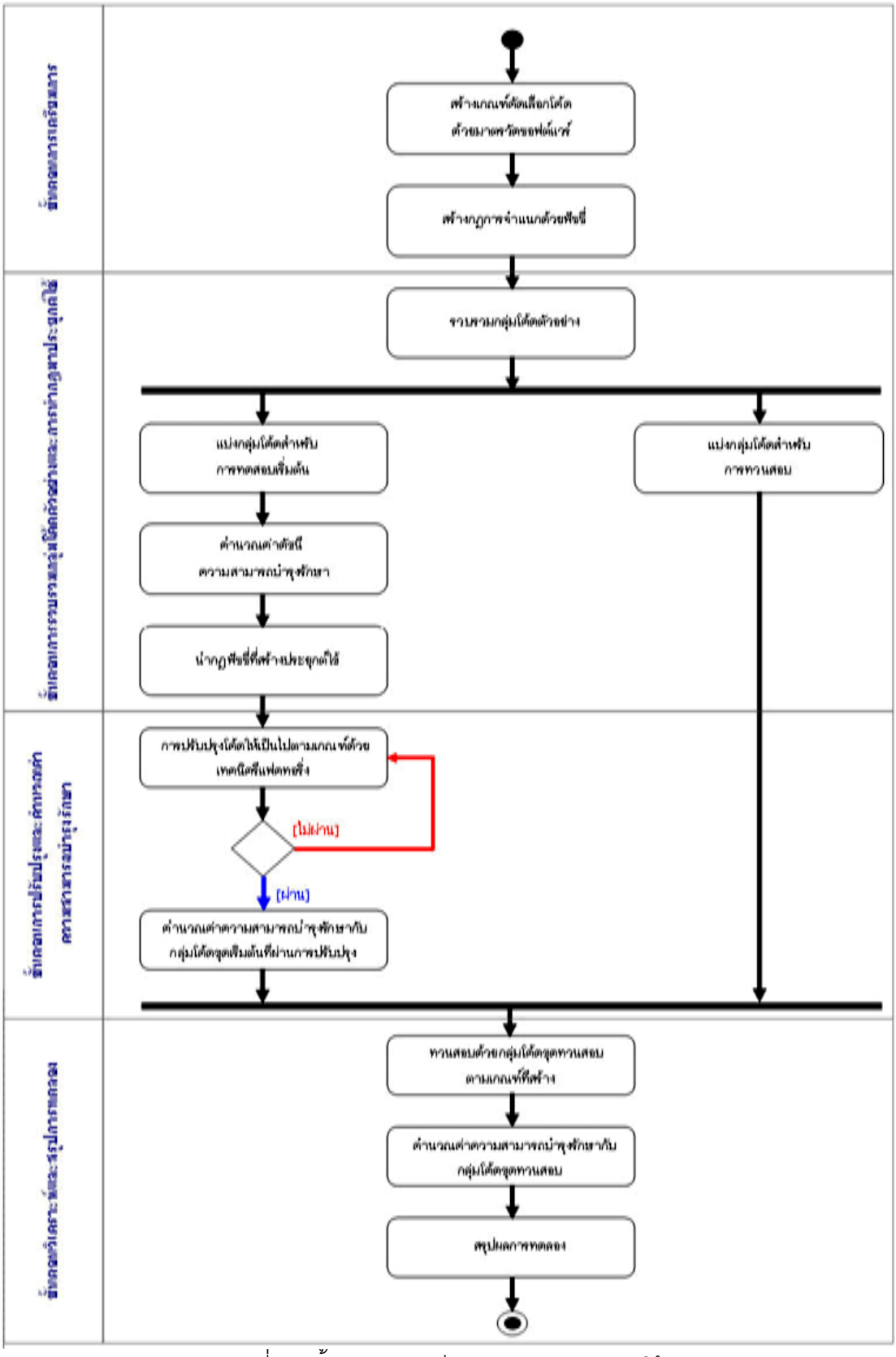
บทที่ 3
การวิเคราะห์และออกแบบวิธีการสร้างเกณฑ์การจำแนกโค้ด ด้วยมาตรวัดซอฟต์แวร์และฟัซซี่โลจิก

ขั้นตอนการวิเคราะห์และออกแบบวิธีการจำแนกโค้ดด้วยมาตรวัดซอฟต์แวร์และฟัซซี่โลจิกของวิทยานิพนธ์นี้ เริ่มจากการกำหนดชุดตัวอย่างเริ่มต้นทดสอบ โดยแบ่งออกเป็น 3 ประเภท ได้แก่ ซับเซตคลีนโค้ด ร่องรอยไม่ดี และความคลุมเครือ ซึ่งเกิดจากการทดสอบตามวิธีการที่ออกแบบไประยะหนึ่ง ทำให้พบว่า มีตัวอย่างโค้ดที่มีความคลุมเครือซึ่งไม่สามารถระบุได้ว่าควรจำแนกอยู่ในกลุ่มของซับเซตคลีนโค้ดหรือร่องรอยไม่ดี ผู้วิจัยจึงได้รวบรวมกลุ่มตัวอย่างโค้ดที่มีลักษณะความคลุมเครือมาทำการทดสอบเพื่อให้วิธีการจำแนกที่ออกแบบในวิทยานิพนธ์นี้มีความสมบูรณ์

จากนั้นจึงเริ่มออกแบบโดยนำมาตรวัดซอฟต์แวร์พื้นฐานมาใช้วัดค่าให้กับซับเซตคลีนโค้ด ความคลุมเครือ และร่อยรอยไม่ดี ในแต่ละรายการ แต่ผลลัพธ์ที่เกิดขึ้นไม่สามารถจำแนก ซับเซตคลีนโค้ดและความคลุมเครือได้อย่างเด่นชัด ดังนั้นจึงสร้างกฎการจำแนกด้วยฟัซซี่เพื่อประยุกต์ในการจำแนก ซับเซตคลีนโค้ดและ ความคลุมเครือ แล้วจึงเพิ่มกฎการจำแนกร่อยรอยไม่ดีเพื่อให้ขั้นตอนวิธีการจำแนกที่ออกแบบเป็นไปตามวิธีการเดียวกันและครอบคลุมขอบเขตประเภทของโค้ดที่ใช้ทดสอบทั้งหมด ซึ่งจะทำให้การแก้ไขคุณภาพของโค้ดเป็นไปเกณฑ์ของซับเซตคลีนโค้ด ที่ออกแบบไว้

ขั้นตอนต่อมาผู้วิจัยได้ออกแบบวิธีการปฏิบัติในการแก้ไขร่องรอยไม่ดีและความคลุมเครือ ซึ่งเกิดขึ้นจากผลลัพธ์การจำแนกโค้ดด้วยด้วย ฟัซซี่ สุดท้ายจึง ทำการทวนสอบเพื่อให้แน่ใจว่าวิธีการที่สร้างมีความถูกต้องแม่นยำและสามารถนำไปใช้งานจริงได้ และการปรับปรุงโค้ดให้เป็นซับเซตคลีนโค้ดที่ เป็นวิธีการที่เพิ่มค่าความสามารถใน การบำรุงรักษา อีกวิธีหนึ่งกับโค้ด ด้วย ดังภาพที่ 3.1 เป็นการอธิบายถึง ขั้นตอนการ ออกแบบ มาตรวัดซอฟต์แวร์ และขั้นตอนการสร้างกฎการจำแนก ให้แก่ทั้ง 3 กลุ่มโค้ดคือ กลุ่ม ซับเซตคลีนโค้ด กลุ่มโค้ดที่มีความคลุมเครือ และกลุ่มร่องรอยไม่ดี ที่ระบุไว้ในวิทยานิพนธ์นี้

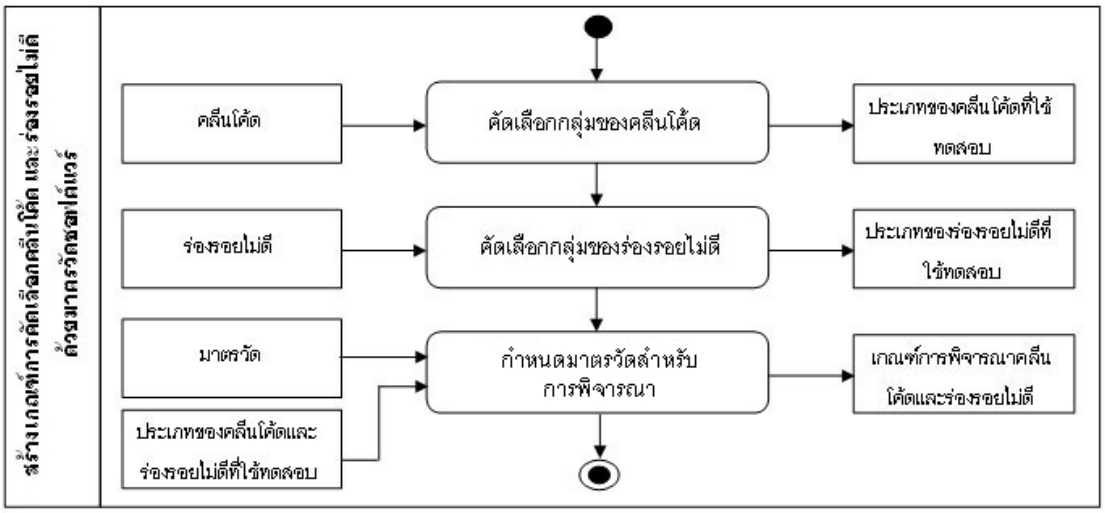
ส่วนในบทที่ 4 นั้น เป็นแสดงวิธีการจำแนกโค้ดและวิธีปฏิบัติในการแก้ไขร่องรอยไม่ดีและความคลุมเครือด้วยใช้เทคนิครีแฟคทอริ่ง เพื่อเพิ่ม คุณภาพของโค้ดให้เป็นไป เกณฑ์ของซับเซตคลีนโค้ดตามวิธีการที่ออกแบบไว้



ภาพที่ 3.1 ขั้นตอนของวิธีการปรับปรุงคุณภาพโค้ด

3.1 การสร้างเกณฑ์คัดเลือกซับเซตคลีนโค้ด ร่องรอยไม่ดีด้วยมาตรวัดซอฟต์แวร์

การสร้างวิธีการการจำแนกร่องรอยไม่ดีและซับเซตคลีนโค้ด เริ่มด้วยการกำหนดประเภทของซับเซตคลีนโค้ดและ ร่องรอยไม่ดีที่ใช้ในการทดลอง โดยซับเซตคลีนโค้ดและร่องรอยไม่ดีในแต่ละรายการจะถูกระบุค่าด้วยมาตรวัดซอฟต์แวร์ โดยผลลัพธ์ที่ได้จากขั้นตอนนี้จะนำไปใช้ในการสร้างเกณฑ์การจำแนกด้วยฟัซซี่ ดังภาพที่ 3.2 โดยมีรายละเอียดดังต่อไปนี้



ภาพที่ 3.2 การสร้างเกณฑ์การคัดเลือกร่องรอยไม่ดี ซับเซตคลีนโค้ด
ด้วยมาตรวัดซอฟต์แวร์

3.1.1 การคัดเลือกกลุ่มของซับเซตคลีนโค้ด

การคัดเลือกและจำแนกกลุ่มของซับเซตคลีนโค้ดต่างๆ ที่ถูกระบุไว้ในหนังสือ คลีนโค้ด ของ Robert C. Martin [9] จากการค้นคว้าพบว่า มีจำนวนคลีนโค้ดอยู่ทั้งสิ้น 12 แบบรูป ประมาณ 80 รายการ ซึ่งรายละเอียดรวบรวมไว้ใน บทที่2 ส่วนซับเซตคลีนโค้ดที่นำมาใช้ในทำการทดลองนั้น ผู้วิจัยคัดเลือกเฉพาะรายการที่มีความสัมพันธ์และกระทบต่อคุณภาพด้านความสามารถในการบำรุงรักษา อีกทั้งรายการที่คัดเลือก จำเป็นต้องผ่านเกณฑ์ในการคัดเลือกตามที่กำหนดและสามารถนำมาเป็นตัวอย่างในการทดลองในเบื้องต้น ซึ่งมีหลักเกณฑ์การคัดเลือกดังนี้

1) ต้องเป็นโค้ดที่สามารถอธิบายได้และมีรูปแบบที่ชัดเจน โดยมีคุณสมบัติ พื้นฐานที่สนับสนุนการเพิ่มโคฮีชันและลดคัปปลิงให้กับโค้ด ฟังก์ชัน และคลาส เป็นต้น นั่นคือเมื่อผู้พัฒนาพิจารณาโค้ดนั้นๆ จำเป็นต้องเข้าใจถึงลักษณะหรือคุณสมบัติของโค้ดที่ได้คัดเลือกว่ามีคุณสมบัติอย่างไร และเชื่อมโยงกับคุณสมบัติที่ดีของโคฮีชันและคับปลิงได้ [22]

2) สามารถนำมาตรวัดซอฟต์แวร์มาใช้อธิบายได้คือ ซอฟต์แวร์ที่เกิดจากการเขียนโปรแกรมอย่างมีระเบียบตามหลักการออกแบบและเขียนโปรแกรม[22] สามารถใช้มาตรวัดซอฟต์แวร์วัดได้ว่าโปรแกรมนั้นให้คุณภาพด้านใดบ้าง[4, 32] โดยมาตรวัดที่ถูกนำมาใช้ในการอธิบายนั้น จำเป็นต้องมีค่าที่ใช้ในการพิจารณาจากแหล่งอ้างอิงที่สามารถตรวจสอบได้[33, 37, 38]

3) เป็นโค้ดที่เขียนจากผู้พัฒนาไม่ได้ถูกสร้างขึ้นโดยอัตโนมัติจากเครื่องมือช่วยพัฒนา คือ การเขียนโค้ดที่ดี ผู้พัฒนาควรมีการวิเคราะห์ วางแผนเป็นอย่างดี ไม่ควรพึ่งพาเครื่องมือเป็นหลัก หากเกิดปัญหาจะทราบถึงจุดที่ต้องการแก้ไขได้อย่างรวดเร็ว

4) ไม่เจาะจงไปที่เครื่องมือที่ช่วยในการพัฒนาหรือระบบปฏิบัติการใด คือ ลักษณะของโค้ดที่ดี ควร มีความยืดหยุ่น (Flexible) สามารถนำไปปรับใช้ได้กับทุกสภาพแวดล้อม (Portability) เพื่อให้กระบวนการพัฒนาดำเนินไปได้อย่างรวดเร็ว

5) กรณีที่มีชนิดของซับเซตคลีนโค้ดถูกกล่าวซ้ำหลายครั้ง เช่น ซับเซตคลีนโค้ดชนิดคำอธิบายโปรแกรมที่มีการกล่าวซ้ำในส่วนของประเภทตัวแปร วัตถุ ฟังก์ชัน และคลาส ควรถูกนับรวมเป็นชนิดเดียวกัน ทั้งนี้ผู้วิจัยวิเคราะห์ว่า การกล่าวซ้ำหลายครั้ง ที่มีความหมายการทำงานเดียวกัน ควรรวมให้เป็นกลุ่มเดียวกันเพื่อไม่ให้เกิดการใช้งานที่ซ้ำซ้อนจนเกินไป

โดยวิทยานิพนธ์นี้คัดเลือกซับเซตคลีนโค้ดที่ผ่านเกณฑ์การพิจารณาทั้งหมด 3 ประเภท ได้แก่ Comment Functions และ Classes โดยมีรายการดังในตารางที่ 3.1 ซึ่งภายในตารางจะประกอบไปด้วย ลำดับที่เพื่อแสดงถึงส่วนย่อยของรายการซับเซตคลีนโค้ดที่อยู่ในกลุ่มฟังก์ชันและคลาส และรายการซับเซตคลีนโค้ดพร้อมรายละเอียด ซึ่งแนวคิดพร้อมแบบรูปของซับเซตคลีนโค้ดผู้วิจัยได้อธิบายไว้ในบทที่ 2

ตารางที่ 3.1 รายการซับเซตคลีนโค้ดที่ผ่านเกณฑ์การพิจารณา

ลำดับที่	รายการซับเซตคลีนโค้ด	รายละเอียด
1	Good Comment	การเขียนคำอธิบายเพื่อบรรยายการทำงานในแต่ละส่วนของโปรแกรม เพื่อให้สามารถทำความเข้าใจได้ง่ายขึ้น
2	Function	ฟังก์ชัน
2.1	Small Functions	ฟังก์ชันควรมีขนาดเล็กให้มากที่สุดเพื่อความสะดวกในการใช้งาน

ตารางที่ 3.1 รายการซับเซตคลีนโค้ดที่ผ่านเกณฑ์การพิจารณา (ต่อ)

ลำดับที่	รายการซับเซตคลีนโค้ด	รายละเอียด
2.2	Do One Thing	ฟังก์ชันควรมีการทำงานเพียงแค่หนึ่งอย่างหรือตอบสนองแค่เรื่องเดียวเท่านั้น
2.3	Nesting Depth	ฟังก์ชันที่มีระดับลึกของเมท็อดหรือขอบเขตการทำงานที่ลึกจะทำให้เกิดความยุ่งยากในการทดสอบ
2.4	Function Arguments	ฟังก์ชันที่มีพารามิเตอร์จำนวนมากส่งผลให้เกิดความซับซ้อนของการทำงานมากขึ้น
2.5	Structured Programming	ฟังก์ชันที่ต้องมีการทำงานในรูปแบบที่มีลำดับชัดเจนเพื่อให้ง่ายต่อการนำไปทดสอบ
3	Classes	คลาส
3.1	Class Organization	คลาสจำเป็นต้องมีการวางแผนและออกแบบการทำงานเพื่อให้ง่ายต่อการนำไปใช้งาน
3.2	Encapsulation	คลาสจำเป็นต้องมีการปกปิดข้อมูลและรายละเอียดของการทำงานไว้ภายใน
3.3	Classes Should Be Small	คลาสจำเป็นต้องมีขนาดเล็กเพื่อความสะดวกใช้งานและง่ายต่อการบำรุงรักษา
3.4	Maintaining Cohesion	คลาสจำเป็นต้องมีการทำงานร่วมกันระหว่างข้อมูลและเมท็อดที่อยู่ภายใน
3.5	Organizing for Change	คลาสจำเป็นต้องมีการวางแผนและออกแบบการทำงานเพื่อให้สามารถรองรับการเปลี่ยนแปลงได้
3.6	The Law of Demeter	คลาสจำเป็นต้องมีการวางแผนและออกแบบการทำงานในการเรียกใช้งานเมท็อดและข้อมูลต่างๆไม่ให้มีความซับซ้อน
3.7	Clean Boundary	คลาสจำเป็นต้องมีการวางแผนและออกแบบขอบเขตการทำงานภายในคลาสและการติดต่อระหว่างคลาสให้ชัดเจน

3.1.2 การ คัดเลือกกลุ่มของร่องรอยไม่ดี

การคัดเลือกร่องรอยไม่ดี ผู้วิจัยได้นำงานวิจัยของ Mäntylä [8] มาวิเคราะห์ก ารจัดระดับความยากง่ายในการตรวจจับร่องรอยไม่ดีไว้ ซึ่งระบุ ระดับไว้ตั้งแต่ 0 ถึง 5 โดยที่ระดับ 0 คือ ยากต่อการตรวจจับและ ระดับ 5 คือ ระดับที่ง่าย สามารถใช้มาตรวัดเพียงหนึ่งหรือสองชนิดในการตรวจจับเท่านั้น โดยงานวิทยานิพนธ์นี้เลือกระดับที่ 3 ถึง 5 เนื่องจากต้องการให้เป็นระบบแบบกึ่งอัตโนมัติ (Semi Auto-System) โดยไม่ต้องพึ่งพาผู้เชี่ยวชาญในนำข้อมูลเข้าในขั้นตอนการทำงาน โดย จะที่ตัดระดับ 0 ถึง 2 ออกซึ่งเป็นระดับมีความซับซ้อนมากจำเป็นต้องอาศัยผู้เชี่ยวชาญหรือความสามารถของคนเท่านั้นในการตัดสินใจ ทั้งนี้ร่องรอยไม่ดีทั้ง 8 รายการมีผลกระทบต่อคุณภาพด้านความสามารถในการบำรุงรักษาด้วย

โดยวิทยานิพนธ์นี้คัดเลือกร่องรอยไม่ดี 5 ชนิด จำนวน 8 รายการ ดังนี้

1) ชนิดที่ 1 The Bloaters ได้แก่ Long Method Large Class Long Parameter List

2) ชนิดที่ 2 The Object-Orientation Abusers ได้แก่ Temporary Field

3) ชนิดที่ 3 The Couplers ได้แก่ Feature Envy Inappropriate Intimacy

4) ชนิดที่ 4 The Dispensables ได้แก่ Lazy Class

5) ชนิดที่ 5 Others ได้แก่ Bad comment

โดยรายละเอียดของร่องรอยไม่ดีที่ผ่านการพิจารณาแสดงในตารางที่ 3.2 และรายละเอียดทั้งหมดของร่องรอยไม่ดีแสดงไว้ในภาคผนวก ก

ตารางที่ 3.2 รายการร่องรอยไม่ดีที่ผ่านเกณฑ์การพิจารณา

ลำดับที่	รายการร่องรอยไม่ดี	รายละเอียด
1	Long Method	ฟังก์ชันหรือเมท็อดที่มีขนาดใหญ่และมีความซ้ำซ้อนก่อให้เกิดความไม่อิสระในการเรียกใช้งานและยากต่อการบำรุงรักษา
2	Large Class	คลาสที่มีหน้าที่การทำงานมากเกินไปและ มีการทำงานภายในคลาสที่ซ้ำซ้อนส่งผลให้การบำรุงรักษาเป็นไปได้ยาก
3	Long Parameter List	จำนวนของพารามิเตอร์ที่ถูกส่งค่าผ่านฟังก์ชันหรือเมท็อดมากเกินไป เกิดความไม่สอดคล้องในการทำงานและทำให้เข้าใจยาก
4	Temporary Field	การจัดเก็บผลลัพธ์ชั่วคราวในระหว่างการทำงานของโปรแกรมเป็นจำนวนมาก ส่งผลให้เกิดความไม่เป็นระเบียบและไม่สามารถบ่งชี้ได้ว่า ส่วนไหนสำคัญหรือถูกใช้งานในสถานะปัจจุบัน

ตารางที่ 3.2 รายการร่องรอยไม่ดีที่ผ่านเกณฑ์การพิจารณา (ต่อ)

ลำดับที่	รายการร่องรอยไม่ดี	รายละเอียด
5	Feature Envy	การเรียกใช้งานเมท็อดหรือข้อมูลจากคลาสอื่นมากกว่าคลาสที่เป็นเจ้าของเมท็อดนั้นเป็นการออกแบบที่ไม่เหมาะสมและไม่เป็นอิสระในการทำงาน
6	Inappropriate Intimacy	การเรียกใช้งานเมท็อดภายในแต่ละคลาสที่มีโครงสร้างการเรียกใช้งานที่ซับซ้อนจากการออกแบบที่ไม่เหมาะสมทำให้เรียกใช้งานยุ่งยากและใช้เวลานาน
7	Lazy Class	คลาสที่มีการทำงานที่ไม่สำคัญหรือไม่มีประโยชน์ต่อโปรแกรมส่งผลให้เสียเวลาในการบำรุงรักษา
8	Bad Comment	ไม่พบคำอธิบายใดๆ ในโปรแกรม

3.1.3 การกำหนดมาตรวัดซอฟต์แวร์ที่เกี่ยวข้อง

1) มาตรวัดซอฟต์แวร์ที่ใช้วัดลักษณะการออกแบบซอฟต์แวร์

เป็นที่เข้าใจกันดีถึงปัญหาที่พบในกระบวนการพัฒนา ว่า ขั้นตอนใดที่พัฒนายากและค่าใช้จ่ายสูงจะถูกยกเลิก ด้วยเหตุนี้โมดูลที่สามารถระบุปัญหาได้อย่างมีประสิทธิภาพก่อนตัดสินใจวางระบบออกแบบให้กับโค้ดจึงเป็นวัตถุประสงค์หลักของ การออกแบบมาตรวัด มีการค้นพบว่า การออกแบบมาตรวัดมีส่วนเชื่อมโยงไปถึงคุณสมบัติของซอฟต์แวร์ ตัวอย่างเช่น Reliability Testability และ Maintainability ซึ่งแต่ละโปรแกรมประยุกต์สามารถนำ มาตรวัดมาเป็นส่วนช่วยให้ผู้ออกแบบทำงานได้ดี ซึ่งอัตราการใช้เครื่องมือ Computer Aid Software Engineering (CASE) หรือแม้แต่อุปกรณ์การเขียนโค้ดถูกสร้างเพิ่มขึ้นอยู่เป็นลำดับ อย่างไรก็ตามอาจเป็นประเด็นที่ สามารถสร้างความ เสี่ยงได้เช่นกัน เมื่อกล่าวถึงประเด็นที่น่าสนใจของการออกแบบมาตรวัดในช่วงปี 2533 โดยส่วนใหญ่จะเป็นระบบที่ใช้ตัวเลขหรือสัญลักษณ์ ซึ่งในเครื่องมือ CASE คู่แข่งบางแห่งได้ออกแบบมาตรวัดเชิงวัตถุมาเป็นส่วนประกอบ

2) มาตรวัดซอฟต์แวร์ที่ใช้วัดขนาดของโค้ด

ขนาดของมาตรวัดใช้เพื่อแสดงการเชื่อมโยง ถึง ความซับซ้อนและความแตกต่างในสิ่งที่ไม่ชัดเจนของคลาสที่มีขนาดใหญ่หรือเมท็อดที่ยากจะเข้าใจ ที่สามารถนำกลับมาใช้ใหม่ฉะนั้นเพื่อการทดสอบหรือเพื่อการบำรุงรักษา ขนาดของมาตรวัดสำหรับคลาสจะเป็นการสะท้อนให้เห็นถึง จำนวนค่าความพยายาม ที่ต้องใช้สร้างความเข้าใจและการบำรุงรักษาคลาสจึงเป็น

เหตุผลที่นำไปปรับใช้กับในเรื่องที่ซับซ้อน ขนาดของ มาตรวัด ถือเป็นส่วนสำคัญสำหรับการคาดการณ์ทรัพยากรที่ใช้ในกระบวนการ ซึ่งโมเดลที่ใช้งานส่วนใหญ่นั้น สามารถ แทนขนาดของวัตถุด้วย (LOCs) กับความซับซ้อนของวัตถุ (Some Subjective for Complexity Measures) นอกจากนี้ยังเป็นส่วนสำคัญของมาตรฐานในการนำมาตรวัดมาประกอบรวมกัน ถ้าไม่มีการกำหนดมาตรฐานไว้ ก็จะไม่มีความหมายใดๆที่จะนำไปเปรียบเทียบ แอตทริบิวต์ เช่น การคำนวณหาผลรวมของค่าความพยายาม (Total Effort) การคำนวณหาค่า Optimal Number of Test Case หรือ จำนวนของความผิดพลาดที่เกิดใน หลายๆโครงการ (Number of Failures among Several Projects) ดังนั้น มาตรวัด ซอฟต์แวร์ที่ใช้ในการวัด จะช่วย ในการเปรียบเทียบระหว่างโครงการได้อย่างสะดวก

3) มาตรวัดซอฟต์แวร์ที่ใช้วัดความซับซ้อนของโค้ด

การวิเคราะห์ฟังก์ชันพอยต์(Function Points Analysis: FPA) เป็นประเด็นที่มุ่งไปยังการวัดขนาดและความซับซ้อนของระบบซอฟต์แวร์ ซึ่งเป็นมาตรฐานที่ชี้วัดความสามารถพบปัญหาการพึ่งพาภาษา(language dependencies) ได้อย่างครอบคลุมซึ่งสามารถพบได้ทุกที่เมื่อใช้ LOC เมื่ออยู่ในสถานะมุมมองผู้ใช้ ทั้งนี้แสดงให้เห็นถึงฟังก์ชัน point count อีกด้วย อย่างไรก็ตามวัตถุที่ถูกนำมาคำนวณเช่น ไฟล์ที่อยู่ภายในไฟล์ที่อยู่ข้างนอกและส่วนต่อประสานที่อยู่ใ น กระบวนการพัฒนาโดยใช้กลุ่มของภาษาที่มีแบบแผน(procedural languages) นั้นไม่ควรเอามาปรับใช้กับการพัฒนาแบบเชิงวัตถุ เพราะ หลักเกณฑ์ ในการใช้งานจำเป็นต้องขึ้นอยู่กับ ภาษาที่เลือกใช้

ตารางที่ 3.3 ประเภทของมาตรวัดซอฟต์แวร์

มาตรวัดซอฟต์แวร์ที่ถูกกำหนดเพื่อใช้ในการจำแนกโค้ด	
มาตรวัดซอฟต์แวร์ที่ใช้วัดลักษณะการออกแบบซอฟต์แวร์	1) Depth of Inheritance Tree 2) Lack of Cohesion Of Methods 3) Lack of Cohesion Of Methods for Henderson-Sellers 4) Percentage of Comment 5) Relational Cohesion

ตารางที่ 3.3 ประเภทของมาตรวัดซอฟต์แวร์ (ต่อ)

มาตรวัดซอฟต์แวร์ที่ถูกกำหนดเพื่อใช้ในการจำแนกโค้ด	
มาตรวัดซอฟต์แวร์ที่ใช้วัดขนาดของโค้ด	1) Number of Fields 2) Number Inline of Instructions 3) Number Line of Codes 4) Number of Interfaces 5) Number of Class not Inheritent much be Seal or Static 6) Number of Methods 7) Number of Overloads 8) Number of Parameters 9) Number of Variables 10) Number of Public Fields 11) Lazy between Method and Field in Class level
มาตรวัดซอฟต์แวร์ที่ใช้วัดความซับซ้อนของโค้ด	1) Afferent Coupling at Method Level 2) Afferent Coupling at Public Field Level 3) Coupling between Object Classes 4) Cyclomatic Complexity 5) Distance from the Main Sequence 6) Efferent Coupling at Method Level 7) Inline of Cyclomatic Complexity 8) Inline of Nesting Depth 9) Is a Structure 10) Number of Children 11) Strange of Afferent Coupling-Outer Method and Field Level 12) Strange of Efferent Coupling-Inner Method Level

ซึ่งการกำหนดมาตรวัดสำหรับการพิจารณา เริ่มจากการนิยามและอธิบายรายละเอียดให้แก่ ซับเซตคลีนโค้ดและร่องรอยไม่ดีแต่ละรายการที่เลือกใช้ในงานวิทยานิพนธ์นี้ ดังเกณฑ์ที่กล่าวมาในข้อ 3.1.1 และ 3.1.2 แล้วจึงกำหนดมาตรวัดในการอธิบาย ซึ่งตารางที่ 3.3 ใช้สำหรับงาน

วิทยานิพนธ์นี้ จะประกอบด้วยนิยาม แรงจูงใจ มาตรวัด ที่ใช้ในการอธิบาย และค่าที่ใช้ในการพิจารณาซับเซตคลีนโค้ด ซับเซตคลีนโค้ดประเภทฟังก์ชันได้ในภาคผนวก ข และ ในส่วนร่องรอยไม่ดี แสดงรายละเอียดไว้ในภาคผนวก ค

หลังจากเลือกมาตรวัดให้แก่กลุ่มของซับเซตคลีนโค้ดและร่องรอยไม่ดี ตามตารางที่ 2.3 และตารางที่ 2.4 ซึ่งปรากฏอยู่ในบทที่ 2 หน้า 36 และ 37 เรียบร้อย จะสังเกตพบความสัมพันธ์ระหว่างร่องรอยไม่ดีและซับเซตคลีนโค้ด ที่ถูกวัดโดยมาตรวัดเดียวกัน อาทิเช่น ซับเซตคลีนโค้ดประเภท Small Functions กับร่องรอยไม่ดีประเภท Long Method ที่เลือกใช้มาตรวัด ซอฟต์แวร์ Number Line of Codes โดยพิจารณาจากจำนวนของบรรทัดโค้ดในฟังก์ชันหนึ่งถ้าน้อยกว่า 22 บรรทัด ถือว่าเป็นประเภท Small Functions ในส่วนซับเซตคลีนโค้ด ในกรณีค่าที่วัดได้มากกว่า 60 บรรทัดถือว่ามีลักษณะในกลุ่มของร่องรอยไม่ดี

ทั้งนี้จำเป็นต้องใช้มาตรวัดอื่นๆ ประกอบในการพิจารณาด้วย ดังนั้นกรณีที่ค่าพิจารณาอยู่ระหว่าง 23 ถึง 59 ไม่สามารถระบุได้ว่า ควรจำแนก อยู่ในกลุ่มของซับเซตคลีนโค้ดหรือกลุ่มร่องรอยไม่ดี ซึ่งในที่นี้ขอนิยามกลุ่มโค้ดประเภทนี้ว่า “ความคลุมเครือ” วิทยานิพนธ์นี้ต้องการที่จะแก้ไขความคลุมเครือให้อยู่ในรูปของซับเซตคลีนโค้ด ซึ่งกลุ่มโค้ดที่มีความคลุมเครือจะต้องผ่านการปรับปรุงคุณภาพโค้ด เช่นเดียวกับกลุ่มร่องรอยไม่ดี รายละเอียดของกลุ่มโค้ดที่มีความคลุมเครือ แสดงในตารางที่ 3.4 ซึ่งใช้ลำดับที่เพื่อแสดงถึงส่วนย่อยของรายการความคลุมเครือที่อยู่ในกลุ่มฟังก์ชันและคลาส

ภาพที่ 3.3 ความคลุมเครือที่เกิดขึ้นระหว่างซับเซตคลีนโค้ดและร่องรอยไม่ดี
เมื่อนำมาใช้วัดในมาตรวัดซอฟต์แวร์เดียวกัน

จากภาพที่ 3.3 แสดงค่าความคลุมเครือที่เกิดขึ้นเมื่อนำค่าพิจารณาของซับเซตคลีนโค้ดมาใช้ จะแทนด้วยสัญลักษณ์A และกลุ่มร่องรอยไม่ดีที่เกิดขึ้นจะแทนด้วยสัญลักษณ์C ส่วนสัญลักษณ์B นั้นคือ ความคลุมเครือที่เกิดขึ้น ในมาตรวัดซอฟต์แวร์เดียวกัน เพื่อไม่ให้เกิดความสับสนในการนำไปใช้ในการคัดแยกด้วยกฎของฟัซซี่ ผู้วิจัยจึงขอกำหนดให้ความคลุมเครือที่เกิดขึ้นนี้เฉพาะกับมาตรวัดซอฟต์แวร์ที่ถูกนำไปใช้พิจารณาซับเซตคลีนโค้ดเท่านั้น

ฉะนั้นเมื่อมาตรวัดใดๆ ก็ตามที่ถูกนำไปพิจารณาซับเซตคลีนโค้ด มีส่วนทำให้ เกิดความคลุมเครือ เกิดขึ้นเสมอ และในกรณีที่มาตร วัดซอฟต์แวร์ที่ ใช้วัดเฉพาะกับร่องรอยไม่ดี อย่างเดียวจะไม่มีการระบุส่วนของความคลุมเครือลงไป ด้วย เนื่องจากวิธีการที่ ผู้วิจัย สร้างต้องการมุ่งเน้นเพื่อแก้ไขปัญหาร่องรอยไม่ดีให้หมดไป ไม่ใช่แก้ไขเพื่อให้เกิดความคลุมเครือ และเพื่อให้ได้คำตอบที่เด่นชัดจึงนำกฎการจำแนกด้วยฟัซซี่มาช่วยในการจำแนกโค้ด

ตารางที่ 3.4 รายการโค้ดที่มีความคลุมเครือ

ลำดับที่	รายการโค้ดที่มีความคลุมเครือ	รายละเอียด
1	Ambiguity in Comment	ความคลุมเค รือที่เกิดจากจำนวน ของ คำอธิบายการทำงานของโปรแกรมมีน้อยเกินไปทำให้เข้าใจการทำงานของโปรแกรมได้ยาก
2	Ambiguity of Function	ความคลุมเครือที่เกิดจากการออกแบบและการใช้งานในฟังก์ชันหรือเมท็อด
2.1	Ambiguity in Method	ความคลุมเครือที่เกิดจากขนาดของฟังก์ชันหรือเมท็อดที่เริ่มมีขนาดใหญ่ทำให้เกิดความลำบากในการนำไปใช้งาน
2.2	Ambiguity in One Thing	ความคลุมเค รือที่เกิดจากการทำงานของฟังก์ชันหรือเมท็อดที่ทำงานหรือตอบสนองมากกว่าหนึ่งอย่าง
2.3	Ambiguity in Nesting Depth	ความคลุมเครือที่เกิดจากระดับความลึกของฟังก์ชันหรือเมท็อดที่เริ่มมีความลึกมากทำให้เริ่มมีความยุ่งยากในการทดสอบ
2.4	Ambiguity in Arguments	ความคลุมเครือที่เกิดจากจำนวนพารามิเตอร์ของฟังก์ชันหรือเมท็อดที่เริ่มมีจำนวนมากทำให้เกิดความซับซ้อนในการทำงาน

ตารางที่ 3.4 รายการโค้ดที่มีความคลุมเครือ (ต่อ)

ลำดับที่	รายการโค้ดที่มีความคลุมเครือ	รายละเอียด
2.5	Ambiguity in Structured Programming	ความคลุมเค รือที่เกิดจากรูปแบบที่มีลำดับที่ไม่ชัดเจนเพื่อให้ยากต่อการนำไปทดสอบ
3	Ambiguity of Classes	ความคลุมเครือที่เกิดจากการออกแบบและการนำไปใช้งานในคลาส
3.1	Ambiguity in Class Organization	ความคลุมเครือที่เกิดขึ้นกับคลาสที่ขาดการวางแผนและออกแบบการทำงานเพื่อให้ยุ่งยากในการนำไปใช้งาน
3.2	Ambiguity in Encapsulation	ความคลุมเครือที่เกิดขึ้นกับคลาสที่มีไม่มีการปกปิดข้อมูลและรายละเอียดของการทำงานไว้ภายในอย่างรอบคอบ
3.3	Ambiguity in Classes	ความคลุมเค รือที่เกิดขึ้นกับคลาสที่เริ่มมีขนาดใหญ่ทำให้เกิดความยุ่งยากในการใช้งานและบำรุงรักษาได้ยาก
3.4	Ambiguity in Cohesion	ความคลุมเค รือที่เกิดขึ้นกับคลาสที่มีการทำงานไม่สัมพันธ์ระหว่างข้อมูลและเมท็อดที่อยู่ภายใน
3.5	Ambiguity for Change	ความคลุมเครือที่เกิดขึ้นกับคลาสที่ขาดการวางแผนและออกแบบการทำงานทำให้ไม่สามารถรองรับการเปลี่ยนแปลงได้
3.6	Ambiguity of Demeter	ความคลุมเครือที่เกิดขึ้นกับคลาสที่ขาดการวางแผนและออกแบบการทำงานทำให้การเรียกใช้งานเมท็อดและข้อมูลต่างๆเกิดความความซับซ้อน
3.7	Ambiguity in Boundary	ความคลุมเครือที่เกิดขึ้นกับคลาสที่ขาดการวางแผนและออกแบบทำให้ขอบเขตการทำงานภายในคลาสและการติดต่อระหว่างคลาสไม่ชัดเจน

มาตรวัดซอฟต์แวร์และเกณฑ์พิจารณาสำหรับโค้ดที่มีความคลุมเครือ แสดงรายละเอียดในส่วนย่อยในกลุ่มฟังก์ชันและคลาส ของในตารางที่ 3.5

ตารางที่ 3.5 เกณฑ์การพิจารณาโค้ดที่มีความคลุมเครือ

ลำดับที่	รายการ	มาตรวัดซอฟต์แวร์	เกณฑ์พิจารณา
1	Ambiguity in Comment	PCC	0<x<20
2.1	Ambiguity in Method	NLOC	22<x<60
		NILI	50<x<200
		CC	10<x<30
		ILCC	20<x<60
2.2	Ambiguity in One Thing	RC	1.5>x>3.5
2.3	Ambiguity in Nesting Depth	ILND	x>4
2.4	Ambiguity in Arguments	NOP	5<x<=7
2.5	Ambiguity in Structured Programming	ISS	0>x>1
3.1	Ambiguity in Class Organization	DIT	x>=6
		NOC	x>=6
		NOI	x>=20
		NOIM	x>=1
3.2	Ambiguity in Encapsulation	NPF	x>=1
3.3	Ambiguity in Classes	NOP	x>5
		NOV	x>8
		NOO	x>6
		NOM	15<x<=20
		NFD	10<x<=20
3.4	Ambiguity in Cohesion	LCOM1	0.5<x<=0.8
		LCOM3	0.8<x<=1.0
3.5	Ambiguity for Change	ACML	x<=0
		ECML	x>50
3.6	Ambiguity of Demeter	DMS	x>0.5
3.7	Ambiguity in Boundary	CBO	x>6

ตารางที่ 3.6 แสดงความถี่ของมาตรวัดซอฟต์แวร์ที่ถูกเลือกใช้ในการจำแนกกลุ่มซับเซตคลีนโค้ด กลุ่มโค้ดที่มีความคลุมเครือและกลุ่มร่องรอยไม่ดี

Categories	Id	Description	ACML	ACAMM	CBO	CC	DIT	DMS	ECML	ILCC	ILND	ISS	LCOMT	LCOM5	LMFC	MFD	MLI	NLOC	NOC	NOI	NOM	NOM	NOO	NOP	NOV	NPF	PCC	RC	SAC-OMF	SEC-IM
Clean Code	C1	Good Comment																								○				
	C2	Function																												
	C2.1	Small Functions				○				○							○	○												
	C2.2	Do One Thing																									●			
	C2.3	Nesting Depth									●																			
	C2.4	Function Arguments																					○							
	C2.5	Structured Programming										●																		
	C3	Classes																												
	C3.1	Class Organization					●											●	●	●										
	C3.2	Encapsulation																							●					
	C3.3	Classes Should Be Small													○						○	●	○	●						
	C3.4	Maintaining Cohesion											○	○																
	C3.5	Organizing for Change	●					●																						
	C3.6	The Law of Demeter					●																							
	C3.7	Clean Boundary			●																									
Ambiguity Code	A1	Ambiguity in Comment																								○				
	A2	Ambiguity of Function																												
	A2.1	Ambiguity in Method				○				○							○	○												
	A2.2	Ambiguity in One Thing																									●			
	A2.3	Ambiguity in Nesting Depth									●																			
	A2.4	Ambiguity in Arguments																					○							
	A2.5	Ambiguity in Structured Programming										●																		
	A3	Ambiguity of Classes																												
	A3.1	Ambiguity in Class Organization					●											●	●	●										
	A3.2	Ambiguity in Encapsulation																							●					
	A3.3	Ambiguity in Classes													○						○	●	○	●						
	A3.4	Ambiguity in Cohesion											○	○																
	A3.5	Ambiguity for Change	●					●																						
	A3.6	Ambiguity of Demeter					●																							
	A3.7	Ambiguity in Boundary			●																									
Bad Smell	B1	Long Method				○				○							○	○												
	B2	Large Class											○	○		○						○								
	B3	Long Parameter List																					○							
	B4	Temporary Field		●																										
	B5	Feature Envy																										●		
	B6	Inappropriate Intimacy																											●	
	B7	Lazy Class												●																
	B8	Bad Comment																								○				
Total			2	1	2	3	2	2	2	3	2	2	3	3	1	3	3	3	2	2	2	3	2	5	2	2	3	2	1	1

○ All Categories ● Clean & Ambiguity Code ● Only Bad Smell

จากตารางที่ 3.6 แสดงความสัมพันธ์ระหว่างมาตรวัดซอฟต์แวร์ที่ถูกคัดเลือกมาใช้ในการอธิบายซับเซตคลีนโค้ด ความคลุมเครือ และร่องรอยไม่ดี กลุ่มโค้ดแต่ละประเภทใน งานวิทยานิพนธ์นี้ โดยใช้สัญลักษณ์

1) วงกลมสีขาว หมายถึง มาตรวัดนี้ถูกใช้อธิบายซับเซตคลีนโค้ด ความคลุมเครือ และร่องรอยไม่ดี ทั้งสามชนิด

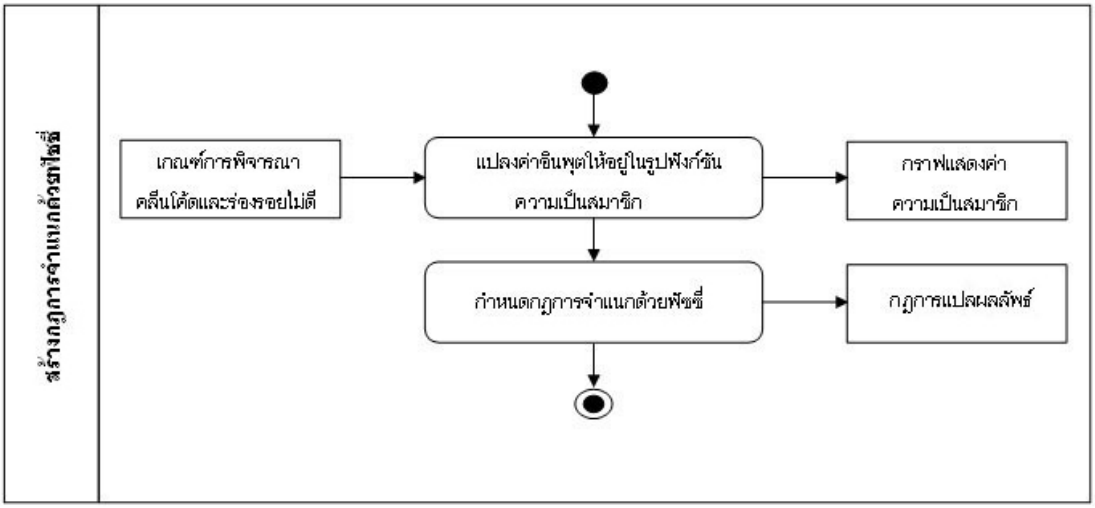
2) วงกลมสีเทา หมายถึง มาตรวัดนี้ถูกใช้อธิบายให้กับซับเซตคลีนโค้ดและความคลุมเครือ

3) วงกลมสีดำ หมายถึง มาตรวัดนี้ถูกใช้อธิบายเฉพาะกลุ่มร่องรอยไม่ดี

ส่วนแถวสุดท้ายของตารางแสดงความถี่ของแต่ละมาตรวัดซอฟต์แวร์ที่ถูกใช้ในการพิจารณาซับเซตคลีนโค้ด ความคลุมเครือ และร่องรอยไม่ดี ทั้งนี้หากพิจารณาจำนวนของมาตรวัดซอฟต์แวร์ที่ถูกใช้เพียงครั้งเดียว จะมีแค่กรณีที่ใช้พิจารณาร่องรอยไม่ดีเท่านั้น ดังนั้นมาตรวัดซอฟต์แวร์ในกลุ่มนี้สามารถจำแนกร่องรอยไม่ดีได้โดยไม่จำเป็นต้องพึ่งพากฎการจำแนกด้วยฟัซซี่ตามที่ออกแบบไว้ แต่ในกรณีร่องรอยไม่ดีประเภทอื่นๆ ยังมีความจำเป็นต้องพึ่งพากฎการจำแนกด้วยฟัซซี่ที่ออกแบบมาเพื่อประยุกต์ใช้ในการจำแนกเพิ่มเติม และมาตรวัดซอฟต์แวร์ที่ถูกใช้มากที่สุดคือ Number of Parameter ซึ่งเป็นมาตรวัดที่ใช้วัดจำนวนพารามิเตอร์นำเข้าของโปรแกรม โดยมีกฎที่ใช้อธิบายหลายข้อและใช้มาตรวัดนี้เป็นส่วนประกอบในการจำแนกตามวิธีการที่ผู้วิจัยออกแบบไว้

3.2 การสร้างกฎการจำแนกซับเซตคลีนโค้ด ร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือด้วยฟัซซี่โลจิก

การสร้างกฎการจำแนกซับเซตคลีนโค้ด ร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือด้วยฟัซซี่โลจิก ฟัซซี่โลจิกเป็นศาสตร์ที่ช่วยแปลความความคิดที่ซับซ้อน ความเชี่ยวชาญที่มนุษย์พยายามจัดการมาเป็นเวลานานหรือการรับรู้ของมนุษย์ โดยสร้างอินพุตเซตจากข้อมูลที่ใช้ในการพิจารณาไปสู่เอาต์พุตเซตที่สร้างขึ้นโดยสร้างกฎการแปลความ(Implication) เพื่อพิจารณาจำแนกความเป็นซับเซตคลีนโค้ด โค้ดที่มีความคลุมเครือและร่องรอยไม่ดี ดังภาพที่ 3.4 โดยมีขั้นตอนดังต่อไปนี้



ภาพที่ 3.4 สร้างกฎการจำแนกด้วยฟัซซี่

3.2.1 การแปลงค่าอินพุตให้อยู่ในรูปฟังก์ชันความเป็นสมาชิก

การนำค่าพิจารณาในแต่ละมาตรวัดที่ระบุให้แก่ซั บเซตคลีนโค้ด ความคลุมเครือ และร่องรอยไม่ดีมาทำการแปลงค่าให้อยู่ในรูปการจำแนกด้วยฟัซซี่ โดยเริ่มจากการแปลงค่ามาตรวัดให้อยู่ในรูปของตัวแปรที่ใช้ในกฎการจำแนกด้วยฟัซซี่ ในการแปลงค่าพิจารณา ที่อยู่ในตัวแปรนั้นพิจารณาจากช่วงของค่ามาตรวัดที่กำหนดไว้ หากค่าช่วงที่แบ่งมีน้อยกว่า 2 ช่วง จะนิยามตัวแปร 2 ตัวคือ Less กับ More และหากช่วงมี 3 ช่วงจะนิยามตัวแปรใหม่เป็น 3 ช่วง ได้แก่ Low Moderate และ High ตามลำดับ โดยรายละเอียดของค่าพิจารณาในแต่ละมาตรวัดและค่าตัวช่วงที่กำหนดให้ ดังตารางที่ 3.7 และ 3.8

ตารางที่ 3.7 การกำหนดตัวแปรฟัซซี่ให้กับค่าพิจารณาในแต่ละมาตรวัดกรณีที่มีค่า 2 ช่วง

มาตรวัดซอฟต์แวร์	ค่าตัวแปรช่วงที่กำหนดให้	
	Less(x)	More(x)
ACML	x<=0	x>0
AC-NPF	x<=3	-
CBO	x<=6	x>6
DIT	x<6	x>=6
DMS	x<=0.5	x>0.5
ECML	x<=50	x>50

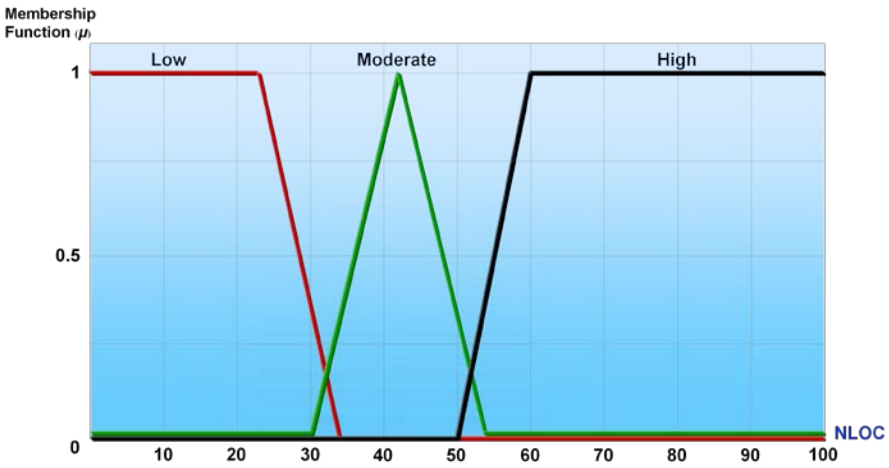
ตารางที่ 3.7 การกำหนดตัวแปรฟัซซี่ให้กับค่าพิจารณาในแต่ละมาตรวัดกรณีที่มีค่า 2 ช่วง (ต่อ)

มาตรวัดซอฟต์แวร์	ค่าตัวแปรช่วงที่กำหนดให้	
	Less(x)	More(x)
ILND	x<=4	x>4
ISS	x<1	x=1
LMFC	-	x>0
NOC	x<6	x>=6
NOI	x<20	x>=20
NOIM	x<1	x>=1
NOO	x<=6	x>6
NOV	x<=8	x>8
NPF	x<1	x>=1
SAC-OMF	-	x>0
SEC-IM	-	x>5

ตารางที่ 3.8 การกำหนดตัวแปรฟัซซี่ให้กับค่าพิจารณาในแต่ละมาตรวัดกรณีที่มีค่า 3 ช่วง

มาตรวัดซอฟต์แวร์	ค่าตัวแปรช่วงที่กำหนดให้		
	Low(x)	Moderate(x)	High(x)
CC	x<=10	10<x<30	x>=30
ILCC	x<=20	20<x<60	x>=60
LCOM1	x<=0.5	0.5<x<=0.8	x>0.8
LCOM3	x<=0.8	0.8<x<=1.0	x>1.0
NFD	x<=10	10<x<=20	x>20
NILI	x<=50	50<x<200	x>=200
NLOC	x<=22	22<x<60	x>=60
NOM	x<=15	15<x<=20	x>20
NOP	x<=5	5<x<=7	x>7
RC	x<1.5	1.5<=x<=3.5	x>3.5
PCC	X=0	0<x<20	x>=20

เมื่อทำการกำหนดช่วงให้กับค่าที่พิจารณาตามแต่ละมาตรวัดแล้ว จากนั้นจึงนำค่าที่วัดได้จากตัวอย่างโค้ดทั้งสามกลุ่มมาสร้างและแปลงเป็นอินพุตเซตสร้างเป็นข้อมูลนำเข้า เพื่อใช้ในการจำแนกด้วยฟัซซี่ ซึ่งการแปลงค่าอินพุตให้อยู่ในค่าฟัซซี่ (Fuzzification) คือ การคำนวณค่าฟัซซี่ผ่านฟังก์ชันความเป็นสมาชิกตามกฎของฟัซซี่เพื่อหาค่าดีกรีระหว่าง 0 ถึง 1 ดังภาพที่ 3.5 ดังต่อไปนี้



ภาพที่ 3.5 กราฟแสดงความคลุมเครือของร่องรอยไม่ดี
ชนิด Long Method

จากภาพกราฟแสดงการแปลงค่าให้อยู่ในรูปของอินพุตเซตของฟัซซี่ ที่สร้างยกตัวอย่าง เช่น การแปลค่าจากมาตรวัด Number Line of Codes โดยเส้นกราฟซ้ายมือจะแสดงค่าความเป็นสมาชิกความเป็นคลีนโค้ด เมื่อค่าที่วัดได้ต่ำกว่า 22 บรรทัด ส่วนเส้นกราฟขวามือจะแสดงถึงค่าความเป็นสมาชิกของร่องรอยไม่ดี เมื่อค่าที่วัดได้มากกว่า 60 บรรทัด และ เส้นกราฟตรงกลางจะแสดงความคลุมเครือที่เกิดจากขึ้นเมื่อค่าได้วัดได้นั้นไม่อยู่ในช่วงของซับเซตคลีนโค้ดและร่องรอยไม่ดี โดยจำนวนของเส้นกราฟที่แสดงความเป็นสมาชิกสามารถคำนวณได้จากผลคูณของมาตรวัดที่ใช้วัดซับเซตคลีนโค้ด ร่องรอยไม่ดี และความคลุมเครือซึ่งกราฟที่สร้าง ขึ้นเพื่อใช้ในขั้นตอนการกำหนดอินพุตเซตของฟัซซี่

3.2.2 การสร้างกฎการจำแนก กลุ่มซับเซตคลีนโค้ด กลุ่มโค้ดที่มีความคลุมเครือและกลุ่มร่องรอยไม่ดีด้วยฟัซซี่

เมื่อทำการแปลงค่าพิจารณาในแต่ละมาตรวัดให้อยู่ในรูปฟัซซี่แล้ว ในขั้นตอนนี้เป็นการตั้งกฎต่างในการแปลความขึ้นโดยกฎของฟัซซี่นั้นจะมีลักษณะ (IF-THEN rule-based form)

เป็นรูปแบบการอนุมานจากความจริงที่ทราบอยู่ก่อนแล้วเราสามารถหาข้อสรุปความจริงอีกอย่างหนึ่งได้ ซึ่งในตารางที่ 3.9 กฎที่ใช้อธิบายรายละเอียดซับเซตคลีนโค้ด ผู้วิจัยได้วิเคราะห์และนำรายละเอียดมาจาก Robert C. Martin ส่วนในตารางที่ 3.10 กฎที่ใช้อธิบายรายละเอียดร่องรอยไม่ดีผู้วิจัยได้วิเคราะห์รายละเอียดจาก Martin Fowler เมื่อได้กำหนดกฎที่ใช้อธิบายรายละเอียดเรียบร้อย ผู้วิจัยจึงมากำหนดเงื่อนไขกฎเพื่อการแปลความ ซึ่งใช้แนวความคิดในการตัดสินใจตามประสบการณ์ของผู้วิจัยสำหรับงานวิจัยนี้ได้กำหนดกฎการจำแนกด้วยฟัซซี่ไว้ดังตารางที่ 3.9, 3.10 และ 3.11

ตารางที่ 3.9 กฎการจำแนกซับเซตคลีนโค้ด

กฎลำดับที่ (Rule No)	**กฎที่ใช้อธิบายรายละเอียดของซับเซตคลีนโค้ด (Subset of Clean Code Type)**	**เงื่อนไขของกฎ (Rule Condition)**
1	Good Comment	(PCC is high)
2	Small Function	(NLOC is low) AND (NILI is low) AND (CC is low) AND (ILCC is low)
3	Do One Thing	(RC is moderate)
4	Nesting Depth	(ILND is less)
5	Function Arguments	(NOP is low)
6	Structured Programming	(ISS is more)
7	Class Organization	(NOC is less) AND (DIT is less) AND (NOIM is less) AND (NOI is less)
8	Encapsulation	(NPF is less)
9	Classes Should Be Small	(LCOM1 is low) AND (LCOM3 is low) AND (NOV is less) AND (NOO is less) AND (NFD is low) AND (NOM is low)
10	Maintain Cohesion	(LCOM1 is low) AND (LCOM3 is low)
11	Organizing for Change	(ACML is more) AND (ECML is less)
12	The Law of Demeter	(DMS is more)
13	Clean Boundary	(CBO is less)

ตารางที่ 3.10 กฎการจำแนกร่องรอยไม่ดี

กฎลำดับที่ (Rule No)	**กฎที่ใช้อธิบายรายละเอียดของร่องรอยไม่ดี (Bad Smell Type)**	**เงื่อนไขของกฎ (Rule Condition)**
1	Long Method	(NLOC is high) OR (NILI is high) OR (CC is high) OR (ILCC is high)
2	Large Class	(LCOM1 is high) OR (LCOM3 is high) OR (NFD is high) OR (NOM is high)
3	Long Parameter List	(NOP is high)
4	Temporary Field	(ACFL is less)
5	Feature Envy	(SAC-OMF is more)
6	Inappropriate Intimacy	(SEC-IM is more)
7	Lazy Class	(LMFC is more)
8	Bad Comment	(PCC is low)

ตารางที่ 3.11 กฎการจำแนกโค้ดที่มีความคลุมเครือ

กฎลำดับที่ (Rule No)	**กฎที่ใช้อธิบายรายละเอียดของความคลุมเครือ (Ambiguous Type)**	**เงื่อนไขของกฎ (Rule Condition)**
1	Ambiguity in Comment	(PCC is moderate)
2	Ambiguity in Method	(NLOC is moderate) AND (NILI is moderate) AND (CC is moderate) AND (ILCC is moderate)
3	Ambiguity in One Thing	(RC is low)
4		(RC is high)
5	Ambiguity in Nesting Depth	(ILND is more)
6	Ambiguity in Arguments	(NOP is moderate)
7	Ambiguity in Structured Programming	(ISS is less)

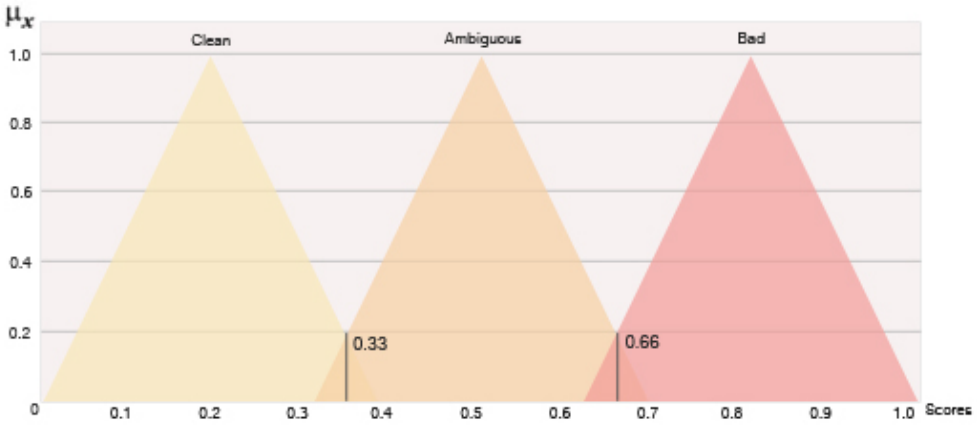
ตารางที่ 3.11 กฎการจำแนกโค้ดที่มีความคลุมเครือ (ต่อ)

กฎลำดับที่ (Rule No)	**กฎที่ใช้อธิบายรายละเอียดของความคลุมเครือ (Ambiguous Type)**	**เงื่อนไขของกฎ (Rule Condition)**
8	Ambiguity in Class Organization	(NOC is more) AND (DIT is more) AND (NOIM is more) AND (NOI is more)
9	Ambiguity in Encapsulation	(NPF is more)
10	Ambiguity in Classes	(LCOM1 is moderate) AND (LCOM3 is moderate) AND (NOV is more) AND (NOO is more) AND (NFD is moderate) AND (NOM is moderate)
11	Ambiguity in Cohesion	(LCOM1 is moderate) AND (LCOM3 is moderate)
12	Ambiguity for Change	(ACML is less) AND (ECML is more)
13	Ambiguity of Demeter	(DMS is less)
14	Ambiguity in Boundary	(CBO is more)

เมื่อได้กฎการจำแนกด้วยฟัซซี่แล้ว จะถูกนำไปใช้ในการขั้นตอนการถ่วงน้ำหนักเพื่อใช้ในการจำแนกซับเซตคลีนโค้ด ความคลุมเครือ และร่องรอยไม่ดี ซึ่งการถ่วงน้ำหนักจะเริ่มจากการกำหนดค่าหนึ่งค่า ด้วยการสุ่มค่า (Random) โดยค่าที่ใช้คือ 1 เพราะแทนค่าผลลัพธ์ที่เกิดขึ้นทั้งหมดจึงแบ่งค่าที่กำหนดออกเป็น 3 ส่วนเท่าๆ กัน เพราะต้องการให้มีผลลัพธ์ 3 ประเภท คือ

1) ตั้งแต่ 0 ถึง 0.33 กำหนดให้เป็นซับเซตคลีนโค้ด
2) ตั้งแต่ 0.34 ถึง 0.66 กำหนดให้เป็นความคลุมเครือ
3) ตั้งแต่ 0.67-1 กำหนดให้เป็นร่องรอยไม่ดี

จากนั้นจึงทำการการแปลงค่าให้อยู่ในรูปของการถ่วงน้ำหนักเพื่อหาผลลัพธ์ดังภาพที่ 3.6



ภาพที่ 3.6 ภาพกราฟแสดงเอาต์พุตเซตสำหรับการจำแนกด้วยฟัซซี่

แม้ว่าการนำมาตรวัดมาเป็นตัวกลางเพื่อใช้อธิบายความเป็นซับเซตคลีนโค้ด ความคลุมเครือและร่องรอยไม่ดีแล้วก็ตาม แต่งานวิทยานิพนธ์นี้ไม่ได้ยึดค่าที่ถูกกำหนดจากมาตรวัดนั้นๆ เพียงอย่างเดียว แต่นับรวมค่าพิจารณาใหม่ที่เกิดจากประสบการณ์ของกลุ่มผู้พัฒนาและผู้วิจัยกำหนดขึ้น โดยยึดหลักตามหนังสือซับเซตคลีนโค้ดของ Robert C. Martin ด้วย [9] ดังนั้นการเลือกค่าที่ใช้ในการพิจารณาเกิดจากประสบการณ์จากผู้เชี่ยวชาญเฉพาะทาง จึงสามารถนำมาผนวกรวมกับทฤษฎีฟัซซี่เซต ที่ต้องการองค์ความรู้ของผู้เชี่ยวชาญในการกำหนดขึ้น และเนื่องจากเซตที่สร้างนั้น เป็นเซตที่มี ขอบเขตแบบไม่ชัด เจน (Unclear Defined Boundary) จึงสามารถนำมาช่วยวิเคราะห์ให้ครอบคลุมในส่วนของความคลุมเครือ และซับเซตคลีนโค้ด โดยใช้หลักวิธีการถ่วงน้ำหนักของพื้นที่ใต้กราฟโดยการประมาณเทียบเคียงค่าจุดศูนย์ถ่วงโดยรวม (Center of Gravity: COG) เพื่อใช้ในการจำแนกโค้ดทั้งสองประเภทได้อย่างชัดเจน

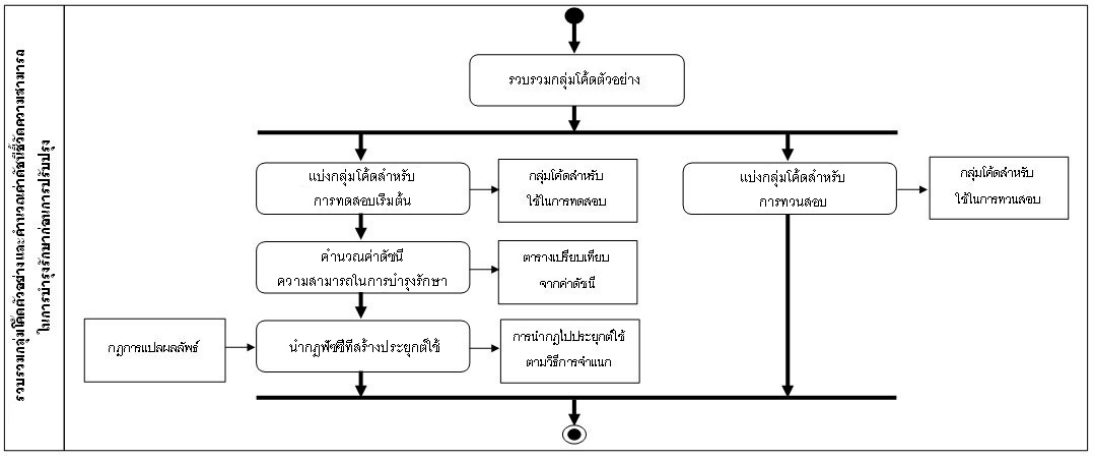
หลังจากขั้นตอนการถ่วงน้ำหนักซึ่งเป็นขั้นตอนย่อยสุดท้ายในการจำแนกตามกฎการจำแนกด้วยพัซซี่เสร็จเรียบร้อยแล้ว จะนำโค้ดที่ถูกจำแนกเป็น กลุ่มร่องรอยไม่ดี และความคลุมเครือมาทำการปรับปรุงแก้ไขโดยใช้เทคนิครีแฟคทอริ่ง ตามวิธีการปฏิบัติที่ได้การออกแบบไว้ ซึ่งจะอธิบายรายละเอียดในบทต่อไป

บทที่ 4
วิธีการจำแนกโค้ดและวิธีการปฏิบัติเพื่อปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี และโค้ดที่มีความคลุมเครือ โดยใช้เทคนิครีแฟคทอริ่ง

วิธีการจำแนกโค้ดและวิธีการปฏิบัติเพื่อปรับปรุงโค้ด เริ่มจากการรวบรวมกลุ่มตัวอย่างโค้ด เพื่อใช้ในการเริ่มทดสอบและการทวนสอบ ซึ่งจะทำการจำแนกและระบุโค้ดออกเป็น 3 กลุ่ม ได้แก่ กลุ่มร่องรอยไม่ดี กลุ่มโค้ดที่มีความคลุมเครือ และ กลุ่มซับเซตคลีนโค้ด โดยกลุ่มตัวอย่างที่ถูกจำแนกเป็นกลุ่มร่องรอยไม่ดีและกลุ่มโค้ดที่มีความคลุมเครือจะถูกแก้ไขโดยการเลือกวิธีการปรับปรุงตามวิธีการปฏิบัติที่กำหนดไว้ จากนั้นจึงนำกลุ่มโค้ดตัวอย่างผ่านขั้นตอนการวัดค่าดัชนีการบำรุงรักษาก่อนและหลังจากการปรับปรุงเพื่อทำการเปรียบเทียบผลลัพธ์เพื่อสรุปผลการทดสอบในขั้นตอนสุดท้าย

4.1 การรวบรวมชุดคำสั่งและนำกฎมาประยุกต์ใช้

รวบรวมกลุ่มโค้ดภาษา C# เพื่อใช้ในการทดสอบเริ่มต้นและใช้สำหรับการทวนสอบ จากนั้นนำกลุ่มโค้ดชุดเริ่มต้นมาทดสอบค่าดัชนีความสามารถบำรุงรักษาก่อนการปรับปรุง พร้อมทั้งนำกฎฟัซซี่ที่ถูกสร้างขึ้นตามขั้นตอนที่ 3.2.2 มาทำการประยุกต์ใช้งานเพื่อจำแนกโค้ดตามแต่ละประเภทที่กำหนดไว้ดังภาพที่ 4.1



ภาพที่ 4.1 รวบรวมกลุ่มโค้ดและคำนวณหาค่าดัชนีชี้วัดความสามารถในการบำรุงรักษาก่อนการปรับปรุง

4.1.1 การรวบรวมกลุ่มโค้ดสำหรับทดสอบ

กลุ่มโค้ดที่นำมาทดสอบ ผู้วิจัยรวบรวมจากแหล่งที่อ้างอิงต่างจากหนังสือที่ระบุไว้รวบรวมตามแหล่งอ้างอิงที่ได้จากงานวิจัยที่เกี่ยวข้อง และตามเว็บไซต์ต่างๆ ซึ่งกลุ่มตัวอย่างผ่านการแปลโปรแกรมโดยไม่มีข้อผิดพลาด โค้ดแต่ละชนิดจะต้องมีคุณสมบัติตามเกณฑ์ที่กำหนดไว้ในขั้นตอนที่ 3.1.1 และ 3.1.2 ซึ่งกลุ่มตัวอย่างผ่านการแปลโปรแกรมโดยไม่มีข้อผิดพลาด

4.1.2 การแบ่งกลุ่มโค้ดสำหรับการทดสอบเริ่มต้น

นำโค้ดภาษาC# ที่ได้รวบรวมแบ่งกลุ่มเป็นออกเป็น 2 ส่วน คือกลุ่มที่ 1 สำหรับใช้ในการทดสอบเริ่มต้นโดยมีวัตถุประสงค์หลักเพื่อทดสอบความสัมพันธ์ของซับเซตคลีนโค้ด โค้ดที่มีความคลุมเครือ และร่องรอยไม่ดี โดยแต่ละชุดจะประกอบด้วยกลุ่มโค้ด3 ชนิด ได้แก่ ชุดซับเซตคลีนโค้ด จำนวน 20 ชุดตัวอย่าง ชุดโค้ดที่มีความคลุมเครือจำนวน20 ชุดตัวอย่าง และ ชุดโค้ดร่องรอยที่ไม่ดี จำนวน 20 ชุดตัวอย่าง โดยมีประเภทร่อยรอยไม่ดีและความคลุมเครือครบตามขอบเขตที่กำหนด ดังตารางที่ 4.1

ตารางที่ 4.1 รายการกลุ่มโค้ดตัวอย่างชุดทดสอบเริ่มต้น

ร่องรอยไม่ดี	1. Converting Hexadecimal	11. LONG_PARAMETER_LISTS
	2. DirectoryInfoExtensions	12. Party Planner 2
	3. QueueExample1	13. PartialClassAddIn
	4. Assignment1MH_Base	14. Using Delegates
	5. EventedListExample	15. Hit_the_keys
	6. ImmutableCollections	16. List_of_Ducks
	7. FeatureEnvy	17. Sloppy_Joe
	8. Inappropriateintimacy	18. TallGuy
	9. LAZY_CLASS1	19. SimpleLinqToXml
	10. LAZY_CLASS2	20. Mixed_Messages

ตารางที่ 4.1 รายการกลุ่มโค้ดตัวอย่างชุดทดสอบเริ่มต้น (ต่อ)

โค้ดที่มีความคลุมเครือ	1. PartialClassInterfaces	11. OfficeSample
	2. TaxApp	12. LinqToNorthwind
	3. UnhandledThreadExcept	13. ObjectDumper
	4. UnhandledWPFException	14. PasteXmlAsLinq
	5. WebBrowser	15. PropertyBag
	6. Joe_and_Bob	16. Rss
	7. Mileage_calculator	17. SimpleLambdas
	8. Time_to_start_coding	18. WinFormsDataBinding
	9. BinaryWriter	19. XMLdoc
	10. LINQ to XML	20. Xquery
ซับเซตคลีนโค้ด	1. Encryption and Decryption	11. Fingers the Clown
	2. PrimeGenerator	12. PlanetMission
	3. WeekValidator	13. Baseball
	4. Set Collections	14. Go Fish
	5. OperatorOverloading	15. JewelThief
	6. ProtectedSnflar	16. Let's Build a House
	7. Generics_CSharp	17. TravellingSalesmanProblem
	8. Pool_Puzzle	18. ADO
	9. Playing Card	19. MutexFun
	10. Secret Ingredients	20. NamedPipes

4.1.3 การแบ่งกลุ่มโค้ดสำหรับการทวนสอบ

ชุดโค้ดภาษา C# กลุ่มที่ 2 คือ ชุดโค้ดที่ได้รวบรวมไว้ สำหรับการทวนสอบ โดยมีวัตถุประสงค์เพื่อใช้ยืนยันความถูกต้องของกฎการจำแนกที่สร้างขึ้น ซึ่งจะประกอบด้วยกลุ่มโค้ด 3 ชนิดคละรวมกันทั้งหมด 60 ชุด ดังตารางที่ 4.2

ตารางที่ 4.2 รายการกลุ่มโค้ดตัวอย่างชุดทวนสอบ

1. Contacts	19. Business Days
2. AnonymousDelegates	20. Familytree
3. Attributes	21. Breakfast for Lumberjacks
4. ConditionalMethods	22. Beehive Simulator
5. Delegates	23. ClassLeaf
6. EmployeeTracker.Common	24. COMInteropPart1
7. EmployeeTracker.Fakes	25. EmployeeTracker.Employee
8. Tokens	26. Generics2
9. Owner drawn	27. Pinvoke
10. Unsafe	28. Security
11. Reflector	29. LinqToXmlDataBinding
12. Swapping_elephants	30. SimpleLinqToObjects
13. CommandLine	31. Animations
14. EmployeeTracker.Model	32. ConsoleTCPServer
15. OleDbSample	33. Custom_IComparer
16. Properties	34. FrmFileCopier
17. Structs	35. CableBill
18. Threading	36. Familiar_math_symbol

ตารางที่ 4.2 รายการกลุ่มโค้ดตัวอย่างชุดทวนสอบ (ต่อ)

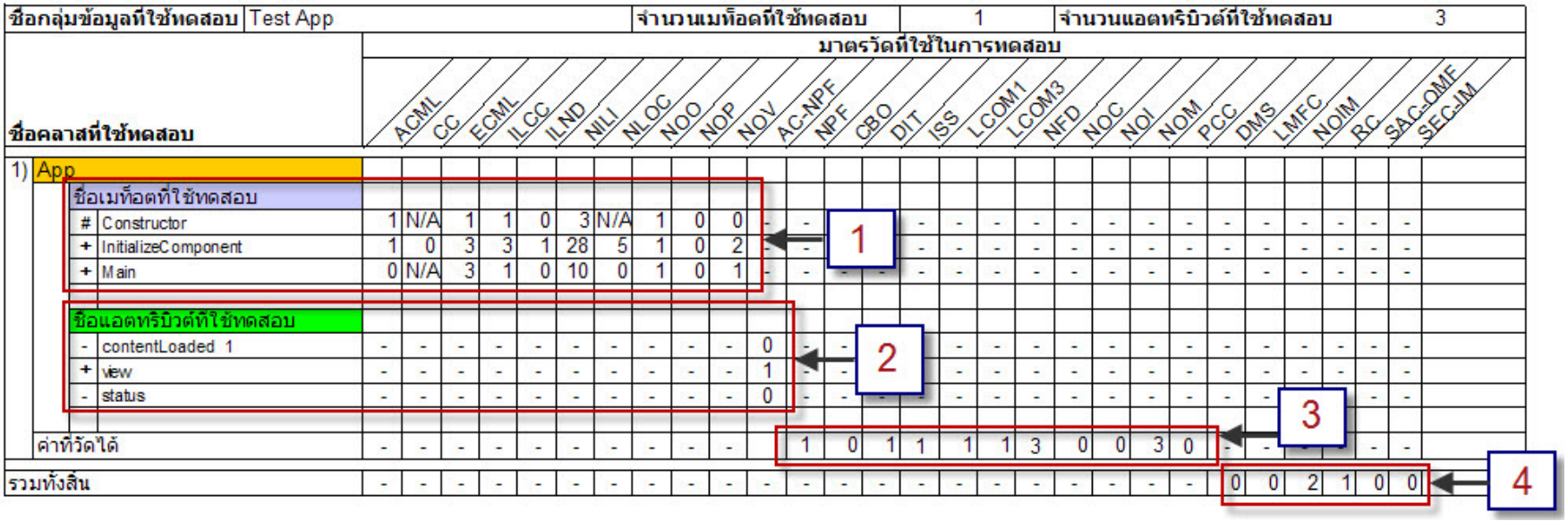
37. Talker_Tester	49. Libraries
38. Two_Decks	50. NamedAndOptional
39. BeeControl	51. Nullable
40. DataContractSerializer	52. PartialTypes
41. Equality	53. SimpleVariance
42. ExcuseManager	54. UserConversions
43. optional_parameters	55. Versioning
44. Simple_Text_Editor	56. Yield
45. Whack_a_mole	57. Image resizing
46. DeclareArraySample	58. FlashyThing
47. Events	59. TestApp
48. ExplicitInterface	60. ImageMapConverter

4.1.4 การทดสอบค่าดัชนีความสามารถบำรุงรักษา

นำกลุ่มโค้ดทั้ง 3 กลุ่มมาทำการคำนวณหาค่าดัชนีความสามารถบำรุงรักษาเพื่อให้ทราบค่าดัชนีความสามารถในการบำรุงรักษาของแต่ละกลุ่มโค้ดก่อนการปรับปรุงและนำค่าที่บันทึกได้มาทำการเปรียบเทียบหลังจากที่ทำการปรับปรุงเสร็จสิ้น โดยผลลัพธ์ค่าดัชนีความสามารถบำรุงรักษาที่วัดแสดงอยู่ในบทที่ 5

4.1.5 การวัดค่าด้วยมาตรวัดซอฟต์แวร์ที่กำหนดอินพุตมาตรวัดซอฟต์แวร์ที่กำหนดเอาต์พุตค่าที่วัดได้จากตัวอย่างทดสอบ

นำโค้ดที่ทดสอบมาผ่านมาตรวัดที่คัดเลือกในข้อ 3.1.3 ที่ปรากฏอยู่ในบทที่ 3 หน้าที่ 39 โดยจะแสดงแบ่งการทดสอบกลุ่มโค้ดตัวอย่างออกเป็นส่วนคลาส เมท็อด และแอตทริบิวต์ เพื่อนำไปใช้งานต่อได้อย่างสะดวก



ภาพที่ 4.2 ตัวอย่างการวัดค่าตามมาตรวัดซอฟต์แวร์ที่กำหนด

จากภาพที่ 4. 2 จะแสดงชื่อของกลุ่มตัวอย่างที่ทดสอบ จำนวนคลาส เมท็อด และแอตทริบิวต์ ที่ใช้ในการทดสอบ สังเกตว่าจะมีสัญลักษณ์ด้านหน้าเมท็อดและแอตทริบิวต์ 3 สัญลักษณ์ ได้แก่

1) สัญลักษณ์ + หมายความว่า เมท็อดและแอตทริบิวต์นั้นมีพับบลิก(Public)

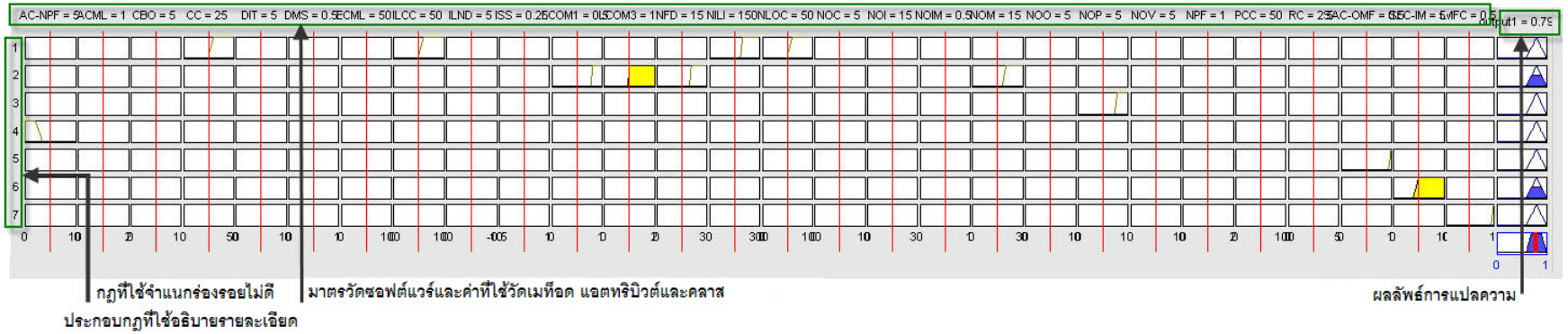
2) สัญลักษณ์ – หมายถึง เมท็อดและแอตทริบิวต์นั้น มีไพรเวท(Private)

3) สัญลักษณ์ # หมายถึง เมท็อดและแอตทริบิวต์นั้นถูกโอเวอร์ไรด์(Override)

เมื่อนำกลุ่มตัวอย่างโค้ดมาผ่านการทดสอบด้วยมาตรวัดซอฟต์แวร์ ซึ่งต้องทดสอบทุกส่วนของโค้ด ซึ่งหมายเลข 1 คือ การวัดค่าเมท็อดที่อยู่ในคลาส ส่วนหมายเลข 2 คือ การวัดค่าแอตทริบิวต์ที่อยู่ในคลาส หมายเลข 3 คือ การวัดค่าการทำงานของคลาส และหมายเลข 4 คือ การวัดการทำงานทั้งหมดของโค้ดตัวอย่างที่นำมาทดสอบ เพื่อนำเอาค่าผลลัพธ์ที่ได้ไปใช้ในขั้นตอนการแปลความด้วยฟัซซี่ในขั้นตอนต่อไป

4.1.6 การประยุกต์ใช้กฎการจำแนกฟัซซี่ในการจำแนก

ขั้นตอนนี้เริ่มจากการนำโค้ดที่ทำการทดสอบมาผ่านการประยุกต์ใช้งานของกฎฟัซซี่ที่สร้างไว้ เริ่มจากนำกลุ่มโค้ดที่ใช้งานมาแบ่งออกเป็นคลาส เมท็อด และแอตทริบิวต์ มาทำการประยุกต์ใช้ด้วยกฎการจำแนก ซึ่งกฎการจำแนกที่สร้างขึ้นทั้งสามประเภทได้แก่ กฎที่ใช้จำแนกซับเซตคลีนโค้ด กฎที่ใช้จำแนกโค้ดที่มีความคลุมเครือ และกฎที่ใช้จำแนกร่องรอยไม่ดีมาประยุกต์ใช้งาน ดังตัวอย่างต่อไปนี้



ภาพที่ 4.3 ตัวอย่างการทำ Defuzzification

ดังภาพที่ 4.3 แสดงการนำค่าที่วัดในแต่ละเมท็อด แอตทริบิวต์ และคลาสมาทำการถ่วงน้ำหนักเพื่อทำการแปลความเพื่อแสดง ชนิดของโค้ดว่า เป็นร่องรอยไม่ดี โค้ดที่มีความคลุมเครือ หรือเป็นซับเซตคลีนโค้ด ซึ่งทำการสรุปของเป็นดังตารางที่ 4.3

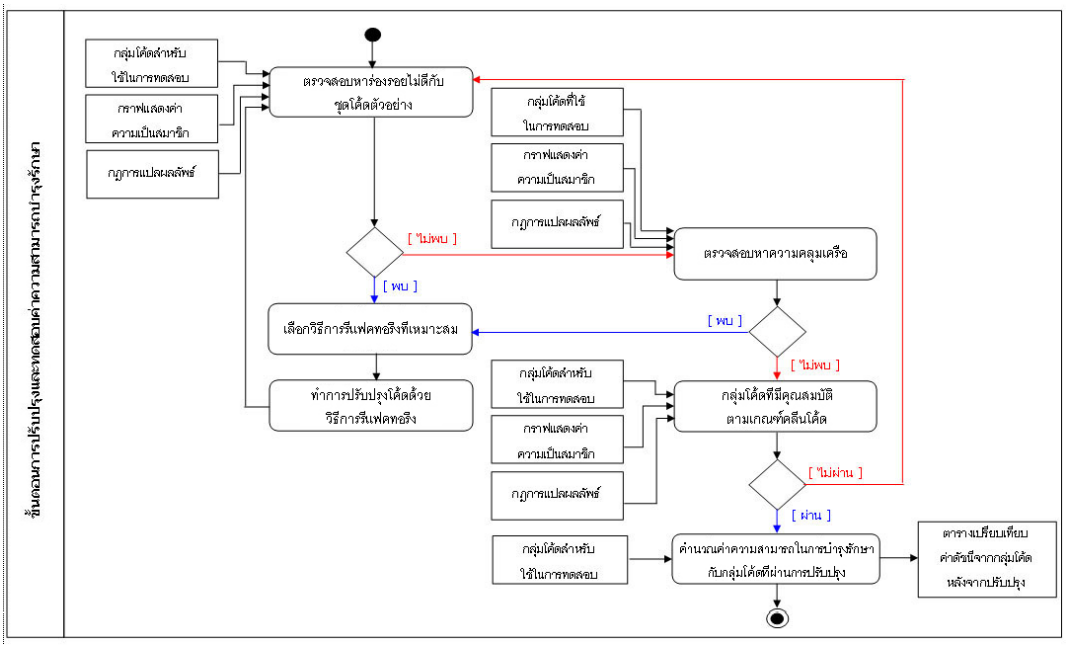
ตารางที่ 4.3 การแปลความกฎฟัซซี่ที่ใช้จำแนกกลุ่มโค้ด

ชุดรายการ				Defuzzification Output	สรุปค่าแปลความ
ลำดับที่	คลาส	เมท็อด	แอตทริบิวต์		
1	App	Constructor	contentLoaded 1	0.237	Clean
2		InitializeComponent	status	0.319	Clean
3		Main		0.279	Clean
4		Constructor	view	0.799	Bad
5		InitializeComponent		0.799	Bad
6		Main		0.799	Bad

จากตารางที่ 4. 3 เป็นการทดสอบทุกส่วนประกอบ คลาส เมท็อด และ แอตทริบิวต์ ในส่วนของแอตทริบิวต์นั้นมีค่าที่วัดได้เพียงสองค่าเท่านั้นคือ 0 และ 1 เพื่อไม่ให้เกิดความซ้ำซ้อนของตารางการทดสอบ จึงยุบการทดสอบ แอตทริบิวต์ที่มีค่า 0 ไว้ด้วยกัน ซึ่งขอจัดแอตทริบิวต์ที่มีค่าเหมือนกันมาแสดงในคอลัมน์ในเดียวกันและผลลัพธ์ของการยุบรวมแอตทริบิวต์ ในตารางนี้ไม่เกิดผลกระทบต่อการประยุกต์ใช้งานกฎการจำแนกแต่อย่างใด การแปลความจะ ประกอบด้วยคอลัมน์ลำดับที่เพื่อแสดงจำนวนครั้งในการทดสอบ โดยจำนวนลำดับการทดสอบทั้งหมดคือค่าที่เป็นไปได้ของ คลาส เมท็อด และ แอตทริบิวต์ เช่น การทดสอบครั้งที่ 1-3 จะทดสอบคลาส เมท็อด และ แอตทริบิวต์ ในกลุ่มนี้จะทดสอบค่าแอตทริบิวต์ที่มีค่า 0 และการทดสอบครั้งที่ 4-6 จะทดสอบค่าแอตทริบิวต์ที่มีค่า 1 และในคอลัมน์ท้ายสุดเป็นการ สรุปค่าแปลความ ในตัวอย่างนี้ตรวจพบร่องรอยไม่ดี จึงต้องนำไปปรับปรุงด้วยเทคนิครีแฟคทอริ่งต่อไป

4.2 การปรับปรุงโค้ดด้วยเทคนิครีแฟคทอริ่ง

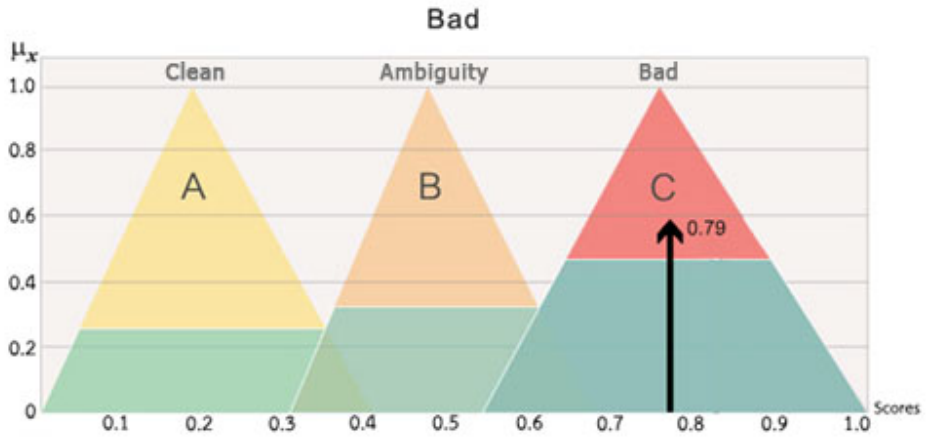
จุดมุ่งหมายการปรับปรุงโค้ดเพื่อให้มีคุณสมบัติตามเกณฑ์คุณภาพของซับเซตคลีนโค้ดซึ่งในที่จะทำการตรวจสอบชนิดของร่องรอยไม่ดีเพื่อคัดเลือกเทคนิคการปรับปรุงที่เหมาะสมให้กับโค้ดกลุ่มนั้น โดยผลลัพธ์จากขั้นตอนที่ได้คือ โค้ดที่ผ่านเกณฑ์การพิจารณาและมีคุณสมบัติที่เป็นซับเซตคลีนโค้ดเท่านั้น ดังภาพที่ 4.4



ภาพที่ 4.4 การปรับปรุงโค้ดให้เป็นไปตามเกณฑ์ด้วยเทคนิครีแฟคทอริง

4.2.1 การตรวจสอบหาร่องรอยไม่ดีกับชุดคำสั่ง

ในขั้นตอนนี้เริ่มด้วยการนำโค้ดที่ได้มาทำวิเคราะห์ตามมาตรวัดที่ได้คัดเลือกเพื่อนำค่าที่ได้ผ่านกระบวนการจำแนกด้วยฟัซซี่ สุดท้ายจึงทำการแปลผลลัพธ์ตามกฎที่สร้างขึ้น โดยเลือกเฉพาะกฎการแปลความที่เกี่ยวกับร่องรอยไม่ดีเท่านั้นเพื่อหาค่าถ่วงน้ำหนัก กรณีที่ตรวจพบว่ามีกลุ่มร่องรอยไม่ดี ค่าถ่วงน้ำหนักจะแสดงผลตามตัวอย่างภาพที่ 4.5



ภาพที่ 4.5 ค่าถ่วงน้ำหนักของกรณีที่ตรวจพบร่องรอยไม่ดีในกลุ่มโค้ด

จากภาพจะเห็นว่าค่าถ่วงน้ำหนักอยู่ที่ 0.79 ซึ่งเทน้ำหนักไปทางด้านขวาซึ่งค่าถ่วงน้ำหนักอยู่ในช่วงของร่องรอยไม่ดี คือมีค่ามากกว่า 0.67 ขึ้นไป จึงสามารถวิเคราะห์ผลได้ว่า กลุ่มโค้ดนั้นมีร่องรอยไม่ดีปรากฏอยู่ ซึ่งประเภทของร่องรอยไม่ดีที่พบนั้น สามารถตรวจสอบได้จากกฎในการอธิบาย หากกฎการอธิบายที่ตรวจสอบร่องรอยไม่ดีข้อใดมีค่าเป็นจริง จะถือว่าโค้ดที่ทดสอบมีการตรวจพบร่องรอยไม่ดีประเภทนั้นและจะถูกปรับปรุงด้วยวิธีการในขั้นตอนที่ 4.2.2 แต่หากผลลัพธ์ที่ได้ตรวจสอบร่องรอยไม่ดีอีก ก็จะกลับไปทำการแก้ไขเพิ่มเติมในขั้นตอน 4.2.1 อีกครั้ง

4.2.2 การเลือกวิธีการรีแฟคทอริ่งที่เหมาะสม

กรณีที่เป็นกลุ่มคำสั่งที่ตรวจพบร่องรอยไม่ดี ในขั้นตอนนี้ผู้วิจัยได้รวบรวมวิธีรีแฟคทอริ่งของ Martin Fowler [6] เพื่อสร้างวิธีการปฏิบัติสำหรับปรับปรุงโค้ดให้เหมาะสมกับชนิดและประเภทของร่องรอยไม่ดีนั้นๆ และความคลุมเครือที่เกิดขึ้นดังนั้น จึงขอยกตัวอย่างประกอบดังตารางที่ 4.4

ตารางที่ 4.4 แนวทางการเลือกวิธีการปรับปรุงโค้ด

<table>
<tr><td colspan="3">ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ</td><td colspan="3">Test App</td></tr>
<tr><td>ลำดับที่</td><td>คลาส</td><td>เมท็อด</td><td>แอตทริบิวต์</td><td>ผลลัพธ์การแปลความ</td><td>สรุปค่าแปลความ</td></tr>
<tr><td>1</td><td rowspan="3">App</td><td>Constructor</td><td>view</td><td>0.799</td><td>Bad</td></tr>
<tr><td>2</td><td>InitializeComponent</td><td rowspan="2"></td><td>0.799</td><td>Bad</td></tr>
<tr><td>3</td><td>Main</td><td>0.799</td><td>Bad</td></tr>
<tr><td colspan="3">ประเภทร่องรอยไม่ดีที่ต้องการแก้ไข</td><td colspan="3">Temporary Field</td></tr>
<tr><td colspan="6">วิธีการรีแฟคทอริ่งสำหรับการแก้ไขร่องรอยไม่ดี</td></tr>
<tr><td colspan="6">1. Moving Features between Objects วิธีย่อยที่ใช้ Extract Class
2. Simplifying Conditional Expression วิธีย่อยที่ใช้ Introduce Null Object</td></tr>
<tr><td colspan="3">วิธีที่เลือกใช้ในการปรับปรุงร่องรอยไม่ดี</td><td colspan="3">Simplifying Conditional Expression วิธีย่อยที่ใช้ Introduce Null Object</td></tr>
<tr><td colspan="6">เหตุผลที่เลือกใช้</td></tr>
<tr><td colspan="6">เมื่อทำการตรวจสอบโปรแกรมTestApp พบว่ามีร่องรอยไม่ดีประเภทTemporary Field ซึ่งเกิดจากแอตทริบิวต์view ไม่ได้ถูกกำหนดค่าเริ่มต้นในการทำงาน เพื่อแก้ไขร่องรอยไม่ดีนี้จึงเลือกใช้</td></tr>
</table>

ตารางที่ 4.4 แนวทางการเลือกวิธีการปรับปรุงโค้ด (ต่อ)

เหตุผลที่เลือกใช้
Introduce Null Object ให้เป็นกำหนด ค่าเริ่มต้นแก่คลาส App และแก้ไขเมท็อดตรวจสอบค่าแอตทริบิวต์ view ที่ก่อนจะที่จะเริ่มการทำงานในแต่ละส่วน ทั้งนี้ไม่สามารถใช้วิธีแก้ไข Extract Class เนื่องจากสาเหตุของปัญหาไม่เกิดจากเมท็อดและแอตทริบิวต์ ทำงานไม่สอดคล้องภายในคลาส หรือไม่ได้เกิดจากการประกาศแอตทริบิวต์ในการใช้งานที่ไม่เหมาะสม

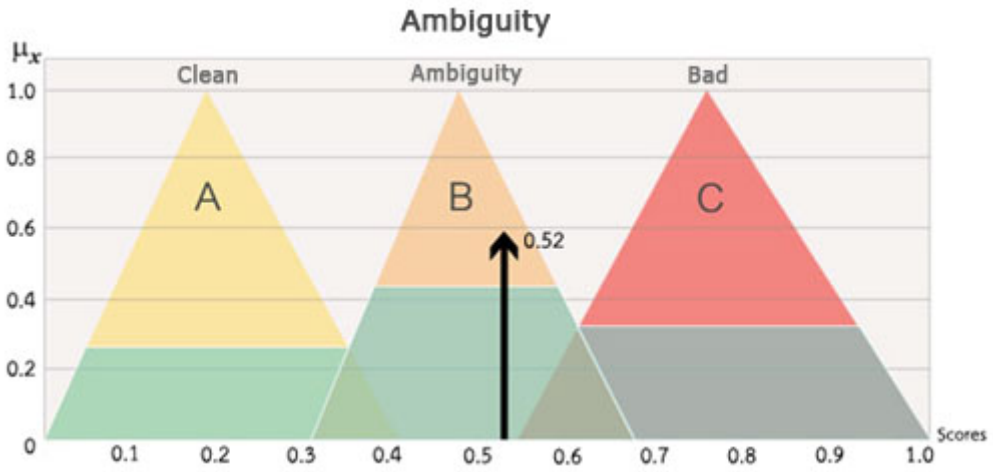
จากตารางที่ 4.4 แสดงให้เห็นถึงประเภทร่องรอยไม่ดีที่ตรวจพบและวิธีการปฏิบัติเพื่อแก้ไขร่องรอยไม่ดีตามรายละเอียดในภาคผนวก ง พร้อมทั้งอธิบายรายละเอียดวิธีการแก้ไขและรายละเอียดที่ใช้สนับสนุนการตัดสินใจเลือกวิธีปฏิบัติที่รวบรวมมาหรือวิธีปฏิบัติของผู้เชี่ยวชาญด้านโปรแกรมเพื่อแก้ไขร่องรอยไม่ดีที่ตรวจพบ แต่ในกรณีที่ตรวจสอบพบกลุ่มโค้ดที่มีความคลุมเครือ ในขั้นตอนนี้ การเลือกวิธีการแก้ไขให้ถือเป็นวิจารณญาณของผู้เชี่ยวชาญด้านโปรแกรมจะเป็นผู้เลือกวิธีที่ใช้ในการปรับปรุงให้แก่กลุ่มโค้ดที่มีความคลุมเครือ

4.2.3 การปรับปรุงโค้ดด้วยการรีแฟคทอริ่ง

ในขั้นตอนนี้คือ การปรับปรุงตามวิธีการที่ได้เลือกไว้ในขั้นตอน 4.2.2 โดยการแก้ไขปรับปรุงนั้นจะไม่มี ผลกระทบกับผู้ใช้งานแต่อย่างใด เมื่อทำการแก้ไขเสร็จตามกระบวนแล้วให้กลับไปตรวจสอบร่องรอยไม่ดีอีกครั้งในขั้นตอน 4.2.1 เพื่อให้มั่นใจว่าไม่มีร่องรอยไม่ดีปรากฏในกลุ่มโค้ดที่ทำการปรับปรุง

4.2.4 การตรวจสอบหาความคลุมเครือ

เมื่อไม่พบร่องรอยไม่ดีในกลุ่มโค้ด ขั้นตอนต่อไปจะเป็นการตรวจสอบหาความคลุมเครือในกลุ่มโค้ด โดยนำกลุ่มโค้ดที่ได้มาทำการวัดตามมาตรวัดที่ได้คัดเลือกเพื่อนำค่าที่วัดได้ไปผ่านกระบวนการจำแนกด้วยฟัซซี่เช่นเดียวกับขั้นตอนการตรวจสอบร่องรอยไม่ดี ในขั้นตอนนี้จะเลือกใช้กฎการแปลความคลุมเครือควบคู่กับกฎการแปลความซับเซตคลีนโค้ดเพื่อนำค่าถ่วงน้ำหนักที่ได้ไปใช้ในการแปลผลลัพธ์ตามกฎที่สร้างขึ้น กรณีที่ตรวจสอบพบความคลุมเครือ ค่าถ่วงน้ำหนักจะแสดงผลตามตัวอย่างภาพที่ 4.6



ภาพที่ 4.6 แสดงค่าถ่วงน้ำหนักของกรณีที่ตรวจพบความคลุมเครือในกลุ่มโค้ด

หลังจากที่ทำการแก้ไขร่องรอยไม่ดีให้หมดไปจากกลุ่มโค้ดแล้ว ได้ทำการตรวจสอบค่าถ่วงน้ำหนักใหม่อีกครั้งกลับกลุ่มโค้ดตามกฎการตรวจสอบที่ได้สร้างขึ้นมา โดยผลลัพธ์ในครั้งนี้ค่าถ่วงน้ำหนักเท่ากับ 0.52 ซึ่งค่าถ่วงน้ำหนักอยู่บริเวณกึ่งกลาง วิเคราะห์ตามค่าถ่วงน้ำหนักที่อยู่ในช่วงตั้งแต่ 0.34 จนถึง 0.66 แสดงให้เห็นว่ากลุ่มโค้ดมีความคลุมเครืออยู่ซึ่งจำเป็นต้องปรับปรุงเพิ่มคุณภาพให้แก่กลุ่มโค้ดที่ทำการตรวจสอบได้ โดยวิเคราะห์หาสาเหตุเพิ่มเติมจากกฎการอธิบายที่สร้างขึ้น เพื่อนำไปเป็นข้อมูลในการตัดสินใจเลือกวิธีการปรับปรุงเพิ่มเติม แต่ถ้าผ่านเกณฑ์นี้จะไปทำการตรวจสอบเพื่อยืนยันความเป็นซับเซตคลีนโค้ดในขั้นตอนต่อไป

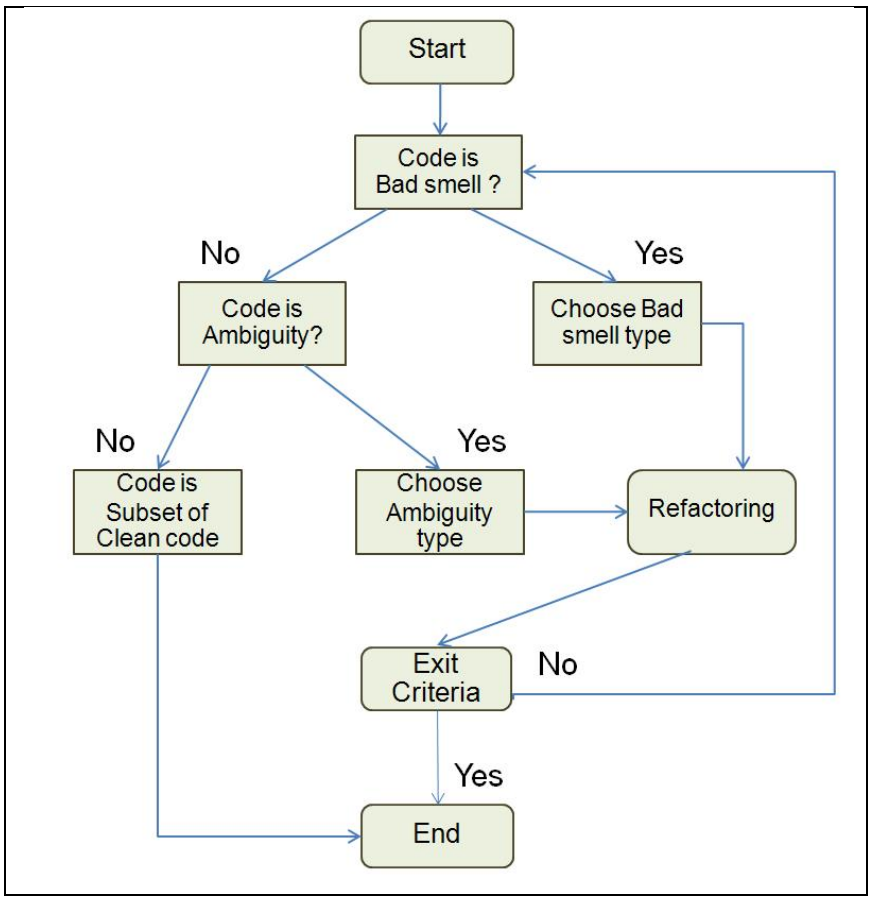
4.2.5 กลุ่มโค้ดมีคุณสมบัติตามเกณฑ์ของซับเซตคลีนโค้ด

เป็นขั้นตอนสำคัญในการตรวจสอบเพื่อยืนยันคุณสมบัติความเป็นซับเซตคลีนโค้ด ซึ่งกระบวนการนี้ไม่ได้มีความแตกต่างจากขั้นตอนการตรวจสอบความคลุมเครือและร่องรอยไม่ดีมากนัก เริ่มจากแปลความตามกฎที่ได้สร้างขึ้น โดยค่าของผลลัพธ์ของความเป็นซับเซตคลีนโค้ดซึ่งจุดนี้จะไม่อนุญาตให้มีค่าร่องรอยไม่ดีปรากฏเลย แต่อนุญาตให้มีความคลุมเครือได้บ้างเล็กน้อย ตามกฎของการแปลความที่ได้สร้างขึ้น โดยค่าถ่วงน้ำหนักของซับเซตคลีนโค้ดนั้นจะแสดงผลตามตัวอย่างภาพที่ 4.7

ภาพที่ 4.7 แสดงค่าถ่วงน้ำหนักของกรณีที่ตรวจพบความซับเซตคลีนโค้ดในกลุ่มโค้ด

จากภาพจะเห็นว่าค่าถ่วงน้ำหนักมีค่า 0.17 ซึ่งเทน้ำหนักไปทางด้านซ้าย วิเคราะห์ตามค่าถ่วงน้ำหนักน้อยกว่า 0.33 แสดงว่า กลุ่มโค้ดนั้นมีคุณสมบัติความเป็นซับเซตคลีนโค้ดตรงตามเกณฑ์ที่ได้สร้างไว้ แต่ในกรณีที่ค่าถ่วงน้ำหนักได้ผลลัพธ์ที่ไม่ได้อยู่ในเกณฑ์ที่กำหนด จำเป็นต้องกลับไปตรวจสอบใหม่ว่ายังเป็นกลุ่มร่องรอยไม่ดีหรือกลุ่มโค้ดที่มีความคลุมเครือหรือไม่ โดยกลับไปที่ขั้นตอนที่ 4.2.1 อีกครั้งเพื่อตรวจสอบและยืนยันว่า ได้ทำการแก้ไขปรับปรุงร่องรอยไม่ดีและความคลุมเครือออกจากข้อมูลเรียบร้อย

ดังนั้นผู้วิจัยขอนำเสนอขั้นตอนการตัดสินใจเพื่อเลือกแก้ไขประเภทของร่องรอยไม่ดีและความคลุมเครือที่เกิดขึ้น พร้อมทั้งแสดงจุดมุ่งหมายในการแก้ไขร่องรอยไม่ดีและความคลุมเครือในแต่ละกรณีที่เกิดขึ้นโดยมีขั้นตอนการตัดสินใจดังภาพที่ 4.8



ภาพที่ 4.8 รูปต้นไม้การตัดสินใจ (Decision tree) เลือกปรับปรุงโค้ดร่องรอยไม่ดี โค้ดที่มีความคลุมเครือ โค้ดที่เป็นซับเซตคลีนโค้ดและกรณีจบการทำงาน

จากภาพที่ 4. 8 แสดงถึงขั้นตอนการปรับปรุงโค้ดโดยแบ่งตามผลลัพธ์ของการจำแนก ในกรณีพบร่องรอยไม่ดีจะทำการตรวจสอบเพิ่มเติมถึงจำนวนร่องรอยไม่ดีที่ตรวจพบ เพื่อคัดเลือกแก้ไขร่องรอยไม่ดีโดยคำนึงถึงผลกระทบที่จะเกิดขึ้น จากนั้นตรวจสอบซ้ำเพื่อหาร่องรอยไม่ดีที่เกิดขึ้นหลังทำการแก้ไขจนกระทั่งไม่พบร่องรอยไม่ดีในโค้ด ส่วนกรณีที่ตรวจสอบพบโค้ดที่มีความคลุมเครือจะคำนวณค่าผลกระทบในการแก้ไขความคลุมเครือในแต่ละรายการ โดยเลือกรายการที่ต้องการแก้ไขที่ใช้ค่าความพยายามน้อยที่สุดมาแก้ไขความคลุมเครือก่อน แล้วจึงตรวจสอบซ้ำจนกระทั่งแก้ไขความคลุมเครือจนหมด ส่วนในกรณีที่จำแนกเป็นซับเซตคลีนโค้ดจะไม่มีการปรับปรุง เช่นเดียวกับกรณีจบการทำงานซึ่งมีความแตกต่างที่ผลลัพธ์ของการจบการทำงาน เพราะไม่สามารถเป็นซับเซตคลีนโค้ดตามวิธีการที่งานวิจัยนี้ได้ออกแบบไว้

เพื่อแสดงรายละเอียดของวิธีการที่นำเสนอ ผู้วิจัยขออธิบายขั้นตอนการปรับปรุงโค้ดด้วยขั้นตอนวิธีกระบวนการปรับปรุงโค้ด โดยมีรายละเอียดการทำงานดังต่อไปนี้

```
PROCEDURE ImproveSourceCodeToBeSubSetofCleanCode (sourcecode)
INPUT: sourcecode คือ ตัวแปรโค้ดที่ถูกนำเข้าเพื่อทำทดสอบตามวิธีการที่นำเสนอ
OUTPUT: sourcecode คือ ตัวแปรโค้ดที่ถูกส่งคืนหลังจากแก้ไขตามวิธีที่นำเสนอ
BEGIN
    REPEAT
        COMMENT: codetype คือ ตัวแปรที่รับค่าประเภทของโค้ดที่ถูกจำแนกได้
        codetype ← ClassifySourceCode(sourcecode);
        IF (codetype is Bad Smell) THEN
                sourcecode ← FixTheBadSmell(sourcecode);
        ELSE IF (codetype is Ambiguous Code) THEN
                sourcecode ← FixTheAmbiguous(sourcecode);
        END IF
    COMMENT:  Exit_Critiria คือ เงื่อนไขการจบการทำงานของโปรแกรม ซึ่งกำหนด
    วิจารณญาณของผู้เชี่ยวชาญเป็นผู้กำหนดการจบการทำงาน
    UNTIL (codetype is Clean Code OR Exit_Critiria is TRUE)
    return {sourcecode}
END
```

ภาพที่ 4.9 Procedure ImproveSourceCodeToBeSubSetofCleanCode เพื่อปรับปรุงโค้ดให้มีคุณสมบัติเป็นซับเซตคลีนโค้ด

1) ขั้นตอนการทำงานของImproveSourceCodeToBeSubSetofCleanCode

ในขั้นตอนนี้เป็นภาพรวมของวิธีการปรับปรุงโค้ดให้มีคุณสมบัติเป็นซับเซตคลีนโค้ดตามวิธีการที่นำเสนอไว้ โดยเริ่มจากตรวจสอบชนิดของโค้ดด้วยกฎการแปลความฟัซซี่ในขั้นตอน ClassifySourceCode จากผลลัพธ์ที่ได้จะแบ่งชนิดของโค้ดแบ่งออก 3 กรณีได้แก่

1.1) กรณีที่ตรวจสอบพบร่องรอยไม่ดี จะทำการแก้ไขร่องรอยไม่ดีตามวิธีที่นำเสนอไว้ในขั้นตอน FixTheBadSmell เมื่อแก้ไขร่องรอยไม่ดีแล้วอาจตรวจพบความคลุมเครือเกิดขึ้นตามมา ซึ่งจะถูกแก้ไขในขั้นตอนถัดไป

1.2) กรณี ตรวจสอบพบคลุมเครือ จะทำการแก้ไขความคลุมเครือตามวิธีที่นำเสนอไว้ในขั้นตอนFixTheAmbiguous โดยจะทำการปรับปรุงจนกระทั้งมีคุณสมบัติเป็นซับเซตคลีนโค้ดตามวิธีการที่ได้นำเสนอ

1.3) กรณีที่เป็นซับเซตคลีนโค้ด ตามวิธีการที่นำเสนอไว้จะไม่มีการปรับปรุงโค้ดแต่อย่างใด ซึ่งโค้ดที่ถูกระบุว่าเป็นซับเซตคลีนโค้ดจะมีคุณสมบัติตามวิธีการที่ได้กำหนดไว้ถือว่าสิ้นสุดตามวิธีการ

หลังจากที่ทำการทดลองแก้ไขกับร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือในชุดตัวอย่างเริ่มต้นด้วยวิธีการปฏิบัติตามที่ได้ออกแบบไว้ พบว่าสามารถแก้ไขชุดตัวอย่างร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือทั้งหมดให้เป็นซับเซตคลีนโค้ดได้ ดังนั้นเงื่อนไขในการจบการทำงานคือ แก้ไขร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือที่ตรวจพบจนกระทั่งเป็นซับเซตคลีนโค้ด เพื่อให้ ครอบคลุมกรณีที่คาดไม่ถึงหากนำไปประยุกต์ใช้งานในอนาคต ผู้วิจัยได้ กำหนด เงื่อนไขจบการทำงานให้เป็นดุลยพินิจของผู้ทำการแก้ไขปรับปรุงโค้ดเป็นผู้เลือกที่จะหยุดหรือสิ้นสุดการทำงาน ในกรณีนี้โค้ดที่ถูกปรับปรุงจะไม่มีคุณสมบัติเป็นซับเซตคลีนโค้ดตามที่วิธีการของงานวิจัยนี้ได้ออกแบบไว้

```
PROCEDURE FixTheBadSmell (sourcecode)
INPUT: sourcecode คือ ตัวแปรโค้ดที่ถูกนำเข้าเนื่องจากตรวจพบร่องรอยไม่ดี
OUTPUT: sourcecode คือ ตัวแปรโค้ดที่ถูกหลังจากแก้ไขร่องรอยไม่ดีเสร็จแล้วเรียบร้อยแล้ว
BEGIN
    REPEAT
        COMMENT: listofbadsmell คือ ตัวแปรรับค่าอาเรย์ประเภทของร่องรอยไม่ดีที่ตรวจพบ
```

ภาพที่ 4.10 Procedure FixTheBadSmell เพื่อแก้ไขร่องรอยไม่ดีที่เกิดขึ้น
โดยพิจารณาตามระดับผลกระทบ

```
        listofbadsmell ← GetTheBadSmellType(sourcecode);
        numberofbadsmell ← Length(listofbadsmell);
        IF (numberofBadSmell > 1) THEN
            COMMENT: badsmell คือ ตัวแปรรับค่าประเภทของร่องรอยไม่ดีที่ถูกแก้ไข
            badsmell ← ChooseTheBadSmell(listofbadsmell, numberofbadsmell);
            COMMENT: techniques คือ ตัวแปรที่รับค่าอาเรย์ของวิธีปฏิบัติที่ถูกรวบรวมมา
            techniques ← GetRefactoringTechnique(listofbadsmell,
            numberofbadsmell);
        ELSE
            badsmell ← listofbadsmell[0];
            techniques ← GetRefactoringTechniquefromTable(badsmell);
        END IF
            COMMENT: technique คือ ตัวแปรที่รับค่าวิธีปฏิบัติที่ถูกเลือกใช้ในการแก้ไข
    ร่องรอยไม่ดี
            technique ← ChooseRefactoringTechniqueByHeuristic(sourcecode,
            badsmell, techniques);
            sourcecode ← DoRefactoring(sourcecode, technique);
            codetype ← ClassifySourceCode(sourcecode);
    UNTIL (codetype is not Bad Smell OR Exit_Critiria is TRUE)
    return {sourcecode}
END
```

ภาพที่ 4.10 Procedure FixTheBadSmell เพื่อแก้ไขร่องรอยไม่ดีที่เกิดขึ้น

โดยพิจารณาตามระดับผลกระทบ(ต่อ)

2) ขั้นตอนการทำงานของFixTheBadSmell

เมื่อตรวจพบร่องรอยไม่ดีที่เกิดขึ้นในโค้ดที่ผ่านการทดสอบ ในขั้นตอนนี้คือวิธีการแก้ไขร่องรอยไม่ดีที่เกิดขึ้น เริ่มจากระบุประเภทร่องรอยไม่ดีที่เกิดขึ้น การแก้ไขร่องรอยไม่ดีจะพิจารณาตามระดับผลกระทบที่เกิดขึ้นแล้วจึงทำการแก้ไขจนกระทั่งร่องรอยไม่ดีถูกแก้ไขจนหมดหรือเงื่อนไขจบการทำงานที่กำหนดไว้เป็นจริง โดยสามารถอธิบายลำดับขั้นตอนย่อยดังต่อไปนี้

2.1) ขั้นตอนการตรวจสอบชนิดของร่องรอยไม่ดีที่เกิดขึ้นโดยนำโค้ดที่ทดสอบมาตรวจสอบในขั้นตอนGetTheBadSmellType ซึ่งเป็นการนำกฎการจำแนกด้วยฟัซซี่ประเภทร่องรอยไม่ดีมาใช้ในการตรวจสอบ หากกฎในแต่ละข้อที่ใช้อธิบายข้อใดมีค่าเป็นจริง แสดงถึงตรวจพบร่องรอยไม่ดีประเภทนั้นเกิดขึ้นในโค้ดที่ผ่านทดสอบ

2.2) เมื่อตรวจสอบชนิดของร่องรอยไม่ดีในขั้นตอนนี้จะเป็นการพิจารณาเพื่อเลือกแก้ไขร่องรอยไม่ดีโดยจะพิจารณาตามระดับของผลกระทบจากร่องรอยไม่ดีแต่ละประเภทสามารถแบ่งออกได้เป็น2 กรณีได้แก่

(1) ตรวจพบร่องรองไม่ดีมากกว่าหนึ่งประเภท ซึ่งจะทำการคัดเลือกร่องรอยไม่ดีเพื่อแก้ไขในขั้นตอนChooseTheBadSmell โดยเลือกร่องรอยไม่ดีที่มีผลกระทบน้อยที่สุดในการแก้ไข แล้วจึงรวบรวมวิธีการปฏิบัติในขั้นตอน GetRefactoringTechnique เพื่อแก้ไขร่องรอยไม่ดีในแต่ละประเภทที่ตรวจพบและสามารถเพิ่มวิธีการปฏิบัติตามที่ผู้เชี่ยวชาญแนะนำเพิ่มเติมได้เพื่อนำไปใช้แก้ไขในขั้นตอนต่อไป

(2) ตรวจพบร่องรองไม่ดีเพียงชนิดเดียว ตามวิธีการที่นำเสนอจะทำการแก้ไขร่องรอยไม่ดีที่เกิดขึ้น และนำวิธีการปฏิบัติเพื่อแก้ไขร่องรอยไม่ดีตามตารางที่ระบุไว้ในขั้นตอน GetRefactoringTechniquefromTable เพื่อนำไปแก้ไขในขั้นตอนต่อไป

ทั้งนี้การเพิ่มวิธีการปฏิบัติเพื่อแก้ไขร่องรอยไม่ดีจากผู้เชี่ยวชาญเนื่องจากวิธีปฏิบัติเดิมที่กำหนดขึ้นตามตารางอาจไม่เหมาะสมสำหรับแก้ไข เพราะจะทำให้เกิดร่องรอยไม่ดีประเภทอื่นหรือไม่สามารถแก้ไขร่องรอยไม่ดีให้หมดไปได้ ผู้เชี่ยวชาญจึงได้พิจารณาตามลักษณะการทำงานของโค้ดที่ทดสอบเพื่อกำหนดวิธีการปฏิบัติเพิ่มเติมในแก้ไขร่องรอยไม่ดีที่เกิดขึ้นอย่างเหมาะสม

2.3) คัดเลือกวิธีการปฏิบัติเพื่อแก้ไขร่องรอยไม่ดีที่เกิดขึ้น เนื่องวิธีการแก้ไขมีหลากหลายวิธี ในขั้นตอน ChooseRefactoringTechniqueByHeuristic จะทำการคัดเลือกวิธีการที่จำกัดร่องรอยไม่ดีโดยพิจารณาจาก 3 องค์ประกอบคือ โค้ดที่ทดสอบ ร่องรอยไม่ดีที่เกิดขึ้น และวิธีการปฏิบัติเพื่อแก้ไขร่องรอยไม่ดีที่เกิดขึ้น

2.4) ทำการแก้ไขร่องรอยไม่ดีตามวิธีการปฏิบัติที่กำหนดในขั้นตอน DoRefactoring โดยผลลัพธ์ที่เกิดขึ้นคือโค้ดที่ทำการแก้ไขร่องรอยไม่ดีตามประเภทที่เลือกเสร็จสิ้น

2.5) ตรวจสอบชนิดของโค้ดอีกครั้งด้วยกฎการแปลความฟัซซี่โดยนำกฎการแปลความทั้ง 3 ประเภทประกอบด้วย ร่องรอยไม่ดี โค้ดที่มีความคลุมเครือและซับเซตคลีนโค้ด มาทำการ

ตรวจสอบในขั้นตอนClassifySourceCode เพื่อแก้ไขร่องรอยไม่ดีให้หมดก่อนที่จะไปสู่ขั้นตอนถัดไป ในกรณีที่ตรวจพบร่องรอยไม่ดีจะทำซ้ำในขั้นตอน2.1 จนกระทั่งไม่พบร่องรอยไม่ดีในโค้ดที่ทดสอบ

```
PROCEDURE ChooseTheBadSmell (listofbadsmell, numberofbadsmell)
INPUT: listofbadsmell คือ ตัวแปรนำเข้าข้อมูลอาเรย์ของประเภทของร่องรอยไม่ดีที่ตรวจพบ
       numberofbadsmell คือ ตัวแปรนำเข้าจำนวนของรายการร่องรอยไม่ดีที่ตรวจพบ
OUTPUT: badsmell คือ ตัวแปรที่รับค่าและส่งกลับประเภทของร่องรอยไม่ดีที่ถูกทำการแก้ไข
BEGIN
    COMMENT: impactlevel คือ ตัวแปรที่รับค่าระดับของผลกระทบเพื่อใช้ในการเปรียบเทียบ
และถูกกำหนดค่าเริ่มต้นก่อนที่จะไปการเปรียบเทียบ
     impactlevel ← 6;
     FOR index ← 0 to numberofbadsmell DO
          IF (listofbadsmell[index] is Feature Envy) OR
            (listofbadsmell[index] is Lazy Class) THEN
               IF (impactlevel > 5) THEN
                    badsmell ← listofbadsmell[index];
                    impactlevel ← 5;
               ELSE IF (impactlevel = 5) THEN
                    badsmell ←GetMinimunImpact(badsmell,
                  listofbadsmell[index]);
               END IF
          ELSE IF (listofbadsmell[index] is Large Classes) OR
                (listofbadsmell[index] is Inappropriate Intimacy) THEN
             IF (impactlevel > 4) THEN
             badsmell←listofbadsmell[index];
```

ภาพที่ 4.11 Procedure ChooseTheBadSmell เพื่อคัดเลือกและปรับปรุงร่องรอยไม่ดี

```
                    impactlevel ← 4;
                ELSE IF (impactlevel = 4) THEN
                        badsmell←GetMinimunImpact(badsmell,
                    listofbadsmell[index]);
                END IF
            ELSE IF (listofbadsmell[index] is Temporary Field) THEN
                IF (impactlevel > 3) THEN
                        badsmell ← listofbadsmell[index];
                        impactlevel ← 3;
                END IF
            ELSE IF (listofbadsmell[index] is Long Method) OR
                (listofbadsmell[index] is Long Parameter List) THEN
                IF (impactlevel > 2) THEN
                        badsmell ← listofbadsmell[index];
                        impactlevel ← 2;
                ELSE IF (impactlevel = 2) THEN
                        badsmell←GetMinimunImpact(badsmell,
                        listofbadsmell[index]);
                END IF
            ELSE IF (listofbadsmell[index] is Comment) THEN
                IF (impactlevel > 1) THEN
                        badsmell ← listofbadsmell[index];
                        impactlevel ← 1;
                END IF
            END IF
        END LOOP
        return {badsmell};
END
```

ภาพที่ 4.11 Procedure ChooseTheBadSmell เพื่อคัดเลือกและปรับปรุงร่องรอยไม่ดี(ต่อ)

```
PROCEDURE GetMinimunImpact (badsmell1, badsmell2)
INPUT: badsmell1 และ badsmell2 คือ ตัวแปรนำเข้าประเภทร่องรอยไม่ดีเพื่อทำการเปรียบเทียบ
OUTPUT: badsmell1 หรือ badsmell2 คือ ตัวแปรส่งกลับประเภทร่องรอยไม่ดีที่ผ่านการวิเคราะห์
โดยมีผลกระทบน้อยที่สุด
BEGIN
    COMMENT: impact1 คือ ตัวแปรที่รับค่าระดับผลกระทบของร่องรอยไม่ดีประเภทที่
    impact1 ← AnalysisTheFailedRules(badsmell1);
    COMMENT: impact2 คือ ตัวแปรที่รับค่าระดับผลกระทบของร่องรอยไม่ดีประเภทที่2
    impact2 ← AnalysisTheFailedRules(badsmell2);
    IF (impact1 < impact2) THEN
        return {badsmell1};
    END IF
    return {badsmell2};
END
```

ภาพที่ 4.12 Procedure GetMinimunImpact เมื่อพบกรณีที่มีร่องรอยไม่ดีต่างชนิดกันแต่ระดับของผลกระทบเท่ากัน จะเลือกร่องรอยไม่ดีที่มีผลกระทบน้อยที่สุด

ขั้นตอน GetMinimunImpact มีการวิเคราะห์ผลลัพธ์ที่ได้จากการกฎการอธิบายเพื่อเลือกแก้ไขร่องรอยไม่ดีที่เกิดขึ้น โดยทำการเปรียบเทียบร่องรอยไม่ดีทั้งสองตัวใน AnalysisTheFailedRules ทั้งนี้ผลวิเคราะห์จากผู้เชี่ยวชาญจะช่วยตัดสินใจเลือกแก้ไขร่องรอยไม่ดี โดยดูจากกฎการอธิบายความส่งค่ากลับที่เป็นเท็จประกอบการตัดสินใจ

3) **ขั้นการคัดเลือกร่องรอยไม่ดี** ChooseTheBadSmell

จะทำการเลือกร่องรอยไม่ดีโดยคำนึงถึงผลกระทบที่เกิดจากแก้ไข ตามตารางที่ 4.3 ซึ่งระดับของผลกระทบสามารถแบ่งออกได้เป็น 5 ระดับ โดยสุ่มค่าขึ้นมาเพื่อแทนระดับผลกระทบโดยเรียงจากน้อยไปหามาก คือ ระดับที่เกิดขึ้นภายใน คำอธิบายของโปรแกรม เมท็อด แอตทริบิวต์ คลาส และระหว่างคลาส โดยระดับผลกระทบ เพื่อใช้ในการเปรียบเทียบในเลือกการแก้ไขร่องรอยไม่ดี มีรายละเอียดดังนี้

(1) ระดับของคำอธิบาย

การเพิ่มเติมคำอธิบายเข้าไปในโปรแกรมเพื่ออธิบายการทำงานของโปรแกรมที่มีส่วนทำงานที่ซับซ้อน และการเพิ่มคำอธิบายเข้าไปในโปรแกรมจะไม่มีผลกระทบต่อการทำงานภายในโปรแกรมที่ถูกเพิ่มเติม

(2) ระดับของเมท็อด

ผู้วิจัยศึกษาลักษณะการเกิดของร่องรอยไม่ดีประเภทLong Method และ Long Parameter List ซึ่งมีสาเหตุเกิดจากภายในการทำงานของเมท็อดเอง ดังนั้นการแก้ไขร่องรอยไม่ดีทั้งสองประเภทนี้จะไม่มีผลกระทบออกนอกขอบเขตของการทำงานในแต่ละเมท็อด หากมองที่เป้าหมายของการทำงานที่ดีของเมท็อดซึ่งซับเซตคลีนโค้ดออกแบบไว้คือทำให้เมท็อดทำงานภายในขอบเขตที่ตัวเองรับผิดชอบ ดังนั้นการแก้ไขร่องรอยไม่ดีระดับนี้จึงไม่มีผลกระทบต่อส่วนที่เกี่ยวข้อง

(3) ระดับของแอตทริบิวต์

หลังจากพิจารณาสาเหตุการเกิดของร่องรอยไม่ดีประเภท Temporary Field ประกอบกับวิธีการแก้ไขปัญหาพบว่า วิธีการแก้ไขมีผลกระทบต่อแอตทริบิวต์และเมท็อดที่ถูกแก้ไข ซึ่งเป็นเหตุให้ร่องรอยไม่ดีประเภทนี้ถูกกำหนดให้มีระดับผลกระทบมากกว่าระดับเมท็อด เพราะเมื่อ มีการแก้ไขอาจจำเป็นจะต้องแก้ไขไปถึงเมท็อดที่เกี่ยวข้องด้วย และหากในคุณลักษณะที่ดีของแอตทริบิวต์ที่ซับเซตคลีนโค้ดกำหนดไว้คือ สร้างเพื่อเก็บข้อมูลระหว่างการทำงานของแต่ละเมท็อดให้ทำงานได้อย่างมีประสิทธิภาพ

(4) ระดับภายในคลาส

ร่องรอยไม่ดีที่จัดอยู่ในระดับผลกระทบนี้เนื่องจากมีวิธีการแก้ไขร่องรอยไม่ดีหลากหลายวิธี ซึ่งแต่ละวิธีมีผลกระทบต่อการทำงานของกลุ่มของแอตทริบิวต์หรือเมท็อด ในหลายๆกรณีจำเป็นต้องแยกคลาสออกเป็นส่วนย่อยๆ เพื่อแก้ไขร่องรอยไม่ดีประเภท Large Classes และ Inappropriate Intimacy การแก้ไขร่องรอยไม่ดีทั้งสองประเภทนี้อาจทำให้มีผลกระทบกระจายไปในแต่ละส่วนของการทำงานของคลาส เมื่อแก้ไข เรียบร้อย อาจ จำเป็น ต้องมีการปรับแต่งเรียบเรียงใหม่ ดังนั้นผลกระทบในระดับคลาสจึงมีมากกว่าระดับเมท็อดและแอตทริบิวต์

(5) ระดับระหว่างคลาส

ระดับระหว่างคลาสถือเป็นผล กระทบสูงสุดของการแก้ไขปัญหา เพราะผลกระทบระดับนี้ต้องแก้ไขในส่วนของคลาสที่ใช้งานหรือถูกใช้งาน จำเป็นต้องมีการเรียบเรียงให้ทำงานสอดคล้องหลังจากแก้ไขร่องรอยไม่ดี เรียบร้อยแล้ว ระดับนี้มี ร่องรอยไม่ดีประเภท Feature Envy และ Lazy Class ที่ส่ง ผลกระทบระหว่างคลาสต้องพิจารณาวิเคราะห์ผลกระทบ อย่างละเอียด เมื่อ

แก้ไขเสร็จ เรียบร้อย อาจมีส่วนที่ต้องเรียบเรียงใหม่เพื่อ ป้องกันข้อผิดพลาดกับ การทำงานของโปรแกรมที่มีการแก้ไข

ตารางที่ 45 ระดับของผลกระทบการเลือกร่องรอยไม่ดีเพื่อทำการแก้ไข

ลำดับที่	ชนิดของร่องรอยไม่ดี	ระดับผลกระทบ
(ร่องรอยไม่ดีที่เกิดขึ้นในระดับคำอธิบาย)		
1	Bad Comment	1
(ร่องรอยไม่ดีที่เกิดขึ้นในระดับเมท็อด)		
2	Long Method	2
3	Long Parameter List	2
(ร่องรอยไม่ดีที่เกิดขึ้นในระดับแอตทริบิวต์)		
4	Temporary Field	3
(ร่องรอยไม่ดีที่เกิดขึ้นในคลาสเดี่ยว)		
5	Large Class	4
6	Inappropriate Intimacy	4
(ร่องรอยไม่ดีที่เกิดขึ้นระหว่างคลาส)		
7	Feature Envy	5
8	Lazy Class	5

ตามตัวอย่างโค้ดชุด Hit_the_keys มีร่องรอยไม่ดีประเภท Long Parameter List และ Temporary Field จึงเลือกแก้ไขร่องรอยไม่ดีตามระดับของผลกระทบที่ต่ำที่สุดก่อน ในที่นี้จะเลือกทำการแก้ไขร่องรอยไม่ดีLong Parameter List เป็นลำดับแรก ซึ่งกรณีที่พบร่องรอยไม่ดีต่างชนิดกันแต่ระดับของผลกระทบเท่ากัน จะเลือกตัวใดตัวหนึ่งในขั้นตอนGetMinimunImpact โดยให้ผู้เชี่ยวชาญพิจารณาจากผลลัพธ์ของกฎการอธิบายที่มีค่าเป็นเท็จ เพื่อหาร่องรอยไม่ดีที่ผลกระทบน้อยที่สุดเป็นลำดับแรก

```
PROCEDURE GetRefactoringTechnique (listofbadsmell, numberofbadsmell)
INPUT: listofbadsmell คือ ตัวแปรนำเข้าข้อมูลอาเรย์ของประเภทของร่องรอยไม่ดีที่ตรวจพบ
       numberofbadsmell คือ ตัวแปรนำเข้าจำนวนของรายการร่องรอยไม่ดีที่ตรวจพบ
OUTPUT: techniques คือ ตัวแปรที่ส่งกลับอาเรย์วิธีปฏิบัติเพื่อแก้ไขร่องรอยไม่ดี
BEGIN
    FOR index ← 0 to numberofbadsmell DO
        techniques← techniques + GetRefactoringTechniquefromTable
        (listofbadsmell[index]);
    END LOOP
    return {techniques};
END
```

ภาพที่ 4.13 Procedure GetRefactoringTechnique เพื่อนำวิธีการปฏิบัติแก้ไขร่องรอยไม่ดีจากตารางที่กำหนด

4) **ขั้นตอน** GetRefactoringTechnique

เป็นการนำวิธีการปฏิบัติเพื่อแก้ไขร่องรอยไม่ดีจากตารางที่กำหนดขึ้นในภาคผนวก ง ซึ่งบางกรณีผู้เชี่ยวชาญสามารถเพิ่มวิธีการปฏิบัตินอกเหนือจากตารางเพื่อเปรียบเทียบกับวิธีแก้ไขปัญหาร่องรอยไม่ดีอื่นๆ ได้ และเพื่อใช้ประกอบการตัดสินใจในครั้งต่อไป

```
PROCEDURE ChooseRefactoringTechniqueByHeuristic (sourcecode, problem,
techniques)
INPUT: sourcecode คือ ตัวแปรนำเข้าโค้ดที่ต้องการตรวจสอบ
       problem คือ ตัวแปรนำเข้าประเภทของร่องรอยไม่ดีหรือความคลุมเครือที่ถูกเลือก
       techniques คือ  ตัวแปรนำเข้าอาเรย์ของวิธีปฏิบัติเพื่อการแก้ไขปัญหา
OUTPUT: technique คือ ตัวแปรส่งคืนวิธีปฏิบัติเพื่อแก้ไขปัญหา
```

ภาพที่ 4.14 Procedure ChooseRefactoringTechniqueByHeuristic เพื่อคัดเลือกวิธีการปฏิบัติแก้ไขปัญหาร่องรอยไม่ดีหรือความคลุมเครือ

```
BEGIN
        COMMENT: numberoftechnique คือ ตัวแปรรับค่าจำนวนของวิธีปฏิบัติที่ใช้ในการแก้ไข
        numberoftechnique ← Length(techniques);
        COMMENT: area คือ ตัวแปรรับค่าบริเวณที่เกิดร่องรอยไม่ดีหรือความคลุมเครือที่ตรวจพบ
        area ← LookupProblemArea(sourcecode, problem);
        FOR index ← 0 to numberoftechnique DO
                IF (CanFixThisProblemRootCause(area, techniques[index]) is TRUE) THEN
                        COMMENT: score คือ ตัวแปรรับค่าคะแนนที่คำนวณได้จากการ
พิจารณาวิเคราะห์วิธีปฏิบัติที่เลือกใช้ในการแก้ไขปัญหาที่เกิดขึ้น
                        score ← RefectToTheProblem(area, techniques[index]);
                        COMMENT: bestscore คือ ตัวแปรรับค่าคะแนนคำนวณที่ดีที่สุดเพื่อทำ
การเปรียบเทียบและหาค่าที่ดีที่สุดต่อไป
                        IF (bestscore < score) THEN
                                technique ← techniques[index];
                                bestscore ← score;
                        END IF
                END IF
        END LOOP
        return {technique};
END
```

ภาพที่ 4.14 Procedure ChooseRefactoringTechniqueByHeuristic เพื่อคัดเลือกวิธีการปฏิบัติแก้ไขปัญหาร่องรอยไม่ดีหรือความคลุมเครือ(ต่อ)

5) **ขั้นตอน** ChooseRefactoringTechniqueByHeuristic

เป็นขั้นตอนการคัดเลือกวิธีการปฏิบัติเพื่อแก้ไขปัญหาร่องรอยไม่ดีหรือโค้ดที่มีความคลุมเครือที่ตามดุลยพินิจของผู้เชี่ยวชาญ โดยมีขั้นตอนย่อยดังต่อไปนี้

(1) พิจารณาบริเวณที่เกิดร่องรอยไม่ดีหรือความคลุมเครือที่ถูกเลือกให้แก้ไข ในขั้นตอน LookupProblemArea โดยข้อมูลนำเข้าประกอบด้วย โค้ดที่ต้องการแก้ไขคือ ร่องรอยไม่ดีหรือโค้ดที่มี

ความคลุมเครือที่ถูกเลือก ผลลัพธ์ที่ได้คือบริเวณที่จะทำแก้ไขร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือที่เกิดขึ้นเพื่อนำไปการประเมินผลกระทบจากการแก้ไขในขั้นตอนต่อไป

(2) พิจารณาวิธีการปฏิบัติแต่ละวิธีที่รวบรวมจากตารางที่กำหนดไว้ รวมถึงวิธีการปฏิบัติจากคำแนะนำของผู้เชี่ยวชาญ เพื่อตรวจสอบวิธีการปฏิบัติแต่ละวิธีสามารถแก้ไขปัญหาที่เกิดขึ้นกับโค้ดที่ถูกทดสอบในขั้นตอน CanFixThisProblemRootCause เนื่องจากวิธีปฏิบัติบางวิธีอาจไม่เหมาะสมกับการแก้ไขปัญหาที่เกิดขึ้นกับโค้ดที่ทดสอบ อาทิเช่น วิธีปฏิบัติบางวิธีอาจไม่สามารถแก้ไขร่องรอยไม่ดีหรือความคลุมเครือที่เกิดขึ้นได้ หรือในกรณีที่แย่ที่สุดอาจจะทำให้เกิดร่องรอยไม่ดีประเภทอื่นเพิ่มเติมได้

(3) คำนวณหาค่าคะแนนแต่ละวิธีการปฏิบัติที่ผ่านการขั้นตอนการตรวจสอบขั้นต้นว่าสามารถแก้ไขร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือที่เกิดขึ้นในขั้นตอน RefectToTheProblem โดยจะเลือกวิธีการปฏิบัติที่มีคะแนนดีที่สุด โดยเปรียบเทียบทุกๆ วิธีปฏิบัติที่ผ่านการตรวจสอบเพื่อหาวิธีปฏิบัติที่สามารถแก้ไขปัญหาที่เกิดขึ้นได้อย่างมีประสิทธิภาพ

```
PROCEDURE FixTheAmbiguous (sourcecode)
INPUT: sourcecode คือ ตัวแปรนำเข้าโค้ดที่ถูกพบความคลุมเครือ
OUTPUT: sourcecode คือ ตัวแปรส่งคืนโค้ดที่ถูกแก้ไขความคลุมเครือ
BEGIN
    COMMENT: minimum คือ ตัวแปรรับค่าความพยายามน้อยที่สุดในการแก้ความคลุมเครือ
    minimum ← MAXINTEGER;
    REPEAT
        COMMENT: listofambiguous คือ ตัวแปรรับค่าอาเรย์ของประเภทคลุมเครือที่พบ
        listofambiguous ← GetTheAmbiguousType(sourcecode);
        COMMENT: numberofambiguous คือ ตัวแปรรับค่าจำนวนความคลุมเครือที่พบ
        numberofambiguous ← Length(listofambiguous);
        COMMENT: listofeffort คือ ตัวแปรรับค่าอาเรย์ของความพยายามที่คำนวณได้
จากความคลุมเครือแต่ละตัวที่ตรวจพบ
listofeffort ← GetRefactoringEffort(sourcecode, listofambiguous,
```

ภาพที่ 4.15 Procedure FixTheAmbiguous เพื่อทำการแก้ไขความคลุมเครือที่ตรวจพบจนกระทั่งโค้ดมีคุณสมบัติซับเซตคลีนโค้ด

```
                numberofambiguous);
            FOR index ← 0 to numberofambiguous DO
                IF (CanFixThisAmbiguous(sourcecode, listofambiguous[index]) is
                TRUE) THEN
                    IF (listofeffort[index] < minimum) THEN
                        minimum ← listofeffort[index];
                        COMMENT: ambiguoustype คือ ตัวแปรรับค่าประเภท
ความคลุมเครือที่ถูกเลือกในการแก้ไข
                        ambiguoustype ← listofambiguous[index];
                    END IF
                END IF
            END LOOP
            COMMENT: techniques คือ ตัวแปรที่รับค่าอาเรย์ของวิธีปฏิบัติแก้ไขความ
คลุมเครือที่ถูกรวบรวมมา
            techniques ← GetRefactoringTechniquefromTable(ambiguoustype);
            COMMENT: technique คือ ตัวแปรที่รับค่าวิธีการปฏิบัติที่ถูกเลือก
            technique ← ChooseRefactoringTechniqueByHeuristic(sourcecode,
            ambiguoustype, techniques);
            sourcecode ← DoRefactoring(sourcecode, technique);
        UNTIL (codetype is not Ambiguous OR Exit_Critiria is TRUE)
        return {sourcecode}
END
```

ภาพที่ 4.15 Procedure FixTheAmbiguous เพื่อทำการแก้ไขความคลุมเครือที่ตรวจพบจนกระทั่งโค้ดมีคุณสมบัติซับเซตคลีนโค้ด(ต่อ)

6) **ขั้นตอน** FixTheAmbiguous

เมื่อตรวจพบความคลุมเครือเกิดขึ้นในโค้ดที่ผ่านการทดสอบ ซึ่งอาจเกิดผลลัพธ์จากการแก้ไขร่องรอยไม่ดีในขั้นตอนที่แล้วหรือเกิดขึ้นอยู่ก่อนแล้ว โดยวิธีการจะเริ่มจากการตรวจสอบประเภทของความคลุมเครือที่เกิดขึ้น จากนั้นจึงทำการรวบรวมวิธีการปฏิบัติในการแก้ไขความ

คลุมเครือ จากนั้นจึงคำนวณหาค่าความพยายามที่ต้องใช้แก้ไขปัญหาความคลุมเครือแต่ละชนิดที่ตรวจพบ และทำการเลือกวิธีการเพื่อแก้ไขความคลุมเครือที่เกิดขึ้นในลำดับถัดไป ในการทำงานของขั้นตอนนี้สามารถแบ่งออกได้ดังต่อไปนี้

(1) ขั้นตอนการตรวจสอบชนิดของความคลุมเครือที่เกิดขึ้นโดยนำโค้ดทดสอบมาทำการตรวจสอบในขั้นตอนGetTheAmbiguousType ซึ่งเป็นการนำกฎการจำแนกด้วยฟัซซี่ประเภทความคลุมเครือมาประยุกต์ใช้ในการตรวจสอบ หากกฎในแต่ละข้อที่ใช้อธิบายข้อใดมีค่าเป็นจริง แสดงว่าตรวจพบความคลุมเครือประเภทนั้นในโค้ดที่ผ่านทดสอบ

(2) หลังจากตรวจหาความคลุมเครือที่เกิดขึ้นเสร็จจะทำการคำนวณเพื่อหาค่าความพยายามในการแก้ไขความคลุมเครือแต่ละประเภทที่เกิดในขั้นตอน GetRefactoringEffort เพื่อนำไปประกอบการตัดสินใจในการเลือกวิธีการแก้ไขประเภทของความคลุมเครือที่พบ

(3) พิจารณาวิธีการปฏิบัติแต่ละวิธีที่ถูกรวบรวมจากตารางที่กำหนดรวมถึงวิธีการปฏิบัติที่ผู้เชี่ยวชาญแนะนำ เพื่อตรวจสอบวิธีการปฏิบัติแต่ละวิธีสามารถแก้ไขปัญหาที่เกิดขึ้นกับโค้ดที่ผ่านขั้นตอนการทดสอบในCanFixThisAmbiguous และทดสอบเบื้องต้นว่าสามารถแก้ไขความคลุมเครือได้ นำไปพิจารณาเปรียบเทียบการเลือกวิธีปฏิบัติในขั้นตอนต่อไป

(4) ทำการเปรียบเทียบเพื่อเลือกวิธีการปฏิบัติเพื่อแก้ไขความคลุมเครือที่เกิดขึ้นโดยพิจารณาจากวิธีปฏิบัติสามารถแก้ไขความคลุมเครือที่เกิดขึ้นได้จริง และโดยเลือกค่าความพยายามน้อยที่สุดในการแก้ไขความคลุมเครือที่เกิดขึ้นเป็นส่วนประกอบการพิจารณา

(5) ทำการแก้ไขความคลุมเครือตามวิธีการปฏิบัติที่กำหนดในขั้นตอน DoRefactoring โดยผลลัพธ์ที่เกิดขึ้นคือโค้ดที่ผ่านการแก้ไขความคลุมเครือตามประเภทที่เลือกเสร็จสิ้น

(6)ตรวจสอบชนิดของโค้ดอีกครั้งด้วยกฎการแปลความฟัซซี่ในขั้นตอน Classify SourceCode ซึ่งตามวิธีการที่ได้นำเสนอ อนุญาตให้มีความคลุมเครือเกิดขึ้นได้เล็กน้อยและไม่มีผลกระทบต่อซับเซตคลีนโค้ดเมื่อถูกนำไปแปลความ ในกรณีที่ยังตรวจพบความคลุมเครือก็จะทำซ้ำในขั้นตอนแรก จนกระทั่งทุกตำแหน่งที่ถูกตรวจสอบจะมีค่าเป็นซับเซตคลีนโค้ด

PROCEDURE GetRefactoringEffort (sourcecode, listofambiguous, numberofambiguous)
INPUT: sourcecode คือ ตัวแปรนำเข้าโค้ดที่ถูกพบความคลุมเครือ
listofambiguous คือ ตัวแปรนำเข้าค่าข้อมูลอาเรย์ของประเภทของความคลุมเครือที่พบ

ภาพที่ 4.16 Procedure GetRefactoringEffort เพื่อคำนวณหาค่าความพยายามในการแก้ไขความคลุมเครือ

```
        numberofambiguous คือ ตัวแปรนำเข้าค่าจำนวนของรายการความคลุมเครือที่ตรวจพบ
OUTPUT: listofeffort คือ ตัวแปรรับค่าและส่งกลับอาเรย์ของค่าความพยายามที่คำนวณ
BEGIN
    FOR index ← 0 to numberofambiguous DO
        COMMENT: effort คือ ตัวแปรที่รับค่าความพยายามจากตารางที่กำหนด
        effort ← GetRefactoringEffortfromTable(listofambiguous[index]);
        COMMENT: numberPlaceofRefactoring คือ ตัวแปรที่รับค่าจำนวนของตำแหน่งของที่
ต้องการแก้ไขในโค้ด
        numberPlaceofRefactoring ← GetRefactoringPlace(sourcecode,
        listofambiguous[index]);
        listofeffort[index] ← (effort x numberPlaceofRefactoring);
    END LOOP
    return {listofeffort};
END
```

ภาพที่ 4.16 Procedure GetRefactoringEffort เพื่อคำนวณหาค่าความพยายาม
ในการแก้ไขความคลุมเครือ(ต่อ)

ขั้นตอนนี้จะเป็นการคำนวณหาค่าผลกระทบสำหรับแก้ไขความคลุมเครือที่ตรวจพบโดยค่าผลกระทบของความคลุมเครือแต่ละประเภทถูกนำมาจาก GetRefactoringEffortfromTable และตำแหน่งที่ต้องการแก้ไขความคลุมเครือที่เกิดขึ้นนำมาจาก GetRefactoringPlace ทั้งสองค่าสามารถนำมาหาผลคูณของค่าผลกระทบที่ใช้แก้ไขความคลุมเครือแต่ละชนิดที่ตรวจพบได้ดังนี้

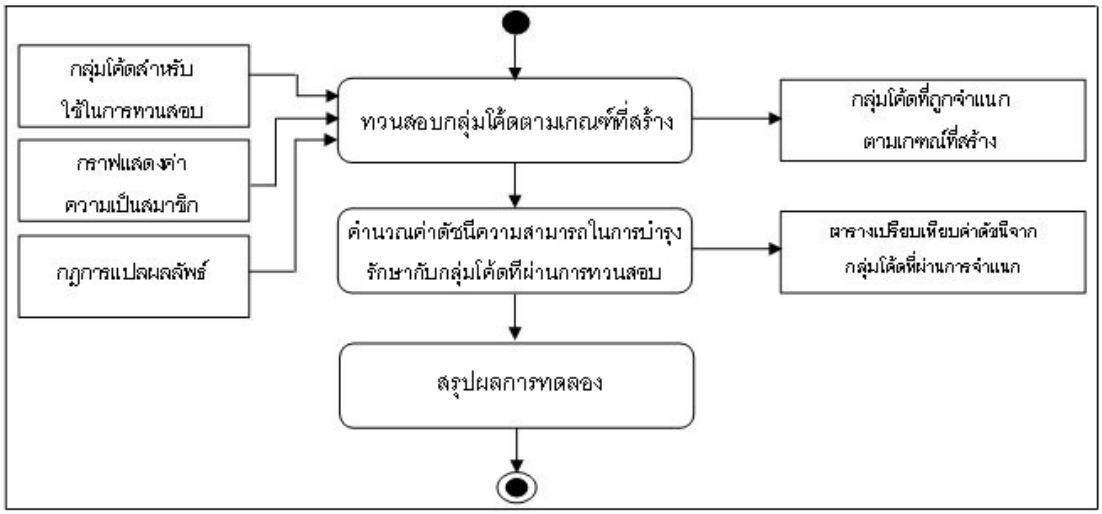
Minimum impact = (efforts x number place of refactoring) (4.1)

ค่าผลกระทบ ของความคลุมเครือแต่ละประเภทสามารถหาได้จากตารางที่ 4. 6 นำมาคูณกับจำนวนตำแหน่งที่ต้องการแก้ไข เมื่อได้ค่า ผลกระทบที่ใช้สำหรับแก้ไขความคลุมเครือในประเภทนั้นๆ จากนั้นจึงนำค่าที่ได้คำนวณได้มาเปรียบเทียบเพื่อหาวิธีการปฏิบัติและแก้ไขความคลุมเครือที่ตรวจพบต่อไป โดยตารางระดับของผลกระทบการเลือกความคลุมเครือเพื่อทำการแก้ไขดังต่อไปนี้

ตารางที่ 4.6 ระดับของผลกระทบการเลือกความคลุมเครือเพื่อทำการแก้ไข

ลำดับที่	รายการความคลุมเครือ	ความพยายามเพื่อทำรีแฟคทอริง (Effort to Refactoring)	ระดับผลกระทบ (Impact Level)									ค่าผลกระทบ (Impact)
			ไม่ได้รับผลกระทบ	ระดับผลกระทบต่ำ(Low Impact)				ระดับผลกระทบสูง (High Impact)				
			0	1	2	3	4	5	6	7	8	
(Related on Single line of Code)												
1	Ambiguity in Comment	1	0	-	-	-	-	-	-	-	-	0
(Related on Single Class Unit)												
2	Ambiguity in Method	1	-	1	-	-	-	-	-	-	-	1
3	Ambiguity in Arguments	2	-	-	4	-	-	-	-	-	-	4
4	Ambiguity in Nesting Depth	3	-	-	-	9	-	-	-	-	-	9
5	Ambiguity in Structured Programming	4	-	-	-	-	16	-	-	-	-	16
6	Ambiguity in One Thing	5	-	-	-	-	-	25	-	-	-	25
(Related on Multiple Class Units)												
7	Ambiguity in Class Organization	1	-	-	2	-	-	-	-	-	-	2
8	Ambiguity in Encapsulation	2	-	-	-	6	-	-	-	-	-	6
9	Ambiguity in Classes	3	-	-	-	-	12	-	-	-	-	12
10	Ambiguity for Change	4	-	-	-	-	-	20	-	-	-	20
11	Ambiguity in Boundary	5	-	-	-	-	-	-	30	-	-	30
12	Ambiguity of Demeter	6	-	-	-	-	-	-	-	42	-	42
13	Ambiguity in Cohesion	7	-	-	-	-	-	-	-	-	56	56

จากตารางที่ 4.6 ผู้เชี่ยวชาญแบ่งระดับ ผลกระทบเพื่อแก้ไขออกเป็น 13 ระดับซึ่งสามารถแบ่งออกเป็นกลุ่มใหญ่ได้สามกลุ่ม ได้แก่ระดับความผลกระทบที่แก้ไขในโค้ดแต่ละบรรทัด ระดับผลกระทบที่จะแก้ไขภายในคลาส และระดับผลกระทบที่จะแก้ไขระหว่างการทำงานของคลาส โดยค่าผลกระทบนี้ได้จากการศึกษาวิธีการแก้ไขความคลุมเครือแต่ละประเภท หากวิธีแก้ไขวิธีใดมีขั้นตอนที่ซับซ้อนหรือมีขั้นตอนมากจะถูกกำหนดให้มีผลกระทบมากกว่า และทำการ คัดเลือกวิธีการปฏิบัติที่เหมาะสมเพื่อแก้ไขความคลุมเคลือ เมื่อแก้ไขเสร็จแล้วจะทดสอบซ้ำจนกว่าผลลัพธ์ของโค้ดเป็นซับเซตคลีนโค้ดหรือจนกระทั่งเป็นไปตามเงื่อนไขจบการทำงาน จึงถือว่าจบการทำงานที่กำหนดไว้



ภาพที่ 4.17 วิธีการปรับปรุงคุณภาพโค้ด

เมื่อผ่านการปรับปรุงโค้ดด้วยวิธีรีแฟคทอริ่งเรียบร้อยจะเข้าสู่ขั้นตอนการทดสอบค่าดัชนีความสามารถในการบำรุงรักษาและการทวนสอบกลุ่มโค้ดตามวิธีการจำแนกที่ออกแบบไว้ ดังภาพที่ 4.17

4.3 การประเมินค่าดัชนีความสามารถในการบำรุงรักษากับกลุ่มโค้ดที่ผ่านการปรับปรุง

ทำการประเมิน ค่าดัชนีความสามารถในการบำรุงรักษา หลังจากปรับปรุงกลุ่มโค้ดให้มีคุณสมบัติที่เป็นซับเซตคลีนโค้ดแล้ว ซึ่งเป็นไปตามวิธีการที่ออกแบบไว้แล้ว โดยการประเมินค่าก่อน

และหลังจากที่ทำการปรับปรุงตามวิธีการ พบว่าค่าที่วัดได้จากกลุ่มโค้ดที่มีความคลุมเครือและร่องรอยไม่ดีมีระดับความสามารถในการบำรุงรักษาที่ดีขึ้น

4.4 การทวนสอบกลุ่มโค้ดตามวิธีการที่สร้าง

การทวนสอบกลุ่มโค้ดตามเกณฑ์ที่สร้างในการทวนสอบกลุ่มโค้ดในที่นี้ได้มาจากการแบ่งกลุ่มตามกระบวนการในข้อ 4.1 โดยมีจำนวนตัวอย่างชุดทดสอบเท่ากับกลุ่มโค้ดเริ่มต้นโดยมีโค้ดแต่ละประเภทคละกันไป เพื่อนำมาใช้ทดสอบวิธีการจำแนกตามที่สร้างขึ้นว่า มีความถูกต้องแม่นยำมากน้อยเพียงใด โดยการจำแนกจะมีผลลัพธ์อยู่ 3 แบบคือ เป็นกลุ่มซับเซตคลีนโค้ด กลุ่มโค้ดที่มีความคลุมเครือ และกลุ่มโค้ดที่มีร่องรอยไม่ดี โดยผลลัพธ์ที่ได้จากขั้นตอนนี้แสดงในบทที่ 5 ทั้งนี้ โดยกลุ่มที่ผ่านการจำแนกในขั้นตอนนี้จะต้องไปผ่านการประเมินค่าดัชนีความสามารถในการบำรุงรักษาเพื่อนำไปสรุปผล

4.5 การประเมินค่าดัชนีความสามารถในการบำรุงรักษากับกลุ่มโค้ดที่ผ่านการทวนสอบ

วิเคราะห์ค่าดัชนีความสามารถในการบำรุงรักษาจากกลุ่มโค้ดที่ผ่านการจำแนกตามเกณฑ์ ในกลุ่มของซับเซตคลีนโค้ด โค้ดที่มีความคลุมเครือ และร่องรอยไม่ดี โดยกลุ่มซับเซตคลีนโค้ดมีคุณสมบัติด้านการบำรุงรักษาในระดับที่ดี ตามที่ได้ศึกษามาในวิทยานิพนธ์นี้จะนำผลลัพธ์ที่ประเมินได้มาทำการวิเคราะห์และพิจารณามาทำการสรุปผลต่อไป

4.6 สรุปผลการทดลอง

จากการทดลองจะเห็นได้ว่ากฎการจำแนกที่สร้างขึ้นสามารถจำแนกกลุ่มโค้ดทั้ง 3 ประเภทได้อย่างมีประสิทธิภาพ และไม่ว่ากลุ่มนั้นจะเป็นร่องรอยไม่ดี หรือโค้ดที่มีความคลุมเครือก็สามารถที่จะทำการปรับปรุงตามวิธีปฏิบัติให้มีคุณสมบัติเป็นซับเซตคลีนโค้ด โดยวิธีการที่ออกแบบนี้สามารถเพิ่มค่าความสามารถในการบำรุงรักษาให้แก่โค้ดที่ทำการทดสอบได้ และยังเป็นวิธีการที่สนับสนุนการผลิตซอฟต์แวร์ที่มีคุณภาพหรือปรับปรุงซอฟต์แวร์ให้มีคุณภาพตามที่วิทยาพนธ์นี้ได้ออกแบบไว้

บทที่ 5
ผลการจำแนกโค้ดและวิธีการปฏิบัติเพื่อปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี และโค้ดที่มีความคลุมเครือ โดยใช้เทคนิครีแฟคทอริ่ง

5.1 ผลการจำแนกโค้ด

วิทยานิพนธ์นี้ได้รวบรวมกลุ่มโค้ดสำหรับทดสอบ โดยการรวบรวมจากหนังสือด้านการเขียนโปรแกรมหรือจากงานวิจัยที่เกี่ยวข้องและตามเว็บไซต์ต่างๆ ผ่านการแปลโปรแกรมโดยไม่มีข้อผิดพลาด ในโค้ดแต่ละชนิดที่ถูกเลือกจะมีคุณสมบัติตามข้อกำหนดในการวิทยานิพนธ์ตามเกณฑ์ 3.1 ปรากฏในหน้าที่ 36 และ 4.1 ปรากฏในหน้าที่ 59 ซึ่งรายละเอียดผลสรุปการจำแนกโค้ดและแหล่งอ้างอิงแสดงไว้ในภาคผนวก ฉ เริ่มต้นจากนำโค้ดภาษา C# ที่ได้รวบรวมแบ่งกลุ่มเป็นออกเป็น 2 ส่วน คือกลุ่มที่ 1 ชุดเริ่มต้น โดยมีวัตถุประสงค์หลักเพื่อสร้างวิธีการจำแนกโค้ดโดยแบ่งออกเป็น 3 ประเภทได้แก่ ซับเซตคลีนโค้ด โค้ดที่มีความคลุมเครือ และร่องรอยไม่ดี กลุ่มตัวอย่างชุดเริ่มต้นประกอบด้วยชุดโค้ดตัวอย่างจำนวน60 ตัวอย่าง แบ่งออกเป็น3 ประเภทเท่าๆกัน ประเภทละ 20 ตัวอย่าง โดยมีประเภทของร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือครบทุกประเภทตามที่กำหนดไว้ในขอบเขตงานวิจัย สำหรับชุดโค้ดกลุ่มที่ 2 คือกลุ่มชุดทวนสอบ ยังคงใช้ชุดโค้ดภาษา C# เช่นเดิมประกอบด้วยกลุ่มโค้ด 3 ประเภทคละกัน และมี ครบทุกประเภทเช่นเดียวกับชุดตัวอย่างเริ่มต้นทดสอบโดยมีวัตถุประสงค์เพื่อใช้ตรวจสอบความถูกต้องของ วิธีการจำแนกที่สร้างขึ้น เมื่อตรวจสอบผลลัพธ์ที่ได้จากการทวนสอบสรุปได้ว่าวิธีการที่สร้างสามารถจำแนกโค้ดได้อย่างถูกต้องโดยจะสรุปผลลัพธ์ไว้ในส่วนการทวนสอบซึ่งอยู่ข้างท้ายของบท

5.2 ผลจากการนำมาตรวัดทางซอฟต์แวร์มาประยุกต์ใช้

มาตรวัดซอฟต์แวร์ที่นำมาใช้ในงานวิทยานิพนธ์นี้ ผู้วิจัยพบข้อจำกัดบางประการในระหว่างดำเนินการวัดค่ากลุ่มตัวอย่างทดสอบด้วยมาตรวัดซอฟต์แวร์ ปัญหาตรวจพบในขั้นตอนการวัดโค้ดนั้นอาจมีกลุ่มโค้ดในส่วนของเมท็อด แอตทริบิวต์ และคลาสที่ถูกสร้างขึ้นด้วยโปรแกรมโดยอัตโนมัติ หรือเป็นกลุ่มโค้ดที่ถูกสร้างเพื่อทดสอบแบบไวท์บ๊อกซ์ ทำให้การทำงานของเมท็อด แอตทริบิวต์ และคลาสมีลักษณะคล้ายกับร่องรอยไม่ดีบางประเภท อาทิเช่น ร่อยรอยไม่ดีประเภท Long Method ที่เกิดขึ้นในเมท็อดร่องรอยไม่ดีประเภท Temporary Field ที่เกิดขึ้นในแอตทริบิวต์ หรือร่องรอยไม่ดีประเภท Large Classes ที่เกิดขึ้นในคลาส ทำให้ผู้วิจัยจำเป็นต้องปรับค่าที่วัดได้

ที่อยู่ในช่วงร่องรอยไม่ดีให้เป็นค่าความคลุมเครือแทน โดยจะอธิบายโดยยกตัวอย่างประกอบต่อไป อย่างไรก็ตามผู้วิจัยจึงกำหนดข้อจำกัดให้กับเมท็อด แอตทริบิวต์และคลาส โดยมีข้อจำกัดมีดังต่อไปนี้

1) **ข้อจำกัดในระดับเมท็อด**ได้แก่

(1) **ข้อจำกัดเมท็อดประเภทที่** 1 (M1)

เป็นเมท็อดที่เกิดจากสร้างอัตโนมัติจากเครื่องมือของภาษา

(2) **ข้อจำกัดเมท็อดประเภทที่** 2 (M2)

เป็นเมท็อดที่สร้างเพื่อทำการทดสอบโค้ดหรือโปรแกรม อาทิเช่นการทดสอบแบบไวท์บ๊อกซ์

(3) **ข้อจำกัดเมท็อดประเภทที่** 3 (M3)

เมท็อดที่ถูกกำหนดให้ถูกอิมพลีเมนท์(Implement) เนื่องจากสืบทอดพฤติกรรมจากคลาสที่ทำการสืบทอด

2) **ข้อจำกัดในระดับแอตทริบิวต์**ได้แก่

(1) **ข้อจำกัดแอตทริบิวต์ประเภทที่** 1 (F1)

แอตทริบิวต์ที่เกิดจากการสร้างขึ้นโดยเครื่องมืออัตโนมัติ เพื่อให้เลือกใช้ข้อมูลทำงานได้สะดวก

(2) **ข้อจำกัดแอตทริบิวต์ประเภทที่** 2 (F2)

แอตทริบิวต์ที่ถูกกำหนดให้ถูกอิมพลีเมนท์ เนื่องจากสืบทอดพฤติกรรมจากคลาสที่ทำการสืบทอด

3) **ข้อจำกัดในระดับคลาส**ได้แก่

(1) **ข้อจำกัดคลาสประเภทที่** 1 (C1)

คลาสที่มีส่วนใดส่วนหนึ่งของเมท็อดหรือแอตทริบิวต์ถูกสร้างขึ้นอัตโนมัติด้วยเครื่องมือของภาษา

(2) **ข้อจำกัดคลาสประเภทที่** 2 (C2)

เป็นคลาสที่ถูกสร้างเพื่อใช้ในการทดสอบการทำงานของโปรแกรมหรือเป็นคลาสหลักที่ถูกเรียกทำงานในตอนเริ่มต้น เมื่อมีการเรียกใช้โปรแกรมประยุกต์

(3) **ข้อจำกัดคลาสประเภทที่** 3 (C3)

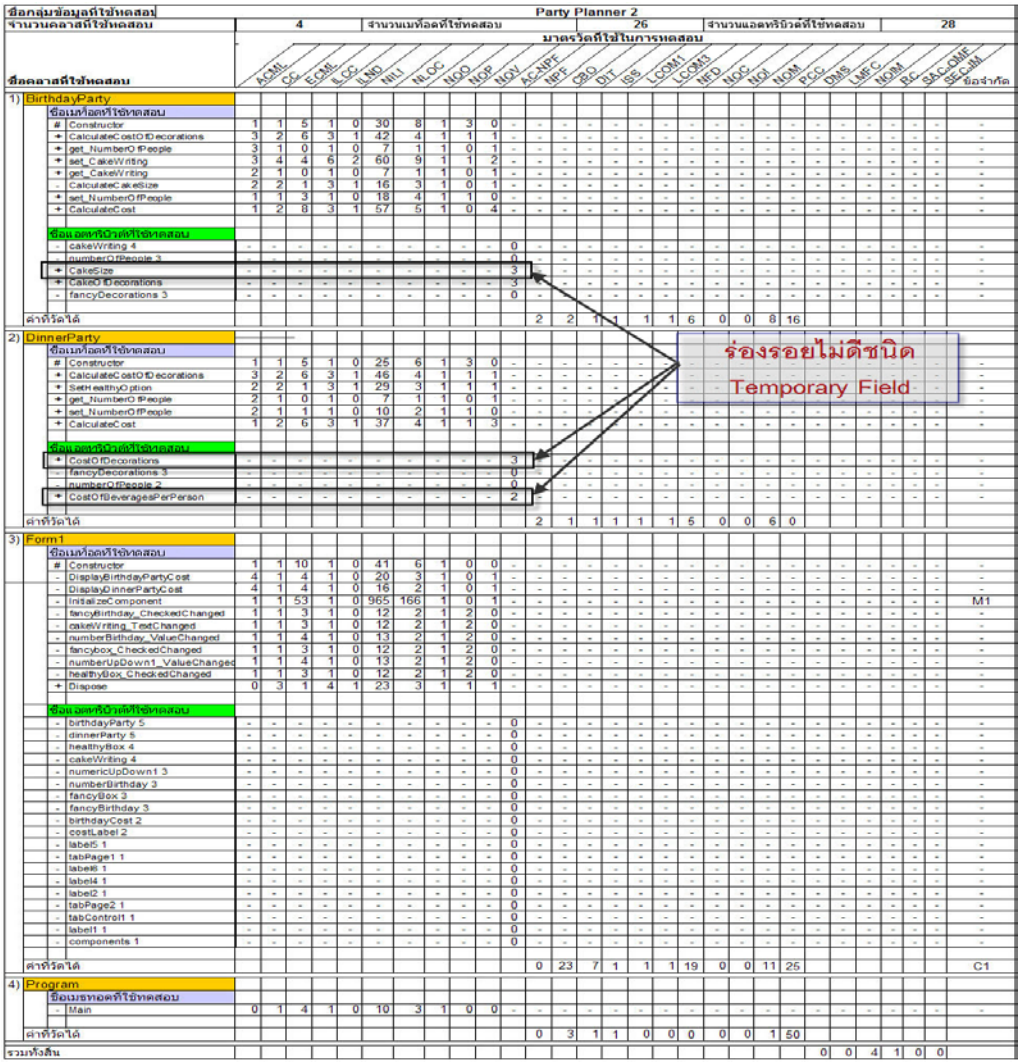
คลาสที่ถูกกำหนดให้ทำการอิมพลีเมนท์เนื่องจากมีการสืบทอดจากคลาสแม่

(4) **ข้อจำกัดคลาสประเภทที่** 4 (C4)

เป็นคลาสที่ถูกสร้างเมท็อดและแอตทริบิวต์โดยอัตโนมัติโดยโครงสร้างข้อมูลในฐานข้อมูล

ต่อไปจะเป็นตัวอย่างผลลัพธ์ที่ได้ผ่านกระบวนการวัดจากมาตรวัดซอฟต์แวร์ที่ผู้วิจัยได้กำหนดไว้ในข้อ 3.1.3

ตารางที่ 5.1 ผลลัพธ์กลุ่มร่องรอยไม่ดีที่ผ่านกระบวนการวัดด้วยมาตรวัดซอฟต์แวร์



จากตารางที่ 5.1 แสดงให้เห็นถึงข้อจำกัดที่เกิดขึ้นกับกลุ่มตัวอย่างการทดลอง เช่น เมท็อด initializeComponent เป็นเมท็อดที่ถูกสร้างอัตโนมัติ ซึ่งเป็นการสร้างฟอร์มเพื่อให้ค่าเริ่มต้นแก่ส่วนต่อประสานกับผู้ใช้และกำหนดค่าเริ่มในการทำงานต่างๆ สังเกตได้จากโค้ดที่อยู่ภายในเมท็อด

นี้มีพฤติกรรมเหมือนร่องรอยไม่ดีประเภท Long method ด้วยเหตุผลเดียวกันนี้ คลาส Form1 ซึ่งสร้างด้วยโปรแกรมอัตโนมัติทำให้เกิดการสืบทอดและอ้างถึงการใช้งานจากวัตถุจำนวนมาก ดังนั้น เมื่อคลาส เมท็อด และแอตทริบิวต์ ถูกตรวจสอบว่าเป็นข้อจำกัด ผู้วิจัยจะกำหนดค่าพิจารณาใหม่ให้ ส่วนในกรณีที่ค่าได้จากการวัดอยู่ที่ระดับร่องรอยไม่ดีจะเปลี่ยนแปลงให้อยู่ในค่ากลางของความคลุมเครือแทน จากนั้นนำค่าพิจารณาที่กำหนดใหม่ในโค้ดที่ตรวจพบข้อจำกัดมาผ่านขั้นตอนการจำแนก โดยมีค่าพิจารณาดังตารางที่ 5.2

ตารางที่ 5.2 ค่ามาตรวัดซอฟต์แวร์ที่แก้ไขให้หลังจากตรวจพบข้อจำกัด

ลำดับที่	มาตรวัดที่ใช้ในการทดสอบ	ค่าที่ทดสอบได้(x)	ค่าที่แก้ไขหลังจากตรวจพบข้อจำกัด(x)
1	CC	x >=30	x=20
2	ILCC	x>=60	x=40
3	LCOM1	x>0.8	x=0.7
4	LCOM3	x>1.0	x=0.9
5	NFD	x>20	x=15
6	NILI	x>=200	x=150
7	NLOC	x>=60	x=41
8	NOM	x>20	x=17
9	NOP	x>7	x=6
10	PCC	X=0	X=10

ดังนั้นหากเมท็อดที่ถูกระบุเป็นข้อจำกัด จะถูกวัดค่าด้วยมาตรวัดที่กำหนดตามตาราง หากค่าที่วัดได้มีผลลัพธ์อยู่ที่ระดับร่องรอยไม่ดีอาทิ เช่น เมท็อดหนึ่งวัดค่าจำนวนของบรรทัดได้ 45 บรรทัด ต้องแก้ไขค่ามาตรวัดให้เหลือเพียง 30 บรรทัดก่อน ซึ่งค่าที่กำหนดให้นี้คือ ค่ากลางของความคลุมเครือแทน

แม้ว่าค่าผลลัพธ์ที่ถูกวัดออกมามีระดับที่อยู่ในเกณฑ์ร่องรอยไม่ดี แต่เนื่องจากเป็นโค้ดที่ถูกสร้างขึ้นด้วยเครื่องมือหรือเกิดขึ้นตามข้อจำกัดแบบอื่นๆ แต่คลาส เมท็อด และแอตทริบิวต์ก็ถือว่าไม่ใช่ร่องรอยไม่ดีที่ถูกสร้างโดยผู้พัฒนาเอง ดังนั้นกลุ่มโค้ดที่ไม่ได้ถูกสร้างจากผู้พัฒนาจึงเป็นเหตุให้ไม่สามารถจะกำหนดค่าให้เป็นซับเซตของคลีนโค้ดได้ กลุ่มโค้ดนี้ถูกปฏิเสธเพราะ

คุณสมบัติของซับเซตของคลีนโค้ดที่ผู้วิจัยออกแบบในขั้นต้น ดังนั้นผู้วิจัยจึงขอกำหนดค่าโดยให้ใช้ค่ากลางของความคลุมเครือ ซึ่งสามารถนำไปจำแนกและปรับปรุงแก้ไขในลำดับต่อไป

เพื่อแสดงถึงขั้นตอนการนำมาตรวัดมาประยุกต์ใช้ให้ชัดเจนขึ้น จึงขอยกตัวผลลัพธ์ที่ได้จากการจำแนกโค้ดออกเป็นกลุ่มซับเซตคลีนโค้ด กลุ่มโค้ดที่มีความคลุมเครือ และกลุ่มร่องรอยไม่ดี โดยแสดงค่าในแต่ละกลุ่มโค้ดถูกแบ่งออกเป็น 3 กรณีดังนี้

1) กรณีที่ 1 ผลลัพธ์เป็นคลีนโค้ด ดังตารางที่ 5.3

2) กรณีที่ 2 ผลลัพธ์เป็นร่องรอยไม่ดี ซึ่งจำเป็นต้องแก้ไขให้เป็นคลีนโค้ด ดังตารางที่ 5.4

3) กรณีที่ 3 ผลลัพธ์เป็นโค้ดที่มีความคลุมเครือ ซึ่งจะถูกแก้ไขให้เป็นคลีนโค้ด ดังตารางที่ 5.5

ตารางที่ 5.3 ผลลัพธ์ที่วัดได้เป็นคลีนโค้ด

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ	Encryption and Decryption																												
จำนวนคลาสที่ใช้ทดสอบ	2				จำนวนเมท็อดที่ใช้ทดสอบ								6						จำนวนแอตทริบิวต์ที่ใช้ทดสอบ							3			
ชื่อคลาสที่ใช้ทดสอบ	มาตรวัดที่ใช้ในการทดสอบ																												
	ACML	CC	ECML	ILCC	ILND	NILI	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-OMF	SEC-IM	ข้อจำกัด
1) ArrayConverter																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
+ ConvertToBytes	1	2	2	3	1	36	6	1	1	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้											0	2	1	1	0	0	0	0	0	1	0								
2) CryptoHelper																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
+ SetKeyAndInitialVector	2	1	6	1	0	19	2	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ SetAlgorithm	2	1	3	1	0	13	3	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Key	1	3	4	3	1	38	6	1	1	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Encrypt	0	1	12	3	2	52	8	1	1	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Decrypt	0	1	11	3	2	59	9	1	1	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
- initialVector	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- key	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- algorithm	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้											0	9	3	1	0	0	3	0	0	0	N/A								
รวมทั้งสิ้น																						0	0	1	1	0	0		

ตารางที่ 5.4 ผลลัพธ์ที่วัดได้เป็นร่องรอยไม่ดี

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ	Hit_the_keys																												
จำนวนคลาสที่ใช้ทดสอบ	3					จำนวนเมท็อดที่ใช้ทดสอบ					8								จำนวนแอตทริบิวต์ที่ใช้ทดสอบ				16						
	มาตรวัดที่ใช้ในการทดสอบ																												
ชื่อคลาสที่ใช้ทดสอบ	ACML	CC	ECML	ILCC	ILND	NILI	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-OMF	SEC-IM	ข้อจำกัด
1) Form1																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
# Constructor	1	1	4	1	0	18	5	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- Form1_KeyDown	1	5	12	6	2	173	23	1	2	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- InitializeComponent	1	1	55	1	0	396	71	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M1
- timer1_Tick	1	2	6	2	1	41	5	1	2	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Dispose	0	3	1	4	1	23	3	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
- components_3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- listBox1_3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- timer1_3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- difficultyProgressBar_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- correctLabel_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- missedLabel_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- accuracyLabel_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- random_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- totalLabel_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- stats_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- statusStrip1_1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- toolStripStatusLabel1_1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	30	7	1	1	1	12	0	0	5	32							C1
2) Program																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
- Main	0	1	4	1	0	10	3	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	3	1	1	0	0	0	0	0	1	50							
3) Stats																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
# Constructor	1	1	1	1	0	16	4	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Update	1	2	0	3	1	74	11	1	8	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
+ Correct	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Accuracy	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Total	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Missed	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้											4	0	1	1	0	0	4	0	0	2	0								
รวมทั้งสิ้น																						0	0	2	1	0	0		

ตารางที่ 5.5 ผลลัพธ์วัดได้เป็นโค้ดที่มีความคลุมเครือ

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ	TaxApp																												
จำนวนคลาสที่ใช้ทดสอบ	2					จำนวนเมท็อดที่ใช้ทดสอบ					8								จำนวนแอตทริบิวต์ที่ใช้ทดสอบ				9						
	มาตรวัดที่ใช้ในการทดสอบ																												
ชื่อคลาสที่ใช้ทดสอบ	ACML	CC	ECML	ILCC	ILND	NILI	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-OMF	SEC-IM	ข้อจำกัด
1) Form1																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
# Constructor	1	1	2	1	0	12	3	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- btnSubmit_Click	1	2	10	2	1	91	7	1	2	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- btnClear_Click	1	1	1	1	0	9	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- InitializeComponent	1	1	35	1	0	384	69	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M1
- getTotal	1	1	2	1	0	12	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- getTxtAmount	0	3	1	4	1	12	2	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Dispose	0	3	1	4	1	23	3	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
- lblAmount_1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- btnClear_1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- btnSubmit_1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- lblTax_1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- lblResult_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- nudTax_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- components_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Tax_4	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- txtAmount_5	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้											1	3	7	1	1	1	8	0	0	5	8								C1
2) Program																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
- Main	0	1	4	1	0	10	3	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้											0	22	1	1	0	0	0	0	0	1	50								
รวมทั้งสิ้น																						0	0	2	1	0	0		

หากพิจารณาผลลัพธ์ในตารางจะพบว่า ร่องรอยไม่ดีบางประเภทสามารถตรวจสอบได้โดยใช้มาตรวัดซอฟต์แวร์แต่ไม่ครอบคลุมประเภทของร่องรอยไม่ดีได้ทุกกรณี และหากพิจารณาผลลัพธ์ในส่วนของโค้ดที่มีความคลุมเครือและซับเซตคลีนโค้ด มาตรวัดซอฟต์แวร์ที่ใช้ก็ไม่สามารถแยกแยะออกได้อย่างชัดเจน ด้วยเหตุนี้ผู้วิจัยจึงนำกฎฟัซซี่โลจิกมาประยุกต์ใช้ในกระบวนการเพื่อจำแนกกลุ่มโค้ดได้ชัดเจนยิ่งขึ้น

5.3 ผลการนำกฎฟัซซี่ที่สร้างประยุกต์ใช้เพื่อการจำแนก

เมื่อถึงขั้นตอนนี้จะเลือกกฎที่สร้างขึ้นในข้อ 4.15 นำไปประยุกต์ใช้งาน โดยแบ่งกฎการใช้งานออกได้เป็น 2 ส่วนคือ กฎที่ใช้ในการจำแนกและกฎที่ใช้อธิบายรายละเอียด กฎที่ใช้เพื่อการจำแนกคือ กฎที่ใช้ระบุประเภทของโค้ดซึ่งแบ่งออกเป็น ซับเซตคลีนโค้ด โค้ดที่มีความคลุมเครือและร่องรอยไม่ดี ส่วนกฎที่ใช้ในการอธิบายรายละเอียดคือ กฎย่อยของกฎการจำแนกเป็นจุดที่ชี้ให้เห็นถึงสถานะของโค้ดที่ทดสอบถูกจัดอยู่ในประเภทย่อยอะไร อาทิเช่น โค้ดที่จำแนกให้เป็นร่องรอยไม่ดี หากพิจารณากฎที่ใช้อธิบายรายละเอียดที่สร้างขึ้น ถ้าหากมีกฎข้อใดมีค่าเป็นจริง จะหมายถึง โค้ดที่นำมาทดสอบมีร่องรอยไม่ดีประเภทนั้นเกิดขึ้นตามกฎที่มีค่าเป็นจริง การนำกฎฟัซซี่มาประยุกต์ใช้ในการจำแนก จะต้องแปลความในทุกคลาส ทุกเมท็อด ทุกแอตทริบิวต์ ผลลัพธ์ที่ได้ต้องเป็นคลีนทั้งหมด จึงถือว่าเป็นซับเซตคลีนโค้ด ดังตัวอย่างตารางที่ 5.6

ตารางที่ 5.6 ตัวอย่างผลลัพธ์ที่แสดงค่าการแปลความที่ถูกจัดให้อยู่ในกลุ่มซับเซตคลีนโค้ด

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			Encryption and Decryption		
	ชุดรายการ			Defuzzification Output	สรุปค่าแปลความ
ลำดับที่	คลาส	เมท็อด	แอตทริบิวต์		
1	ArrayConverter	ConvertToBytes	-	0.242	clean
2	CryptoHelper	SetKeyAndInitialVector	initialVector	0.261	clean
3		SetAlgorithm	key	0.261	clean
4		Key	algorithm	0.261	clean
5		Encrypt		0.268	clean
6		Decrypt		0.282	clean

ในส่วนที่เป็นกลุ่มโค้ดที่มีความคลุมเครือ จะพิจารณาค่าแปลความได้โดยสังเกตจาก หากมีค่าความคลุมเครือเกิดขึ้นในตำแหน่งใดๆ ของคลาส เมท็อด หรือ แอตทริบิวต์ ก็ถือว่าจัดอยู่ในกลุ่มโค้ดที่มีความคลุมเครือซึ่งต้องได้รับการแก้ไข ดังตารางที่ 5.7

ตารางที่ 5.7 ตัวอย่างผลลัพธ์ที่แสดงค่าการแปลความที่ถูกจัดให้อยู่ในกลุ่มโค้ดที่มีความคลุมเครือ

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		TaxApp			
	ชุดรายการ			Defuzzification Output	สรุปค่าแปลความ
ลำดับที่	คลาส	เมท็อด	แอตทริบิวต์		
1	Form1	Constructor	lblAmount 1	0.326	Clean
2		btnSubmit_Click	btnClear 1	0.346	Ambiguous
3		btnClear_Click	btnSubmit 1	0.326	Clean
4		InitializeComponent	lblTax 1	0.35	Ambiguous
5		getTotal	lblResult 2	0.326	Clean
6		getTxtAmount	nudTax 2	0.35	Ambiguous
7		Dispose	components 2	0.35	Ambiguous
8			Tax 4		
9			txtAmount 5		
10	Program	Main		0.287	Clean

ซึ่งการพิจารณากลุ่มโค้ด หากค่าแปลความแสดงให้เห็นว่ามีร่องรอยไม่ดีอยู่ในคลาส ไม่ว่าจะมีค่าของคลีนโค้ดและโค้ดที่มีความคลุมเครือปะปนอยู่ ก็ยังคงจัดกลุ่มให้เป็น กลุ่มโค้ดร่องรอยไม่ดีเพื่อนำไปปรับปรุงแก้ไข ดังตารางที่ 5.8

ตารางที่ 5.8 ตัวอย่างผลลัพธ์ที่แสดงค่าการแปลความที่ถูกจัดให้อยู่ในกลุ่มร่องรอยไม่ดี

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		Hit_the_keys			
	ชุดรายการ			Defuzzification Output	สรุปค่าแปลความ
ลำดับที่	คลาส	เมท็อด	แอตทริบิวต์		
1	Form1	Constructor	components 3	0.297	Clean
2		Form1_KeyDown	listBox1 3	0.317	Clean
3		InitializeComponent	timer1 3	0.334	Ambiguous
4		timer1_Tick	difficultyProgressBar 2	0.297	Clean
5		Dispose	correctLabel 2	0.321	Clean
6			missedLabel 2		
7			accuracyLabel 2		
8			random 2		
9			totalLabel 2		
10			stats 2		
11			statusStrip1 1		
12			toolStripStatusLabel1 1		
13	Program	Main		0.263	Clean
14	Stats	Constructor	Correct	0.799	Bad
15		Update	Accuracy	0.799	Bad
16			Total		
17			Missed		

จากตารางตัวอย่างการแปลความทั้ง 3 กลุ่ม สำหรับกลุ่มโค้ดทดสอบที่ถูกจัดให้เป็นกลุ่มโค้ดที่มีความคลุมเครือและกลุ่มร่องรอยไม่ดี จำเป็นต้องนำมาปรับปรุงโค้ดก่อน โดยที่ผู้วิจัยได้กำหนดวิธีและวิธีการปฏิบัติเพื่อปรับปรุงไว้ ซึ่งมีรายละเอียดในขั้นตอนต่อไป

5.4 ผลการปรับปรุงโค้ดให้เป็นไปตามเกณฑ์ด้วยเทคนิครีแฟคทอริ่ง

การปรับปรุงโค้ดให้เป็นไปตามเกณฑ์ด้วยเทคนิครีแฟคทอริ่ง มีจุดมุ่งหมายเพื่อทำการปรับปรุงโค้ด ให้มีคุณสมบัติตามเกณฑ์คุณภาพของซับเซตคลีนโค้ดซึ่งในที่จะทำการตรวจสอบ

ชนิดของร่องรอยไม่ดีเพื่อคัดเลือกเทคนิคการปรับปรุงที่เหมาะสมให้กับโค้ดกลุ่มนั้น โดยผลลัพธ์จากขั้นตอนที่ได้คือ โค้ดที่ผ่านเกณฑ์การพิจารณาและมีคุณสมบัติที่เป็นซับเซตคลีนโค้ดเท่านั้น ดังตัวอย่างตารางที่ 5.6

กรณีที่เป็นกลุ่มคำสั่งที่ตรวจพบร่องรอยไม่ดี ในขั้นตอนนี้แสดงวิธีการรีแฟคทอริ่งของ Martin Fowler [6] เพื่อเป็นทางเลือกสำหรับปรับปรุงโค้ดให้เหมาะสมกับชนิดและประเภทของร่องรอยไม่ดีนั้นๆ แต่ในกรณีที่เป็นกลุ่มโค้ดที่มีความคลุมเครือ ในขั้นตอนนี้คือ การ เลือกวิธีการแก้ไขให้ถือเป็นวิจารณญาณของผู้เชี่ยวชาญด้านโปรแกรม จะเป็นผู้เลือกวิธีที่ใช้ในการปรับปรุงให้แก่กลุ่มโค้ดที่มีความคลุมเครือโดยรายละเอียดวิธีการแก้ไขจะแสดงอยู่ในภาคผนวก ง

ซึ่งผู้วิจัยขอยกตัวอย่างวิธีการปรับปรุงร่องรอยไม่ดีตามวิธีการที่ออกแบบไว้ โดยนำตัวอย่างชุดโค้ดร่องรอยไม่ดีในข้อ 5.2 ตารางที่ 5.4 มาปรับปรุงโดยใช้วิธีปฏิบัติที่กำหนดไว้และได้ผลลัพธ์ดังตารางที่ 5.9 จากนั้นจะทำซ้ำตามวิธีการที่ได้ออกแบบไว้ โดยใช้มาตรวัดซอฟต์แวร์วัดผลอีกครั้ง จึงได้ผลลัพธ์ดังตารางที่ 5.10 พร้อมกับใช้กฎฟัซซี่เพื่อแปลความ และทำการพิจารณาจากผลลัพธ์เพื่อตัดสินใจว่าต้องการปรับปรุงอีกครั้งหรือไม่ โดยดูผลลัพธ์จากตารางที่ 5.11

ตารางที่ 5.9 วิธีการปฏิบัติเพื่อปรับปรุงโค้ดกลุ่มร่องรอยไม่ดีครั้งที่ 1

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			Hit_the_keys		
ลำดับที่	**คลาส**	**เมท็อด**	**แอตทริบิวต์**	**ผลลัพธ์การแปลความ**	**สรุปค่าแปความ**
1	Stats	Constructor	Correct	0.799	Bad
2		Update	Accuracy	0.799	Bad
3			Total		
4			Missed		
ประเภทร่องรอยไม่ดีที่ต้องการแก้ไข			Long Parameter List และ Temporary Field		
วิธีการรีแฟคทอริ่งสำหรับการแก้ไขร่องรอยไม่ดี					
1. Composing Method วิธีย่อยที่ใช้ Introduce Parameter Object					
2. Making Method Calls Simpler วิธีย่อยที่ใช้ Preserve Whole Object					
3. Making Method Calls Simpler วิธีย่อยที่ใช้ Replace Parameter with Method					
4. Moving Features between Objects วิธีย่อยที่ใช้ Extract Class					
5. Simplifying Conditional Expression วิธีย่อยที่ใช้ Introduce Null Object					

ตารางที่ 5.9 วิธีการปฏิบัติเพื่อปรับปรุงโค้ดกลุ่มร่องรอยไม่ดีครั้งที่ 1 (ต่อ)

วิธีที่เลือกใช้ในการปรับปรุงร่องรอยไม่ดี	Making Method Calls Simpler วิธีย่อยที่ใช้ Replace Parameter with Method
เหตุผลที่เลือกใช้	
ตัวอย่างมีร่องรอยไม่ดีอยู่สองชนิด ดังนั้นควรจัดการ Long Parameter List ตามลำดับของผลกระทบที่กำหนดไว้ตามวิธีปฏิบัติที่ออกแบบไว้ ในส่วนนี้ค่อนข้างที่จะเห็นสาเหตุของปัญหาได้ง่ายโดยสังเกตได้จากจำนวนของพารามิเตอร์ที่รับเข้าทำงานในเมท็อดมีจำนวนมาก ซึ่งพารามิเตอร์บางตัวไม่จำเป็นต้องนำเข้า ดังนั้นแก้ไขด้วยการลดจำนวนของพารามิเตอร์ลงแล้วให้เรียกใช้ค่าจากเมท็อดแทนโดยวิธีการ Replace Parameter with Method เพื่อสร้างเมท็อดใหม่ในการเรียกใช้งานภายในเมท็อด Update สามารถลดจำนวนของพารามิเตอร์ที่รับเข้ามาได้แล้วจึงทำการทดสอบก่อนจะดำเนินการในขั้นตอนต่อไป	

ตารางที่ 5.10 ตัวอย่างผลลัพธ์หลังจากการปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี ครั้งที่ 1

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		Hit_the_keys																										
จำนวนคลาสที่ใช้ทดสอบ		3				จำนวนเมท็อดที่ใช้ทดสอบ					9								จำนวนเอตทริบิวต์ที่ใช้ทดสอบ				16					
		มาตรวัดที่ใช้ในการทดสอบ																										
ชื่อคลาสที่ใช้ทดสอบ		AGML	CC	ECML	ILCC	ILND	NIL1	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	ISS	LCOM1	LCOM5	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOAM	RC	SAC-OMF	SIE-CJM	ข้อจำกัด
1)	Form1																												
	ชื่อเมท็อดที่ใช้ทดสอบ																												
	# Constructor	1	1	4	1	0	18	5	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- Form1_KeyDown	1	5	12	6	2	145	16	1	2	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- InitializeComponent	1	1	55	1	0	396	71	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M1
	- timer1_Tick	1	2	6	2	1	41	5	1	2	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ Dispose	0	3	1	4	1	23	3	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																												
	- components_3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- listBox1_3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- timer1_3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- difficultyProgressBar_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- correctLabel_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- missedLabel_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- accuracyLabel_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- random_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- totalLabel_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- stats_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- statusStrip1_1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- toolStripStatusLabel1_1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ค่าที่วัดได้											0	30	7	1	1	1	12	0	0	5	32							C1
2)	Program																												
	ชื่อเมท็อดที่ใช้ทดสอบ																												
	- Main	0	1	4	1	0	10	3	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ค่าที่วัดได้											0	3	1	1	0	0	0	0	0	1	50							
3)	Stats																												
	ชื่อเมท็อดที่ใช้ทดสอบ																												
	# Constructor	1	1	1	1	0	16	4	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ Update	1	2	0	3	1	80	11	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ get_Current	0	1	0	1	0	8	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																												
	+ Correct	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ Accuracy	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ Total	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ Missed	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ค่าที่วัดได้											4	0	1	1	0	1	5	0	0	3	0							
รวมทั้งสิ้น																							0	0	2	1	0	0	

ตารางที่ 5.11 ตัวอย่างผลลัพธ์แสดงค่าการแปลความหลังจากการปรับปรุงโค้ดกลุ่มร่องรอยไม่ดีครั้งที่ 1

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		Hit_the_keys			
	ชุดรายการ			Defuzzification Output	สรุปค่าแปลความ
ลำดับที่	คลาส	เมท็อด	แอตทริบิวต์		
1	Form1	Constructor	components 3	0.303	Clean
2		Form1_KeyDown	listBox1 3	0.324	Clean
3		InitializeComponent	timer1 3	0.339	Ambigous
4		timer1_Tick	difficultyProgressBar 2	0.303	Clean
5		Dispose	correctLabel 2	0.327	Clean
6			missedLabel 2		
7			accuracyLabel 2		
8			random 2		
9			totalLabel 2		
10			stats 2		
11			statusStrip1 1		
12			toolStripStatusLabel1 1		
13	Program	Main		0.263	Clean
14	Stats	Constructor	Correct	0.799	Bad
15		Update	Accuracy	0.799	Bad
16		get_Current	Total	0.799	Bad
17			Missed	0.799	Bad

จากตารางที่ 5.11 ผลลัพธ์ของการแปลความพบว่ามีคลาส Stats ถูกระบุให้เป็นโค้ดที่มีร่องรอยไม่ดี โดยดูผลลัพธ์ตามตารางแสดงให้เห็นว่า จำเป็นต้องปรับปรุงโค้ดอีกครั้ง ด้วยวิธีการปฏิบัติและเหตุผลที่เลือกใช้แก้ไขตามรายละเอียดดังตารางที่ 5.12 จากนั้นจึงทำการแก้ไขตามวิธีการปฏิบัติที่ออกแบบดังผลลัพธ์ตามตารางที่ 5.13 จากนั้นให้พิจารณาผลลัพธ์การแปลความตามตารางที่ 5.14 จะเห็นว่าหลังจากที่ปรับปรุงโค้ดครั้งที่ 2 แล้ว ผลลัพธ์ได้เป็นคลีนโค้ดทั้งหมด โดยผู้วิจัยจะตรวจสอบทุกคลาส ทุกเมท็อด และทุกแอตทริบิวต์ว่ามีค่าเป็นคลีนโค้ด ซึ่งค่าการแปลความตามตารางมีค่าเป็นคลีนโค้ดทั้งหมด จึงจะถือว่ากระบวนการการปรับปรุงโค้ดสิ้นสุดเพียงเท่านี้และจัดให้กลุ่มโค้ดตัวอย่างชุดนี้เป็นซับเซตของคลีนโค้ดตามที่ได้ออกแบบและกำหนดไว้

ตารางที่ 5.12 วิธีการปฏิบัติเพื่อปรับปรุงโค้ดกลุ่มร่องรอยไม่ดีครั้งที่ 2

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			Hit_the_keys		
ลำดับที่	คลาส	เมท็อด	แอตทริบิวต์	ผลลัพธ์การแปลความ	สรุปค่าแปลความ
1	Stats	Constructor	Correct	0.799	Bad
2		Update	Accuracy	0.799	Bad
3		get_Current	Total	0.799	Bad
4			Missed	0.799	Bad
ประเภทร่องรอยไม่ดีที่ต้องการแก้ไข			Temporary Field		

ตารางที่ 5.12 วิธีการปฏิบัติเพื่อปรับปรุงโค้ดกลุ่มร่องรอยไม่ดีครั้งที่ 2

วิธีการรีแฟคทอริ่งสำหรับการแก้ไขร่องรอยไม่ดี	
1. Moving Features between Objects วิธีย่อยที่ใช้ Extract Class 2. Simplifying Conditional Expression วิธีย่อยที่ใช้ Introduce Null Object 3. Organizing Data วิธีย่อยที่ใช้ Encapsulate Field	
วิธีที่เลือกใช้ในการปรับปรุงร่องรอยไม่ดี	Organizing Data วิธีย่อยที่ใช้ Encapsulate Field
เหตุผลที่เลือกใช้	
เมื่อแก้ไขปัญหาแรกเสร็จแล้ว จึงค่อยแก้ไขปัญหาเรื่องแอตทริบิวต์ที่มีการเรียกใช้ เนื่องจากแอตทริบิวต์ที่เรียกใช้งานถูกสร้างขึ้น เพื่อเก็บค่าชั่วคราว เท่านั้น ดังนั้นควรใช้วิธีการ Encapsulate Field เพื่อเป็นการป้องกันการแก้ไขค่าโดยตรงและปกปิดข้อมูลให้กับโปรแกรมมิ วิธีการปฏิบัติอื่น เช่น Extract Class ไม่เหมาะสม เนื่องจากคลาสมีหน้าที่รับผิดชอบเพียงอย่างเดียวการแบ่งแยกอาจนำไปสู่ร่องรอยไม่ดีประเภท Lazy Class ได้ และการเลือก ใช้ Introduce Null Object จึงไม่เหมาะสมกับกรณีนี้ เพราะแอตทริบิวต์ภายในคลาสนี้เป็นประเภทตัวเลข	

ตารางที่ 5.13 ตัวอย่างผลลัพธ์หลังจากการปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี ครั้งที่ 2

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ	Hit_the_keys																												
จำนวนคลาสที่ใช้ทดสอบ	3					จำนวนเมท็อดที่ใช้ทดสอบ						17								จำนวนแอตทริบิวต์ที่ใช้ทดสอบ					17				
	มาตรวัดที่ใช้ในการทดสอบ																												
ชื่อคลาสที่ใช้ทดสอบ	ACML	CC	ECML	ILCC	ILND	NILI	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-OMF	SEC-IM	ข้อจำกัด
1) Form1																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
# Constructor	1	1	4	1	0	18	5	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- Form1_KeyDown	1	5	16	6	2	145	16	1	2	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- InitializeComponent	1	1	55	1	0	396	71	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M1
- timer1_Tick	1	2	6	2	1	41	5	1	2	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Dispose	0	3	1	4	1	23	3	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
- components_3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- listBox1_3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- timer1_3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- difficultyProgressBar_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- correctLabel_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- missedLabel_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- accuracyLabel_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- random_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- totalLabel_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- stats_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- statusStrip1_1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- toolStripStatusLabel1_1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	30	7	1	1	1	12	0	0	7	31							C1
2) Program																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
- Main	0	1	4	1	0	10	3	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	3	1	1	0	0	0	0	0	1	50							
3) Stats																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
# Constructor	1	1	1	1	0	18	5	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_Total	2	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ dumpResult	1	1	1	1	0	41	6	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Counter	1	1	0	1	0	8	1	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Update	1	1	3	1	0	24	4	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ isCorrectKey	1	2	0	3	1	23	3	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_Missed	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_Accuracy	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_Correct	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ getCurrentState	0	1	0	1	0	8	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ ResetCounter	0	1	0	1	0	5	1	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
- _missed_5	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- _correct_5	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- _accuracy_4	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- _total_4	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- _currentState_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	1	1	1	1	1	5	0	0	11	0							
รวมทั้งสิ้น																							0	0	2	1	0	0	

ตารางที่ 5.14 ตัวอย่างผลลัพธ์แสดงค่าการแปลความหลังจากการปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี ครั้งที่ 2

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			Hit_the_keys		
	ชุดรายการ			Defuzzification	สรุปค่าแปลความ
ลำดับที่	คลาส	เมธอด	ฟิลด์	Output	
1	Form1	Constructor	components 3	0.303	Clean
2		Form1_KeyDown	listBox1 3	0.324	Clean
3		InitializeComponent	timer1 3	0.317	Clean
		InitFirst	difficultyProgressBar 2	0.317	Clean
		InitForm	correctLabel 2	0.321	Clean
4		timer1_Tick	missedLabel 2	0.303	Clean
5		Dispose	accuracyLabel 2	0.327	Clean
6			random 2		
7			totalLabel 2		
8			stats 2		
9			statusStrip1 1		
10			toolStripStatusLabel1 1		
11					
12					
13	Program	Main		0.263	Clean
14	Stats	Constructor	_missed 5	0.306	Clean
15		get_Total	_correct 5	0.306	Clean
16		dumpResult	_accuracy 4	0.306	Clean
17		Counter	_total 4	0.306	Clean
18		Update	_currentState 2	0.306	Clean
19		isCorrectKey		0.306	Clean
20		get_Missed		0.306	Clean
21		get_Accuracy		0.306	Clean
22		get_Correct		0.306	Clean
23		getCurrentState		0.329	Clean
24		ResetCounter		0.329	Clean

ผลลัพธ์การแปลตามตารางที่ 5.14 พบว่าค่าแปลความทุกค่าในคอลัมน์สรุปค่าแปลความมีค่าเป็นซับเซตคลีนโค้ด จึงถือเป็นการสิ้นสุดการปรับปรุงคุณภาพโค้ดตามวิธีการที่ออกแบบไว้

จากการทดลองโดยใช้กลุ่มโค้ดตัวอย่างชุดเริ่มต้น ผู้วิจัยพบชนิดของร่องรอยไม่ดีและชนิดของความคลุมเครือ โดยมีรายละเอียดของประเภทร่องรอยไม่ดี ประเภทความคลุมเครือ จำนวนครั้งในการปรับปรุงโค้ดและเทคนิครีแฟคทอริ่งที่ใช้เพื่อปรับปรุง สรุปได้ตามตารางที่ 5.15 และตารางที่ 5.16

ตารางที่ 5.15 รายการชุดทดสอบเริ่มต้นกลุ่มร่องรอยไม่ดีที่ตรวจพบ

ลำดับที่	ตัวอย่างการทดสอบร่องรอยไม่ดี	จำนวนรอบแก้ไข	เทคนิคการแก้ไข	ประเภทร่องรอยไม่ดีและความคลุมเครือที่แก้ไข
1	Converting Hexadecimal	1	Introduce Null Object	Temporary Field
2	DirectoryInfoExtensions	1	Add Comment	Bad Comment
		2	Encapsulate Field	Lazy Class
3	QueueExample1	1	Extract Method	Long Method
4	Assignment1MH_Base	1	Add Comment	Bad Comment
		2	Extract Class	Large Classes
		3	Remove Parameter	Ambiguity in Parameter
5	EventedListExample	1	Extract Subclass	Lazy Class, Large Classes และ Temporary Field
		2	Extract Class	Large Classes
6	ImmutableCollections	1	Extract Class	Large Classes
7	Feature Envy	1	Add Comment	Bad Comment
		2	Move Method	FeatureEnvy และ Lazy Class
		3	Collapse Hierarchy	FeatureEnvy
8	inappropriateintimacy	1	Add Comment	Bad Comment
		2	Move Field	Inappropriate Intimacy
9	LAZY_CLASS1	1	Add Comment	Bad Comment
		2	Inner Class	Lazy Class

ตารางที่ 5.15 รายการชุดทดสอบเริ่มต้นกลุ่มร่องรอยไม่ดีที่ตรวจพบ (ต่อ)

ลำดับที่	ตัวอย่างการทดสอบร่องรอยไม่ดี	จำนวนรอบแก้ไข	เทคนิคการแก้ไข	ประเภทร่องรอยไม่ดีและความคลุมเครือที่แก้ไข
10	LAZY_CLASS2	1	Add Comment	Bad Comment
		2	Pull up	Lazy Class
11	LONG_PARAMETER_LISTS	1	Add Comment	Bad Comment
		2	Introduce Parameter Object	Long Parameter Lists
12	Mixed_Messages	1	Introduce Object Parameter	Lazy Class และ Temporary Field
		2	Collapse Hierarchy	Lazy Class
13	SimpleLinqToXml	1	Add Comment	Bad Comment
		2	Extract Method	Long Method
14	Party Planner 2	1	Encapsulate Field	Temporary Field
		2	Extract Method	Ambiguity in Method
15	PartialClassAddIn	1	Move Field	Inappropriate Intimacy
16	Using Delegates	1	Add Comment	Bad Comment
		2	Inner Class	Lazy Class
17	Hit_the_keys	1	Remove parameter to whole object	Long Parameter List
		2	Encapsulate Field	Temporary Field
18	List_of_Ducks	1	Move Method	FeatureEnvy และ Temporary Field
19	Sloppy_Joe	1	Introduce Whole Object	FeatureEnvy และ Temporary Field
20	TallGuy	1	Introduce Object Parameter	Temporary Field

ตารางที่ 5.16 รายการชุดทดสอบเริ่มต้นกลุ่มโค้ดที่มีความคลุมเครือที่ตรวจพบ

ลำดับที่	ตัวอย่างการทดสอบความคลุมเครือ	จำนวนรอบแก้ไข	เทคนิคการแก้ไข	ประเภทความคลุมเครือที่แก้ไข
1	PartialClassInterfaces	1	Encapsulate Field	Ambiguity in Encapsulation
2	TaxApp	1	Inline Method	Ambiguity in Method
3	UnhandledThreadException	1	Encapsulate Field	Ambiguity in Encapsulation
4	UnhandledWPFException	1	Add Comment	Ambiguity in Comment
5	WebBrowser	1	Add Comment	Ambiguity in Comment
6	Joe_and_Bob	1	Encapsulate Field	Ambiguity in Encapsulation
7	Mileage_calculator	1	Inline Method	Ambiguity in Method
8	Time_to_start_coding	1	Inline Method	Ambiguity in Method
9	BinaryWriter	1	Extract Method	Ambiguity in Method
10	LINQ to XML	1	Extract Method	Ambiguity in Method
11	OfficeSample	1	Extract Method	Ambiguity in Method
12	LinqToNorthwind	1	Seal Classes	Ambiguity in Class Organization
13	ObjectDumper	1	Extract Method	Ambiguity in Method
		2	Seal Classes	Ambiguity in Class Organization
14	PasteXmlAsLinq	1	Extract Method	Ambiguity in Method
		2	Seal Classes	Ambiguity in Class Organization

ตารางที่ 5.16 รายการชุดทดสอบเริ่มต้นกลุ่มโค้ดที่มีความคลุมเครือที่ตรวจพบ (ต่อ)

ลำดับที่	ตัวอย่างการทดสอบความคลุมเครือ	จำนวนรอบแก้ไข	เทคนิคการแก้ไข	ประเภทความคลุมเครือที่แก้ไข
15	PropertyBag	1	Seal Classes	Ambiguity in Class Organization
16	Rss	1	Add Comment	Ambiguity in Comment
17	SimpleLambdas	1	Seal Classes	Ambiguity in Class Organization
18	WinFormsDataBinding	1	Seal Classes	Ambiguity in Class Organization
19	XMLdoc	1	Encapsulate Field	Ambiguity in Encapsulation
20	XQuery	1	Remove Parameter	Ambiguity in Arguments

จากตารางที่ 5.15 ผลลัพธ์การแก้ไขตามวิธีปฏิบัติที่กำหนดไว้สามารถแก้ไขร่องรอยไม่ดีที่และโค้ดที่มีความคลุมเครือ ซึ่งเมื่อตรวจพบแล้วจึงปรับปรุงโค้ดจนกระทั่งเป็นซับเซตของคลีนโค้ดตามวิธีการที่ออกแบบไว้ มีบางกรณีที่ตรวจไม่พบความคลุมเครือหลังจากแก้ไขร่องรอยไม่ดีที่เกิดขึ้น ซึ่งอาจมีโอกาสเกิดขึ้นได้ไม่มากนัก หลังจากแก้ไขชุดทดสอบเริ่มต้นในกลุ่มร่องรอยไม่ดีจะแก้ไขชุดทดสอบในที่มีความคลุมเครือด้วยวิธีการปฏิบัติที่ออกแบบเพื่อทำการสรุปผลต่อไป ส่วนในตารางที่ 5.16 หากพิจารณาผลลัพธ์ที่ได้จากการแก้ไขพบว่า สามารถแก้ไขความคลุมเครือที่ตรวจพบและปรับปรุงจนกระทั่งเป็นซับเซตของคลีนโค้ดตามวิธีการที่ออกแบบไว้ โดยส่วนมากความคลุมเครือจะถูกปรับปรุงเพียงครั้งเดียวก็สามารถเป็นซับเซตของคลีนโค้ดได้ แสดงให้เห็นว่าผู้วิจัยออกแบบวิธีการปฏิบัติไว้อย่างรัดกุมและมีข้อมูลสนับสนุนการตัดสินใจแก้ไขปัญหาที่เกิดขึ้น ทำให้ประหยัดเวลาในการแก้ไข ซึ่งเป็นไปตามวิทยานิพนธ์นี้ที่ให้ความสำคัญกับความสามารถในการบำรุงรักษา

5.5 ผลการทดสอบค่าดัชนีความสามารถบำรุงรักษาก่อนการปรับปรุงและกลุ่มโค้ดที่ผ่านการปรับปรุง

เมื่อรวบรวมกลุ่มโค้ดภาษา C# เพื่อใช้ในการทดสอบเริ่มต้นจำนวน 60 ชุดและใช้สำหรับการทวนสอบจำนวน 60 ชุด จากนั้นนำเฉพาะกลุ่มโค้ดสำหรับทดสอบเริ่มต้นมาทดสอบค่าดัชนีความสามารถบำรุงรักษาก่อนการปรับปรุง ตามขั้นตอนที่ 4.1.4 ซึ่งผลทดสอบได้ค่าดัชนีความสามารถบำรุงรักษาดังตารางที่ 5.17 ตารางที่ 5.18 และ ตารางที่ 5.19

ตารางที่ 5.17 ค่าดัชนีความสามารถบำรุงรักษาก่อนปรับปรุงโค้ดกลุ่มซับเซตคลีนโค้ด

กลุ่มซับเซตคลีนโค้ด				
ลำดับที่	**รายการตัวอย่างทดสอบ**	**ค่าดัชนีบำรุงรักษาที่ไม่ถูกนับรวมคำอธิบายโปรแกรม (Miwoc)**	**ค่าดัชนีบำรุงรักษาที่มีการคำนวณคำอธิบายโปรแกรม (Miwc)**	**ค่าดัชนีความสามารถในการบำรุงรักษา (MI)**
1	Encryption and Decryption	79	5	118.11
2	PrimeGenerator	76	24	95.55
3	WeekValidator	76	1	122.24
4	Set Collections	77	60	85.07
5	OperatorOverloading	76	1	122.24
6	ProtectedSnflar	88	1	142.76
7	Generics_CSharp	77	32	93.25
8	Pool_Puzzle	72	1	115.40
9	Playing Card	88	38	109.66
10	Secret_Ingredients	84	30	106.12
11	Fingers the Clown	81	30	100.99
12	PlanetMission	78	31	95.40
13	Baseball	78	27	97.34
14	Go Fish	77	11	107.10

ตารางที่ 5.17 ค่าดัชนีความสามารถบำรุงรักษาก่อนปรับปรุงโค้ดกลุ่มซับเซตคลีนโค้ด (ต่อ)

กลุ่มซับเซตคลีนโค้ด				
ลำดับที่	**รายการตัวอย่างทดสอบ**	**ค่าดัชนีบำรุงรักษาที่ไม่ถูกนับรวมคำอธิบายโปรแกรม (Miwoc)**	**ค่าดัชนีบำรุงรักษาที่มีการคำนวณคำอธิบายโปรแกรม (Miwc)**	**ค่าดัชนีความสามารถในการบำรุงรักษา (MI)**
15	TravellingSalesmanProblem	69	7	98.07
16	ADO	72	8	101.91
17	MutexFun	64	8	88.23
18	NamedPipes	56	1	88.04
19	JewelThief	89	1	144.47
20	Let's Build a House	81	19	107.25

ตารางที่ 5.18 ค่าดัชนีความสามารถบำรุงรักษาก่อนปรับปรุงโค้ดกลุ่มโค้ดที่มีความคลุมเครือ

กลุ่มโค้ดที่มีความคลุมเครือ				
ลำดับที่	**รายการตัวอย่างทดสอบ**	**ค่าดัชนีบำรุงรักษาที่ไม่ถูกนับรวมคำอธิบายโปรแกรม (Miwoc)**	**ค่าดัชนีบำรุงรักษาที่มีการคำนวณคำอธิบายโปรแกรม (Miwc)**	**ค่าดัชนีความสามารถในการบำรุงรักษา (MI)**
1	PartialClassInterfaces	71	1	113.69
2	TaxApp	70	35	79.17
3	UnhandledThreadException	77	51	87.09
4	UnhandledWPFException	82	15	111.13
5	WebBrowser	75	34	88.98
6	Joe_and_Bob	71	25	86.44

ตารางที่ 5.18 ค่าดัชนีความสามารถบำรุงรักษาก่อนปรับปรุงโค้ดกลุ่มโค้ดที่มีความคลุมเครือ (ต่อ)

กลุ่มโค้ดที่มีความคลุมเครือ				
ลำดับที่	**รายการตัวอย่างทดสอบ**	**ค่าดัชนีบำรุงรักษาที่ไม่ถูกนับรวมคำอธิบายโปรแกรม (Miwoc)**	**ค่าดัชนีบำรุงรักษาที่มีการคำนวณคำอธิบายโปรแกรม (Miwc)**	**ค่าดัชนีความสามารถในการบำรุงรักษา (MI)**
7	Mileage_calculator	67	25	79.60
8	Time_to_start_coding	71	39	80.41
9	BinaryWriter	55	1	86.33
10	LINQ to XML	62	1	98.30
11	OfficeSample	77	5	114.69
12	LinqToNorthwind	77	12	105.76
13	ObjectDumper	65	1	103.43
14	PasteXmlAsLinq	66	23	79.04
15	PropertyBag	82	35	100.66
16	Rss	71	1	113.69
17	SimpleLambdas	80	7	116.88
18	WinFormsDataBinding	56	39	53.89
19	XMLdoc	85	1	137.63
20	XQuery	62	4	90.77

ตารางที่ 5.19 ค่าดัชนีความสามารถบำรุงรักษาก่อนปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี

กลุ่มร่องรอยไม่ดี				
ลำดับที่	**รายการตัวอย่างทดสอบ**	**ค่าดัชนีบำรุงรักษาที่ไม่ถูกนับรวมคำอธิบายโปรแกรม (Miwoc)**	**ค่าดัชนีบำรุงรักษาที่มีการคำนวนคำอธิบายโปรแกรม (Miwc)**	**ค่าดัชนีความสามารถในการบำรุงรักษา (MI)**
1	Converting Hexadecimal	58	34.5	59.71
2	DirectoryInfoExtensions	74.5	0	127.40
3	QueueExample1	74.5	30	89.88
4	Assignment1MH_Base	54	28	55.79
5	EventedListExample	85.66	8	125.26
6	ImmutableCollections	74	44	83.74
7	FeatureEnvy	79.66	22	103.00
8	inappropriateintimacy	83.33	30	104.98
9	LAZY_CLASS1	82	26	104.70
10	LONG_PARAMETER_LISTS	84	0	143.64
11	SimpleLinqToXml	66	0	112.86
12	Party Planner 2	71.5	23	88.44
13	PartialClassAddIn	65	42	68.97
14	Using Delegates	85	33.33	106.36
15	Hit_the_keys	66	27	76.82
16	List_of_Ducks	75.8	2.6	117.26
17	Sloppy_Joe	67	26.33	68.33
18	TallGuy	79.66	30	98.70
19	LAZY_CLASS2	84.66	0	144.77
20	Mixed_Messages	85.5	3.5	131.92

จากทั้ง 3 ตารางจะเห็นได้ว่า การทดสอบจะแสดงค่าดัชนีความสามารในการบำรุงรักษาก่อนที่จะทำการปรับปรุงตามวิธีการปฏิบัติเพื่อปรับปรุงโค้ดที่ได้สร้างไว้ มีข้อสังเกตได้2 ประเด็นคือ

1) โค้ดกลุ่มซับเซตคลีนโค้ดมีค่าดัชนีความสามารถในการบำรุงรักษาอยู่ในเกณฑ์ที่ดี

2) กลุ่มโค้ดที่นำมาใช้ทดสอบชนิดกลุ่มร่องรอยไม่ดีประเภท Lazy Class มักจะมีระดับของค่าดัชนีความสามารถในการบำรุงรักษาสูง ซึ่งจะขออธิบายสาเหตุไว้ในส่วนของการวัดค่าหลังการปรับปรุงโค้ด

เมื่อผ่านขั้นตอนการจำแนกและปรับปรุงโค้ดเรียบร้อยแล้ว จากนั้นจึงทดสอบค่าดัชนีชี้วัดความสามารถในการบำรุงรักษากับกลุ่มโค้ดที่ผ่านกระบวนการปรับปรุงให้มีคุณสมบัติที่เป็นซับเซตคลีนโค้ดตามเกณฑ์ที่กำหนดแล้ว ผลการตรวจสอบค่าดัชนีความสามารถในการบำรุงรักษาอีกครั้ง พบว่า ค่าในกลุ่มของความคลุมเครือและร่องรอยไม่ดีมีระดับความสามารถในการบำรุงรักษาที่ดีขึ้น ดังตารางการเปรียบเทียบค่าดัชนีค่าการบำรุงรักษาก่อนและหลังการจำแนกและปรับปรุงโค้ดตามตารางที่5.20 ตารางที่ 5.21 และ ตารางที่5.22

ตารางที่ 5.20การเปรียบเทียบค่าดัชนีค่าการบำรุงรักษาก่อนและหลังกลุ่มซับเซตคลีนโค้ด

ลำดับที่	ชื่อตัวอย่างทดสอบ	ชนิดที่ตรวจสอบได้	ค่าดัชนีบำรุงรักษาก่อนการปรับปรุง	ระดับค่าดัชนีบำรุงรักษา
1	Encryption and Decryption	คลีนโค้ด	118.11	สูง
2	PrimeGenerator	คลีนโค้ด	95.55	สูง
3	WeekValidator	คลีนโค้ด	122.24	สูง
4	Set Collections	คลีนโค้ด	85.07	สูง
5	OperatorOverloading	คลีนโค้ด	122.24	สูง
6	ProtectedSnflar	คลีนโค้ด	142.76	สูง
7	Generics_CSharp	คลีนโค้ด	93.25	สูง
8	Pool_Puzzle	คลีนโค้ด	115.40	สูง
9	Playing Card	คลีนโค้ด	109.66	สูง
10	Secret_Ingredients	คลีนโค้ด	106.12	สูง
11	Fingers the Clown	คลีนโค้ด	100.99	สูง

ตารางที่ 5.20การเปรียบเทียบค่าดัชนีค่าการบำรุงรักษาก่อนและหลังกลุ่มซับเซตคลีนโค้ด(ต่อ)

ลำดับที่	ชื่อตัวอย่างทดสอบ	ชนิดที่ตรวจสอบได้	ค่าดัชนีบำรุงรักษาก่อนการปรับปรุง	ระดับค่าดัชนีบำรุงรักษา
12	PlanetMission	คลีนโค้ด	95.40	สูง
13	Baseball	คลีนโค้ด	97.34	สูง
14	Go Fish	คลีนโค้ด	107.10	สูง
15	TravellingSalesmanProblem	คลีนโค้ด	98.07	สูง
16	ADO	คลีนโค้ด	101.91	สูง
17	MutexFun	คลีนโค้ด	88.23	สูง
18	NamedPipes	คลีนโค้ด	88.04	สูง
19	JewelThief	คลีนโค้ด	144.47	สูง
20	Let’s Build a House	คลีนโค้ด	107.25	สูง

ตารางที่ 5.21 การเปรียบเทียบค่าดัชนีค่าการบำรุงรักษาก่อนและหลังการปรับปรุงกลุ่มโค้ดที่มีความคลุมเครือ

ลำดับที่	ชื่อตัวอย่างทดสอบ	ชนิดของความคลุมเครือที่ได้รับการปรับปรุง	ค่าดัชนีบำรุงรักษาก่อนการปรับปรุง		ค่าดัชนีบำรุงรักษาหลังการปรับปรุง		ดัชนีความสามารถในการบำรุงรักษา
1	PartialClassInterfaces	Ambiguity in Classes	113.69	สูง	127.40	สูง	เพิ่มขึ้น
2	TaxApp	Ambiguity in Classes	79.17	ปานกลาง	79.17	ปานกลาง	ไม่เปลี่ยนแปลง
3	UnhandledThreadException	Ambiguity in Encapsulation	87.09	สูง	87.58	สูง	เพิ่มขึ้น
4	UnhandledWPFException	Ambiguity in Comment	111.13	สูง	93.24	สูง	ลดลง
5	WebBrowser	Ambiguity in Comment	88.98	สูง	85.75	สูง	ลดลง
6	Joe_and_Bob	Ambiguity in Classes	86.44	สูง	86.44	สูง	ไม่เปลี่ยนแปลง
7	Mileage_calculator	Ambiguity in Classes	79.60	ปานกลาง	79.60	ปานกลาง	ไม่เปลี่ยนแปลง
8	Time_to_start_coding	Ambiguity in Encapsulation	80.41	ปานกลาง	81.27	ปานกลาง	เพิ่มขึ้น
9	BinaryWriter	Ambiguity in Method	86.33	สูง	116.28	สูง	เพิ่มขึ้น
10	LINQ to XML	Ambiguity in Method	98.30	สูง	123.12	สูง	เพิ่มขึ้น

ตารางที่ 5.21 การเปรียบเทียบค่าดัชนีค่าการบำรุงรักษาก่อนและหลังการปรับปรุงกลุ่มโค้ดที่มีความคลุมเครือ(ต่อ)

ลำดับที่	ชื่อตัวอย่างทดสอบ	ชนิดของความคลุมเครือที่ได้รับการปรับปรุง	ค่าดัชนีบำรุงรักษาก่อนการปรับปรุง		ค่าดัชนีบำรุงรักษาหลังการปรับปรุง		ดัชนีความสามารถในการบำรุงรักษา
11	OfficeSample	Ambiguity in Method และ Ambiguity in Class Organization	114.69	สูง	123.23	สูง	เพิ่มขึ้น
12	LinqToNorthwind	Ambiguity in Class Organization	105.76	สูง	105.99	สูง	เพิ่มขึ้น
13	ObjectDumper	Ambiguity in Method และ Ambiguity in Class Organization	103.43	สูง	114.57	สูง	เพิ่มขึ้น
14	PasteXmlAsLinq	Ambiguity in Method และ Ambiguity in Class Organization	79.04	ปานกลาง	83.69	ปานกลาง	เพิ่มขึ้น
15	PropertyBag	Ambiguity in Class Organization	100.66	สูง	104.87	สูง	เพิ่มขึ้น

ตารางที่ 5.21 การเปรียบเทียบค่าดัชนีค่าการบำรุงรักษาก่อนและหลังการปรับปรุงกลุ่มโค้ดที่มีความคลุมเครือ(ต่อ)

ลำดับที่	ชื่อตัวอย่างทดสอบ	ชนิดของความคลุมเครือที่ได้รับการปรับปรุง	ค่าดัชนีบำรุงรักษาก่อนการปรับปรุง		ค่าดัชนีบำรุงรักษาหลังการปรับปรุง		ดัชนีความสามารถในการบำรุงรักษา
16	Rss	Ambiguity in Comment	113.69	สูง	92.37	สูง	ลดลง
17	SimpleLambdas	Ambiguity in Class Organization	116.88	สูง	116.88	สูง	ไม่เปลี่ยนแปลง
18	WinFormsDataBinding	Ambiguity for Change	53.89	ต่ำ	73.30	ปานกลาง	เพิ่มขึ้น
19	XMLdoc	Ambiguity in Encapsulation	137.63	สูง	147.06	สูง	เพิ่มขึ้น
20	Xquery	Ambiguity in Classes	90.77	สูง	94.19	สูง	เพิ่มขึ้น

ตารางที่ 5.22 การเปรียบเทียบค่าดัชนีค่าการบำรุงรักษาก่อนและหลังการปรับปรุงกลุ่มร่องรอยไม่ดี

ลำดับที่	ชื่อตัวอย่างทดสอบ	ชนิดของร่องรอยไม่ดีและความคลุมเครือที่ได้รับการปรับปรุง	ค่าดัชนีบำรุงรักษาก่อนการปรับปรุง		ค่าดัชนีบำรุงรักษาหลังการปรับปรุง		ดัชนีความสามารถในการบำรุงรักษา
1	Converting Hexadecimal	Temporary Field	59.71	ต่ำ	72.26	ปานกลาง	เพิ่มขึ้น

ตารางที่ 5.22 การเปรียบเทียบค่าดัชนีค่าการบำรุงรักษาก่อนและหลังการปรับปรุงกลุ่มร่องรอยไม่ดี(ต่อ)

ลำดับที่	ชื่อตัวอย่างทดสอบ	ชนิดของร่องรอยไม่ดีและความคลุมเครือที่ได้รับการปรับปรุง	ค่าดัชนีบำรุงรักษาก่อนการปรับปรุง		ค่าดัชนีบำรุงรักษาหลังการปรับปรุง		ดัชนีความสามารถในการบำรุงรักษา
2	DirectoryInfoExtensions	Lazy Class และ Ambiguity in Comment	127.40	สูง	101.85	สูง	ลดลง
3	QueueExample1	Long Method	55.79	ต่ำ	70.50	ปานกลาง	เพิ่มขึ้น
4	Assignment1MH_Base	Large Classes และ Ambiguity in Parameter	108.09	สูง	110.69	สูง	เพิ่มขึ้น
5	EventedListExample	Lazy Class, Large Classes และ Temporary Field	125.26	สูง	122.26	สูง	ลดลง
6	ImmutableCollections	Large Classes	83.74	ปานกลาง	85.29	สูง	เพิ่มขึ้น
7	FeatureEnvy	FeatureEnvy และ Lazy Class	103.00	สูง	113.26	สูง	เพิ่มขึ้น
8	inappropriateintimacy	Inappropriate Intimacy	104.98	สูง	104.41	สูง	ลดลง

ตารางที่ 5.22 การเปรียบเทียบค่าดัชนีค่าการบำรุงรักษาก่อนและหลังการปรับปรุงกลุ่มร่องรอยไม่ดี(ต่อ)

ลำดับที่	ชื่อตัวอย่างทดสอบ	ชนิดของร่องรอยไม่ดีและความคลุมเครือที่ได้รับการปรับปรุง	ค่าดัชนีบำรุงรักษาก่อนการปรับปรุง		ค่าดัชนีบำรุงรักษาหลังการปรับปรุง		ดัชนีความสามารถในการบำรุงรักษา
9	LAZY_CLASS1	Lazy Class	104.70	สูง	108.12	สูง	เพิ่มขึ้น
10	LAZY_CLASS2	Lazy Class	144.77	สูง	108.38	สูง	ลดลง
11	LONG_PARAMETER_LISTS	Long Parameter List	143.64	สูง	105.22	สูง	ลดลง
12	Mixed_Messages	Lazy Class และ Temporary Field	131.92	สูง	93.24	สูง	ลดลง
13	SimpleLinqToXml	Inappropriate Intimacy, Long Method และ Ambiguity in Comment	112.86	สูง	86.57	สูง	ลดลง
14	Party Planner 2	Temporary Field และ Ambiguity in Method	88.44	สูง	90.45	สูง	เพิ่มขึ้น
15	PartialClassAddIn	Inappropriate Intimacy	68.97	ปานกลาง	72.88	ปานกลาง	เพิ่มขึ้น

ตารางที่ 5.22 การเปรียบเทียบค่าดัชนีค่าการบำรุงรักษาก่อนและหลังการปรับปรุงกลุ่มร่องรอยไม่ดี(ต่อ)

ลำดับที่	ชื่อตัวอย่างทดสอบ	ชนิดของร่องรอยไม่ดีและความคลุมเครือที่ได้รับการปรับปรุง	ค่าดัชนีบำรุงรักษาก่อนการปรับปรุง		ค่าดัชนีบำรุงรักษาหลังการปรับปรุง		ผลการปรับปรุง
16	Using Delegates	Lazy Class	106.36	สูง	118.84	สูง	ลดลง
17	Hit_the_keys	Longparameter List และ Temporary Field	76.82	ปานกลาง	88.63	สูง	เพิ่มขึ้น
18	List_of_Ducks	FeatureEnvy และ Temporary Field	117.26	สูง	120.68	สูง	เพิ่มขึ้น
19	Sloppy_Joe	FeatureEnvy และ Temporary Field	68.33	ปานกลาง	97.52	สูง	เพิ่มขึ้น
20	TallGuy	Temporary Field	98.70	สูง	99.28	สูง	เพิ่มขึ้น

จากตารางการเปรียบเทียบค่าดัชนีความสามารถในการบำรุงรักษา การนำโค้ดทุกตัวผ่านวิธีปรับปรุง โค้ดกลุ่มร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือ โดยใช้เทคนิครีแฟคทอริ่งเพื่อเพิ่มค่าความสามารถในการบำรุงรักษา ทำให้พบว่า

1) ค่าดัชนีความสามารถในการบำรุงรักษาที่เพิ่มขึ้น คือ โค้ดร่องรอยไม่ดีและความคลุมเครือถูกปรับปรุงด้วยเทคนิครีแฟคทอริ่ง ทำให้ค่าดัชนีความสามารถในการบำรุงรักษาดีขึ้น

2) ค่าดัชนีความสามารถในการบำรุงรักษาที่ลดลง จากการทดลองพบว่า เมื่อทำการปรับปรุงกลุ่มโค้ดตัวอย่างมีค่าดัชนีความสามารถในการบำรุงรักษาลดน้อยลง ผู้วิจัยวิเคราะห์ว่า ค่าที่ลดลงนั้น ไม่ได้เกิดจากการปรับปรุงโค้ด แต่เป็นการสะท้อนค่าจริงของกลุ่มตัวอย่างนั้นออกมา เช่น กลุ่มร่องรอยไม่ดี lazy class มีคลาสที่ไม่ได้ถูกเรียกใช้งานจำนวนหนึ่ง เมื่อทำการวัดค่าเฉลี่ยด้วยค่าดัชนีความสามารถในการบำรุงรักษาพบว่ามีค่าสูงขึ้นหลังปรับปรุง เพราะมีการยุบรวมหรือตัดคลาสที่ไม่ใช้งานออก เมื่อวัดค่าดัชนีความสามารถในการบำรุงรักษาทำให้ได้ผลลัพธ์ที่ต่ำลง

3) ค่าดัชนีความสามารถในการบำรุงรักษาที่มีผลลัพธ์ไม่เปลี่ยนแปลงเกิดจาก2 สาเหตุ คือ

(1) กลุ่มทดสอบประเภทซับเซตคลีนโค้ด ในงานวิทยานิพนธ์จะไม่ทำการปรับปรุงตัวอย่างโค้ดที่จำแนกได้ผลลัพธ์เป็นซับเซตคลีนโค้ด

(2) เมื่อแก้ไขปรับปรุงแล้ว ค่าที่ได้จากมาตรวัดไม่มีการเปลี่ยนแปลงในส่วนของค่าดัชนีความสามารถในการบำรุงรักษา เช่น การเพิ่มและลดส่วนต่อประสาน การเปลี่ยนเมท็อดจากพับบลิคเป็นไพรเวท และการลดจำนวนพารามิเตอร์ ซึ่งการปรับปรุงดังกล่าว จะไม่มีผลทำให้ค่าดัชนีความสามารถในการบำรุงรักษาเปลี่ยนแปลง

ทั้งนี้หลังจากทำการทดลองโดยนำชุดตัวอย่างจำนวน 60 ชุด ประกอบด้วย ชุดตัวอย่างซับเซตของคลีนโค้ด ชุดตัวอย่างโค้ดที่มีความคลุมเครือ และชุดตัวอย่างร่องรอยไม่ดี โดยพิจารณาจากกลุ่มโค้ดที่ได้รับการปรับปรุงทั้งหมด 40 ชุดพบว่า 25 ชุดตัวอย่างหรือคิดเป็นร้อยละ 62.5 เป็นไปตามวิธีการที่ได้ออกแบบไว้คือ สามารถปรับปรุงโค้ดเพื่อเพิ่มค่าดัชนีความสามารถในการบำรุงรักษาและสามารถป้องกันไม่ให้เกิดปัญหาเรื่องโค้ดที่มีคุณภาพแย่ลงได้ มี 11 ชุดตัวอย่างเท่านั้นหรือคิดเป็นอัตราร้อยละ 27.5 ของชุดตัวอย่างทั้งหมดที่มีค่าดัชนี ความสามารถในการบำรุงรักษา ลดลง สุดท้ายมีเพียง 4 ชุดตัวอย่างหรือคิดเป็นร้อยละ 10 ที่ค่าดัชนีความสามารถในการบำรุงรักษาไม่เปลี่ยนแปลง ซึ่งกรณีที่ค่าไม่เปลี่ยนแปลงหรือลดลงถือว่าไม่เป็นไปตามวิธีการที่ออกแบบตามวิทยานิพนธ์นี้

ตารางที่ 5.23 เปรียบเทียบค่าคัปปลิงของร่องรอยไม่ดีก่อนและหลังการปรับปรุงโค้ด

ลำดับที่	รายการร่องรอยไม่ดี	ผลรวมค่าคัปปลิงที่วัดได้ก่อนการปรับปรุง	ผลรวมค่าคัปปลิงที่วัดได้หลังการปรับปรุง	ผลการปรับปรุง
1	Converting Hexadecimal	17	17	ไม่เปลี่ยนแปลง
2	DirectoryInfoExtensions	16	10	ลดลง
3	QueueExample1	4	4	ไม่เปลี่ยนแปลง
4	Assignment1MH_Base	9	9	ไม่เปลี่ยนแปลง
5	EventedListExample	54	56	เพิ่มขึ้น
6	ImmutableCollections	70	72	เพิ่มขึ้น
7	FeatureEnvy	5	6	เพิ่มขึ้น
8	inappropriateintimacy	5	5	ไม่เปลี่ยนแปลง
9	LAZY_CLASS1	3	1	ลดลง
10	LAZY_CLASS2	3	2	ลดลง
11	LONG_PARAMETER_LISTS	25	11	ลดลง
12	Mixed_Messages	7	2	ลดลง
13	SimpleLinqToXml	19	19	ไม่เปลี่ยนแปลง
14	Party Planner 2	29	29	ไม่เปลี่ยนแปลง
15	PartialClassAddIn	53	53	ไม่เปลี่ยนแปลง
16	Using Delegates	9	3	ลดลง
17	Hit_the_keys	33	34	เพิ่มขึ้น
18	List_of_Ducks	21	21	ไม่เปลี่ยนแปลง
19	Sloppy_Joe	17	21	เพิ่มขึ้น
20	TallGuy	19	19	ไม่เปลี่ยนแปลง

ตารางที่ 5.24 เปรียบเทียบค่าคัปปลิงของโค้ดที่มีความคลุมเครือก่อนและหลังการปรับปรุงโค้ด

ลำดับที่	รายการร่องรอยไม่ดี	ผลรวมค่าคัปปลิงที่วัดได้ก่อนการปรับปรุง	ผลรวมค่าคัปปลิงที่วัดได้หลังการปรับปรุง	ผลการปรับปรุง
1	PartialClassInterfaces	13	14	เพิ่มขึ้น
2	TaxApp	25	25	ไม่เปลี่ยนแปลง
3	UnhandledThreadException	34	34	ไม่เปลี่ยนแปลง
4	UnhandledWPFException	28	28	ไม่เปลี่ยนแปลง
5	WebBrowser	31	31	ไม่เปลี่ยนแปลง
6	Joe_and_Bob	21	21	ไม่เปลี่ยนแปลง
7	Mileage_calculator	21	8	ลดลง
8	Time_to_start_coding	17	17	ไม่เปลี่ยนแปลง
9	BinaryWriter	7	7	ไม่เปลี่ยนแปลง
10	LINQ to XML	16	16	ไม่เปลี่ยนแปลง
11	OfficeSample	15	15	ไม่เปลี่ยนแปลง
12	LinqToNorthwind	94	94	ไม่เปลี่ยนแปลง
13	ObjectDumper	10	10	ไม่เปลี่ยนแปลง
14	PasteXmlAsLinq	50	50	ไม่เปลี่ยนแปลง
15	PropertyBag	93	93	ไม่เปลี่ยนแปลง
16	Rss	17	17	ไม่เปลี่ยนแปลง
17	SimpleLambdas	15	15	ไม่เปลี่ยนแปลง
18	WinFormsDataBinding	104	104	ไม่เปลี่ยนแปลง
19	XMLdoc	1	1	ไม่เปลี่ยนแปลง
20	XQuery	14	14	ไม่เปลี่ยนแปลง

ตารางที่ 5.25 เปรียบเทียบค่าโคฮีชันของร่องรอยไม่ดีก่อนและหลังการปรับปรุงโค้ด

ลำดับที่	รายการร่องรอยไม่ดี	ค่าเฉลี่ย LCOM1 ที่วัดได้ก่อนการปรับปรุง	ค่าเฉลี่ย LCOM1 ที่วัดได้หลังการปรับปรุง	ค่าเฉลี่ย LCOM3 ที่วัดได้ก่อนการปรับปรุง	ค่าเฉลี่ย LCOM3 ที่วัดได้หลังการปรับปรุง	ผลการปรับปรุง
1	Converting Hexadecimal	0.375	0.45	0.375	0.45	ไม่เปลี่ยนแปลง
2	DirectoryInfoExtensions	0	0	0.33	0.375	เพิ่มขึ้น
3	QueueExample1	0.25	0.5	0.25	0.5	ไม่เปลี่ยนแปลง
4	Assignment1MH_Base	0.471	0.518	0.492	0.551	เพิ่มขึ้น
5	EventedListExample	0.089	0.088	0.76	0.22	ลดลง
6	ImmutableCollections	0.423	0.494	0.395	0.461	ลดลง
7	FeatureEnvy	0.16	0.22	0.2	0.25	เพิ่มขึ้น
8	inappropriateintimacy	0.366	0.456	0.366	0.456	ไม่เปลี่ยนแปลง
9	LAZY_CLASS1	0.066	0.133	0.225	0.3	เพิ่มขึ้น
10	LAZY_CLASS2	0	0	0.13	0.2	เพิ่มขึ้น
11	LONG_PARAMETER_LISTS	0.516	0.57	0.516	0.57	ไม่เปลี่ยนแปลง
12	Mixed_Messages	0.187	0.25	0.33	0.4	เพิ่มขึ้น
13	SimpleLinqToXml	0	0	0	0	ไม่เปลี่ยนแปลง

ตารางที่ 5.25 เปรียบเทียบค่าโคฮีชันของร่องรอยไม่ดีก่อนและหลังการปรับปรุงโค้ด (ต่อ)

ลำดับที่	รายการร่องรอยไม่ดี	ค่าเฉลี่ย LCOM1 ที่วัดได้ก่อนการปรับปรุง	ค่าเฉลี่ย LCOM1 ที่วัดได้หลังการปรับปรุง	ค่าเฉลี่ย LCOM3 ที่วัดได้ก่อนการปรับปรุง	ค่าเฉลี่ย LCOM3 ที่วัดได้หลังการปรับปรุง	ผลการปรับปรุง
14	Party Planner 2	0.492	0.526	0.587	0.64	เพิ่มขึ้น
15	PartialClassAddIn	0.85	0.965	0.8	0.89	ลดลง
16	Using Delegates	0.2	0.4	0.25	0.33	เพิ่มขึ้น
17	Hit_the_keys	0.193	0.24	0.383	0.436	เพิ่มขึ้น
18	List_of_Ducks	0.156	0.182	0.236	0.282	เพิ่มขึ้น
19	Sloppy_Joe	0.166	0.274	0.286	0.403	เพิ่มขึ้น
20	TallGuy	0.5	0.66	0.25	0.33	ลดลง

ตารางที่ 5.26 เปรียบเทียบค่าโคฮีชันของโค้ดที่มีความคลุมเครือก่อนและหลังการปรับปรุงโค้ด

ลำดับที่	รายการร่องรอยไม่ดี	ค่าเฉลี่ย LCOM1 ที่วัดได้ก่อนการปรับปรุง	ค่าเฉลี่ย LCOM1 ที่วัดได้หลังการปรับปรุง	ค่าเฉลี่ย LCOM3 ที่วัดได้ก่อนการปรับปรุง	ค่าเฉลี่ย LCOM3 ที่วัดได้หลังการปรับปรุง	ผลการปรับปรุง
1	PartialClassInterfaces	0	0	0	0	ไม่เปลี่ยนแปลง

ตารางที่ 5.26 เปรียบเทียบค่าโคฮีชันของโค้ดที่มีความคลุมเครือก่อนและหลังการปรับปรุงโค้ด (ต่อ)

ลำดับที่	รายการร่องรอยไม่ดี	ค่าเฉลี่ย LCOM1 ที่วัดได้ก่อนการปรับปรุง	ค่าเฉลี่ย LCOM1 ที่วัดได้หลังการปรับปรุง	ค่าเฉลี่ย LCOM3 ที่วัดได้ก่อนการปรับปรุง	ค่าเฉลี่ย LCOM3 ที่วัดได้หลังการปรับปรุง	ผลการปรับปรุง
2	TaxApp	0.335	0.42	0.335	0.42	ไม่เปลี่ยนแปลง
3	UnhandledThreadException	0.29	0.35	0.375	0.45	เพิ่มขึ้น
4	UnhandledWPFException	0.81	0.965	0.81	0.965	ไม่เปลี่ยนแปลง
5	WebBrowser	0.43	0.455	0.43	0.455	ไม่เปลี่ยนแปลง
6	Joe_and_Bob	0.343	0.433	0.343	0.433	ไม่เปลี่ยนแปลง
7	Mileage_calculator	0.36	0.455	0.36	0.455	ไม่เปลี่ยนแปลง
8	Time_to_start_coding	0.38	0.44	0.425	0.5	เพิ่มขึ้น
9	BinaryWriter	0	0	0	0	ไม่เปลี่ยนแปลง
10	LINQ to XML	0	0	0	0	ไม่เปลี่ยนแปลง
11	OfficeSample	0	0	0	0	ไม่เปลี่ยนแปลง
12	LinqToNorthwind	0.5	0.524	0.5	0.524	ไม่เปลี่ยนแปลง
13	ObjectDumper	0.72	0.8	0.76	0.82	เพิ่มขึ้น
14	PasteXmlAsLinq	0.77	0.84	0.79	0.845	เพิ่มขึ้น

ตารางที่ 5.26 เปรียบเทียบค่าโคฮีชันของโค้ดที่มีความคลุมเครือก่อนและหลังการปรับปรุงโค้ด (ต่อ)

ลำดับที่	รายการร่องรอยไม่ดี	ค่าเฉลี่ย LCOM1 ที่วัดได้ก่อนการปรับปรุง	ค่าเฉลี่ย LCOM1 ที่วัดได้หลังการปรับปรุง	ค่าเฉลี่ย LCOM3 ที่วัดได้ก่อนการปรับปรุง	ค่าเฉลี่ย LCOM3 ที่วัดได้หลังการปรับปรุง	ผลการปรับปรุง
15	PropertyBag	0.528	0.578	0.53	0.58	เพิ่มขึ้น
16	Rss	0	0	0	0	ไม่เปลี่ยนแปลง
17	SimpleLambdas	0	0	0	0	ไม่เปลี่ยนแปลง
18	WinFormsDataBinding	0.51	0.63	0.55	0.578	ไม่เปลี่ยนแปลง
19	XMLdoc	0.6	0.75	0.6	0.75	ไม่เปลี่ยนแปลง
20	XQuery	1	1	1	1	ไม่เปลี่ยนแปลง

จากตารางการเปรียบเทียบค่าคัปปลิงและโคฮีชัน จากกลุ่มตัวอย่างที่ผ่านการปรับปรุงโดยแบ่งออกเป็นโค้ดกลุ่มร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือพบว่า

1) ค่าคัปปลิงและโคฮีชัน ในกลุ่มของร่องรอยไม่ดีและความคลุมเครือหลังจากทำการปรับปรุงแล้ว ค่าคัปปลิง และโคฮีชัน ไม่มีการเปลี่ยนแปลง เนื่องจากวิธีการปฏิบัติที่กำหนดขึ้นมุ่งเน้นแก้ไขโดยยึดหลักผลกระทบตามวิธีการที่ออกแบบไว้ ทำให้ผลลัพธ์หลังจากการแก้ไขโดยส่วนมากไม่เปลี่ยนแปลง ทั้งนี้หากพิจารณาวิธีการแก้ไขความคลุมเครือตามวิธีการที่ออกแบบไว้จะเห็นได้ว่ามุ่งเน้นแก้ไขปัญหาที่เกิดขึ้นในส่วนย่อยๆ ภายในคลาสก่อนเป็นลำดับแรก และขยายไปยังส่วนต่างๆ ที่อยู่ภายนอกที่เกี่ยวข้องตามลำดับ

2) ค่าคัปปลิงในกลุ่มร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือ หลังจากทำการปรับปรุงแล้วมีค่าคัปปลิง เพิ่มขึ้น เพราะขึ้นอยู่กับวิธีการแก้ไขตามชนิดของปัญหาที่เกิดขึ้น ประกอบกับการแบ่งคลาสออกเป็นส่วนย่อยๆ ทำให้การเรียกใช้งานในคลาสเดิมลดลง แต่ไปเพิ่ม ค่าคัปปลิงในส่วนของคลาสใหม่ที่เพิ่มมาแทนในโปรแกรม เช่น วิธีการแก้ไข Extract Class และ Extract SubClass เป็นต้น ทำให้ผลรวมของค่าคัปปลิงเพิ่มขึ้น แต่หากทำการเฉลี่ยค่าตามจำนวนคลาสจะพบว่า แท้จริงแล้วไม่ได้เพิ่มขึ้นแต่อย่างใด

3) ค่าคัปปลิงในกลุ่มของร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือ หลังจากปรับปรุงแล้วจะมีค่าคัปปลิงลดลง เนื่องจากการยุบรวมของคลาสหรือการย้ายการทำงานเมท็อดหรือแอตทริบิวต์ระหว่างคลาสนั้น ด้วยวิธีปฏิบัติ อาทิเช่น Collapse Hierarchy Move Field Move Method และ Pull Up เป็นต้น เป็นสาเหตุให้ค่าคัปปลิงของคลาสลดลง ซึ่งแสดงให้เห็นถึงความสามารถในการบำรุงรักษาของโค้ดดีขึ้น

4) ค่าโคฮีชัน ในกลุ่มร่องรอยไม่ดีและความคลุมเครือหลังจากทำการปรับปรุงแล้วมี ค่าโคฮีชันเพิ่มขึ้น ซึ่งขึ้นอยู่กับชนิดร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือที่ถูกแก้ไข หากการแก้ไขปัญหาที่เกิดขึ้นจากการทำงานภายในคลาสหรือการทำงานร่วมกันระหว่างส่วนต่างๆในคลาส หลังจากการแก้ไขแล้ว ค่าโคฮีชัน เพิ่มขึ้น เป็นเพราะเกิดจากการทำงานภายในที่เป็นระบบและทุกส่วนย่อยของคลาสมีความสัมพันธ์มากขึ้น

5) ค่าโคฮีชันในกลุ่มของร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือหลังจากทำการปรับปรุงแล้วมีค่าโคฮีชัน ลดลง เพราะเกิดจากการแก้ไขร่องรอยไม่ดีบางวิธี ที่ทำให้ต้องรวมเมท็อดหรือคลาส ซึ่งทำให้จำนวนของการทำงานหรือหน้าที่รับผิดชอบของคลาสเพิ่มขึ้น

หากพิจารณาค่าดัชนีบำรุงรักษาในกรณีที่มีค่าไม่เปลี่ยนแปลงของกลุ่มของร่องรอยไม่ดีและความคลุมเครือโดยพิจารณาควบคู่กับค่า คัปปลิง และโคฮีชัน พบว่ามีบางกรณีที่ค่าดัชนีบำรุงรักษามีค่าไม่เปลี่ยนแปลงแต่ค่า คัปปลิง กลับลดน้อยลงและ โคฮีชัน กลับมีค่าเพิ่มขึ้น ซึ่งหมายถึงตัวอย่างการทดลองที่ตรวจสอบนั้นมีระดับค่าความสามารถในการบำรุงรักษาดี แต่ไม่สามารถระบุจากค่าดัชนีความสามารถในการบำรุงรักษาตามที่คัดเลือกมาได้ เพราะวิทยานิพนธ์ได้กำหนดข้อจำกัดของมาตรวัดที่ใช้พิจารณาไว้

5.6 ผลจากขั้นตอนการทวนสอบกลุ่มโค้ดตามเกณฑ์ที่สร้าง

การทวนสอบกลุ่มโค้ดตามเกณฑ์ที่สร้าง เพื่อการทวนสอบกลุ่มโค้ดในที่นี้ ได้มาจากการแบ่งกลุ่มตามกระบวนการในข้อ 5.2.2 โดยมีจำนวนเท่ากับกลุ่มโค้ดชุดเริ่มต้น เพื่อนำมาใช้ทดสอบการจำแนกตามกระบวนที่ถูกสร้างขึ้นว่ามีความถูกต้องแม่นยำมากน้อยเพียงใด ด้วยการจำแนกตัวอย่างโค้ด 60 ชุด ผลลัพธ์ที่ได้จากการจำแนกโค้ดตามเกณฑ์ออกเป็น 3 กลุ่มคือ กลุ่มซับเซตคลีนโค้ด กลุ่มโค้ดที่มีความคลุมเครือ และกลุ่มโค้ดที่มีร่องรอยไม่ดี พร้อมกับวัดค่าดัชนีความสามารถในการบำรุงรักษา ได้ผลลัพธ์ดังตารางต่อไปนี้

ตารางที่ 5.27 ผลการจำแนกโค้ดชุดทวนสอบพร้อมค่าดัชนีความสามารถในการบำรุงรักษา

ประเภทโค้ด	รายการโค้ดตัวอย่างชุดทวนสอบ	ค่าดัชนีบำรุงรักษาที่ไม่ถูกนับรวมคำอธิบายโปรแกรม (Miwoc)	ค่าดัชนีบำรุงรักษาที่นับรวมคำอธิบายโปรแกรม (Miwc)	ค่าดัชนีความสามารถในการบำรุงรักษา (MI)
คลีนโค้ด	1. ConsoleTCPServer	72	25	88.15
	2. Custom_IComparer	78	30	95.86
	3. FileCopier	73	28	88.28
	4. CableBill	75	28	91.70
	5. Familiar_math_symbols	72	48	79.18
	6. Talker_Tester	72	27	87.08
	7. Two_Decks	74	14	99.15

ตารางที่ 5.27 ผลการจำแนกโค้ดชุดทวนสอบพร้อมค่าดัชนีความสามารถในการบำรุงรักษา (ต่อ)

ประเภทโค้ด	รายการโค้ดตัวอย่างชุดทวนสอบ	ค่าดัชนีบำรุงรักษาที่ไม่ถูกนับรวมคำอธิบายโปรแกรม (Miwoc)	ค่าดัชนีบำรุงรักษาที่นับรวมคำอธิบายโปรแกรม (Miwc)	ค่าดัชนีความสามารถในการบำรุงรักษา (MI)
คลีนโค้ด	8. BeeControl	72	28	86.57
	9. Equality	74	40	85.02
	10. ExcuseManager	72	24	88.71
	11. optional_parameters	74	39	85.37
	12. Simple_Text_Editor	69	40	76.47
	13. Whack_a_mole	73	25	89.86
	14. DeclareArraySample	73	31	86.85
	15. Events	81	37	98.06
	16. ExplicitInterface	83	57	95.90
	17. Libraries	80	37	96.35
	18. NamedAndOptional	83	50	97.47
	19. Nullable	73	31	86.85
	20. PartialTypes	75	20	96.31
	21. SimpleVariance	91	6	137.09
	22. UserConversions	87	42	106.59
	23. Versioning	90	21	121.31
	24. Yield	84	32	105.22
	25. Image resizing	75	35	88.58
โค้ดที่มีความคลุมเครือ	1. COMInteropPart1	71	1	113.69
	2.EmployeeTracker.Employee	82	52	95.28
	3. Generics2	77	26	96.15
	4. Pinvoke	96	1	156.44

ตารางที่ 5.27 ผลการจำแนกโค้ดชุดทวนสอบพร้อมค่าดัชนีความสามารถในการบำรุงรักษา (ต่อ)

ประเภทโค้ด	รายการโค้ดตัวอย่างชุดทวนสอบ	ค่าดัชนีบำรุงรักษาที่ไม่ถูกนับรวมคำอธิบายโปรแกรม (Miwoc)	ค่าดัชนีบำรุงรักษาที่นับรวมคำอธิบายโปรแกรม (Miwc)	ค่าดัชนีความสามารถในการบำรุงรักษา (MI)
โค้ดที่มีความคลุมเครือ	5. Security	64	3	96.18
	6. LinqToXmlDataBinding	100	15	142.77
	7. SimpleLinqToObjects	65	1	103.43
ร่องรอยไม่ดี	1. Contacts	89	7	133.98
	2. AnonymousDelegates	82	16	110.81
	3. Attributes	88	36	150.48
	4. ConditionalMethods	84	0	143.64
	5. Delegates	79	33	150.48
	6. EmployeeTracker.Common	76	69	95.28
	7. EmployeeTracker.Fakes	64	44	105.42
	8. Tokens	82	46	102.26
	9. Owner drawn	69	10	87.44
	10. Unsafe	68	26	77.34
	11. Reflector	61	11	93.44
	12. Mixed_Messages2	86	3	133.80
	13. Swapping_elephants	78	34	95.86
	14. CommandLine	84	28	141.93
	15. EmployeeTracker.Model	89	76	105.42
	16. OleDbSample	54	27	92.34
	17. Properties	84	21	141.93
	18. Structs	83	0	157.32
	19. Threading	73	28	126.54

ตารางที่ 5.27 ผลการจำแนกโค้ดชุดทวนสอบพร้อมค่าดัชนีความสามารถในการบำรุงรักษา (ต่อ)

ประเภทโค้ด	รายการโค้ดตัวอย่างชุดทวนสอบ	ค่าดัชนีบำรุงรักษาที่ไม่ถูกนับรวมคำอธิบายโปรแกรม (Miwoc)	ค่าดัชนีบำรุงรักษาที่นับรวมคำอธิบายโปรแกรม (Miwc)	ค่าดัชนีความสามารถในการบำรุงรักษา (MI)
ร่องรอยไม่ดี	20. Business Days	56	15	67.53
	21. Familytree	82	14	112.83
	22. Breakfast for Lumberjacks	70	24	85.29
	23. Beehive Simulator	68	20	104.28
	24. ClassLeaf	75	29	87.80
	25. StartingPoint	78	37	93.36
	26. ImageMapConverter	68	37	77.86
	27. Animations	84	0	143.64
	28. DataContractSerializer	75	0	128.25

ผลจากการจำแนกโค้ดชุดทวนสอบเห็นได้ว่า วิธีการจำแนกโค้ดนั้นสามารถแยกได้ออกเป็น 3 กลุ่มเช่นเดียวกับกลุ่มโค้ดทดสอบชุดเริ่มต้นคือ กลุ่มซับเซตคลีนโค้ด กลุ่มโค้ดที่มีความคลุมเครือ และกลุ่มร่องรอยไม่ดี ซึ่งผู้วิจัยได้ตรวจความถูกต้องอีกครั้ง โดยตรวจเก็บรายละเอียดและข้อผิดพลาดที่อาจเกิดขึ้นจากการจำแนกตามกระบวนการที่ออกแบบ ทั้งนี้เพื่อให้มั่นใจว่าวิธีการที่ออกแบบมีความถูกต้องแม่นยำ

ผลลัพธ์จากการตรวจสอบด้วยผู้วิจัยพบว่า กลุ่มตัวอย่าง 51 ชุดตัวอย่างหรือคิดเป็นอัตราร้อยละ 85 สามารถจำแนกโค้ดตามวิธีการที่ออกแบบไว้ในวิทยานิพนธ์นี้ได้จริง และส่วนที่เหลือ 9 ชุดตัวอย่างหรือคิดเป็นร้อยละ 15 เกิดข้อผิดพลาดจากการทดลองบางประการ ซึ่งผู้วิจัยขอสรุปข้อผิดพลาดได้ดังต่อไปนี้

1) ตัวอย่างทดลองที่ถูกจำแนกเป็นซับเซตคลีนโค้ดและความคลุมเครือ มีประเภทร่องรอยไม่ดีชนิดใหม่ที่ไม่ได้ถูกระบุอยู่ในขอบเขตงานวิทยานิพนธ์นี้ ซึ่งประเภทของร่องรอยไม่ดีที่ตรวจพบอาทิเช่น Middle man Speculative Generality เป็นต้น ซึ่งปรากฏอยู่ในกลุ่มโค้ดตัวอย่างเช่น ExcuseManager ExplicitInterface เป็นต้น

2) กลุ่มโค้ดตัวอย่างถูกจำแนกเป็นซับเซตคลีนโค้ดและความคลุมเครือ เป็นโค้ดที่ถูกสร้างจากเครื่องมือทั้งหมดโดยไม่มีผู้พัฒนาเข้าไปเกี่ยวข้อง ตามข้อกำหนดพื้นฐานของวิทยานิพนธ์ ตัวอย่างทดลองต้องมีส่วนใดส่วนหนึ่งของโปรแกรมถูกแก้ไขหรือพัฒนาเพิ่มเติมด้วยผู้พัฒนาก่อน เพื่อทราบความต้องการหรือแนวความคิดของการออกแบบเขียนโปรแกรมของผู้พัฒนา ซึ่งจำนวนกลุ่มโค้ดตัวอย่างที่พบมีจำนวน 4 ชุด เช่น Animations, EmployeeTracker.Employee เป็นต้น

เมื่อเสร็จสิ้นขั้นตอนการทวนสอบ ถือเป็นขั้นตอนสุดท้ายของกระบวนการ ผู้วิจัยได้รวบรวมประเด็นสำคัญและข้อจำกัดที่พบระหว่างการทดสอบตามกระบวนการ โดยอธิบายรายละเอียดพร้อมข้อเสนอแนะในบทถัดไป

บทที่ 6
สรุปผลการวิจัยและข้อเสนอแนะ

งานวิทยานิพนธ์นี้ได้สร้างวิธีการสนับสนุนให้เกิดกระบวนการพัฒนาซอฟต์แวร์ที่นำไปสู่ผลผลิตที่มีคุณภาพ เริ่มต้นจากนำ มาตรวัดซอฟต์แวร์และฟัซซี่โลจิ กมาทำการจำแนกกลุ่มโค้ดออกเป็น 3 กลุ่มได้แก่ กลุ่ม ซับเซตคลีนโค้ด กลุ่มโค้ดที่มี ความคลุมเครือ และกลุ่มร่องรอยไม่ดี เมื่อนำดัชนีชี้วัดความสามารถในการบำรุงรักษามาประเมินกลุ่ม ซับเซตคลีนโค้ด จึงพบว่าเป็นกลุ่มโค้ดที่มีความสามารถในด้านการบำรุงรักษาสูง ผู้วิจัยจึงออกแบบวิธีปฏิบัติด้วยเทคนิครีแฟคทอริ่งเพื่อที่ปรับปรุงแก้ไขโค้ดในกลุ่มร่องรอยไม่ดี และกลุ่มโค้ดที่มีความคลุมเครือให้มีคุณสมบัติเป็นซับเซตคลีนโค้ดโดยจะเป็นการเพิ่มค่าดัชนีความสามารถในการบำรุงรักษา ซึ่งทำให้ได้ซอฟต์แวร์ที่มีคุณภาพตามจุดมุ่งหมายของวิทยานิพนธ์ ผู้วิจัยขอสรุปผลการวิจัยดังต่อไปนี้

6.1 สรุปผลการวิจัย

สำหรับงานวิทยานิพนธ์นี้ ได้นำมาตรวัดซอฟต์แวร์และกฎการจำแนกด้วยฟัซซี่โลจิกมาใช้สำหรับสร้างวิธีการจำแนกเพื่อให้ได้ผลลัพธ์ที่ถูกต้อง และเน้นให้วิธีการที่ออกแบบมีความแม่นยำ จึงใช้การวิธีการทวนสอบเพื่อตรวจสอบความถูกต้องของวิธีการจำแนกที่สร้างขึ้น นอกจากนี้นำเทคนิค การรีแฟคทอริ่งที่ออกแบบ โดยผู้เชี่ยวชาญมาแก้ไขร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือมาสร้างวิธีการปฏิบัติเพื่อปรับปรุงร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือให้กลับไปเป็นซับเซตคลีนโค้ด จากนั้นจะประเมินค่าดัชนีความสามารถในการบำรุงรักษาของแต่ละกลุ่มโค้ดเพื่อเปรียบเทียบให้มั่นใจกับวิธีการปฏิบัติที่ออกแบบนั้นสามารถ เพิ่มค่าดัชนีความสามารถในการบำรุงรักษาได้จริง โดยมีข้อสรุปตามรายละเอียดดังนี้

6.1.1 ด้านการจำแนกโค้ดด้วยมาตรวัดซอฟต์แวร์และกฎฟัซซี่โลจิก

วิธีการนี้สามารถจำแนกและระบุประเภทของโค้ด ได้ออกเป็น 3 กลุ่ม ได้แก่ กลุ่มซับเซตคลีนโค้ด กลุ่มร่องรอยไม่ดี และกลุ่มโค้ดที่มีความคลุมเครือ โดยการคัดเลือกมาตรวัดซอฟต์แวร์ที่เหมาะสมกับกลุ่มร่องรอยไม่ดีและกลุ่ม ซับเซตคลีนโค้ด อย่างไรก็ตามกลุ่มตัวอย่างที่ผ่านขั้นตอนการวัดจำเป็นต้องผ่านตรวจสอบและกำหนดข้อจำกัดของโค้ดที่ผู้วิจัยกำหนดขึ้นเพื่อทำการปรับค่าที่วัดได้ให้เหมาะสมก่อนนำไปใช้แปลความในกระบวนการฟัซซี่โลจิกเพื่อจำแนกโค้ด

ออกเป็นกลุ่มต่างๆ ตามที่กำหนด จากตัวอย่างทดสองสามารถระบุประเภทซับเซตของคลีนโค้ด ความคลุมเครือ และร่องรอยไม่ดีครบทุกประเภทตามขอบเขตของวิทยานิพนธ์นี้ ทั้งนี้ การประเมินกระบวนการที่นำเสนอ นี้ ทำโดยใช้วิธีการ ทวนสอบ ซึ่งผลของการ ทวนสอบพบว่า กลุ่มตัวอย่างทวนสอบจำนวน 51 ชุดตัวอย่างหรือคิดเป็นอัตราร้อยละ 85 สามารถจำแนกโค้ดได้อย่างถูกต้องตามวิธีการที่ออกแบบไว้ในวิทยานิพนธ์ แต่มีเพียงตัวอย่าง 9 ชุดหรือคิดเป็นร้อยละ 15 ที่เกิดข้อผิดพลาดขึ้น ซึ่งจะขออธิบายในส่วนของปัญหาและข้อจำกัดในวิทยานิพนธ์ ในขั้นตอนการจำแนกด้วยกฎการจำแนกด้วยฟัซซี่ยังมีการนำกฎการอธิบายมาช่วยระบุตำแหน่งของปัญหาที่เกิดขึ้นในคลาส เมท็อดและแอตทริบิวต์ ซึ่งเป็นการช่วยเหลือผู้เชี่ยวชาญในการพิจารณาแก้ไขร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือได้สะดวกและชัดเจนขึ้น

6.1.2 ด้านวิธีการปฏิบัติในการปรับปรุงด้วยเทคนิครีแฟคทอริ่ง

ตามที่ผู้วิจัย ได้รวบรวมและออกแบบวิธีการปฏิบัติเพื่อแก้ไขปัญหา โค้ดกลุ่มร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือให้เป็นซับเซตคลีนโค้ดหรือโค้ดที่มีคุณภาพ จากตัวอย่างเริ่มต้นทดสอบที่ต้องทำการปรับปรุงตามวิธีการที่ออกแบบไว้จำนวน 40 ตัวอย่าง แบ่งออกเป็นร่องรอยไม่ดี 20 ตัวอย่างและความคลุมเครือ 20 ตัวอย่าง สามารถแก้ไขเป็นซับเซตคลีนโค้ดตามวิธีการที่ออกแบบไว้ได้อย่างถูกต้อง และได้มีการเพิ่มเติมวิธีการปฏิบัติเพื่อแก้ไขความคลุมเครือ 2 วิธีได้แก่ Add Comment และ Seal Classes เพื่อช่วยในการแก้ไขปรับปรุงความคลุมเครือที่เกิดขึ้นให้มีความคุณสมบัติซับเซตคลีนโค้ด แต่ในขั้นตอนการปรับปรุงจำเป็นต้องพึ่งพาผู้เชี่ยวชาญในการเลือกวิธีการแก้ไขปรับปรุงโค้ด ตามวิธีการที่ออกแบบไว้ ในวิทยานิพนธ์ นี้มีส่วนที่เป็นประโยชน์ต่อผู้พัฒนาโปรแกรมคือ สามารถช่วยในการตัดสินใจเลือกแก้ไขก่อนหลังตามระดับของผลกระทบของร่องรอยไม่ดีในแต่ละประเภท และวิธีการตัดสินใจที่ช่วยเลือกแก้ไขความคลุมเครือ ซึ่งจะเลือกตามค่าพยายามที่น้อยที่สุด ทั้งนี้ผลสรุปของการปรับปรุงสามารถแก้ไขร่องรอยไม่ดีทุกประเภทตามขอบเขตที่กำหนดขึ้น แต่ในส่วนของความคลุมเครือ ตามแนวทางที่ออกแบบจะเลือกแก้ไขตามค่าความพยายามที่คำนวณได้ ทำให้ความคลุมเครือบางชนิดที่มีค่าพยายามสูงไม่ได้ถูกแสดงวิธีการแก้ไขตามวิธีการ อาทิเช่น Ambiguity in Cohesion และ Ambiguity for Change เป็นต้น

หากวิเคราะห์การแก้ไขร่องรอยไม่ดีประเภท Long Method Long Parameter List และ Bad Comment ถ้าหากปรับปรุงโดยไม่มีการวิเคราะห์อย่างรอบคอบแล้ว อาจก่อให้เกิดความคลุมเครือหลังการแก้ไขร่องรอยไม่ดีขึ้นได้ อาทิเช่น การแก้ไขร่องรอยไม่ดี Long Method อาจทำให้เกิดความคลุมเครือประเภท Ambiguity in Method ขึ้นได้ ดังนั้นจำเป็นต้องวิเคราะห์อย่าง

รอบคอบก่อนจึงทำการแก้ไข จะช่วยลดขั้นตอนการแก้ไขความคลุมเครือที่อาจเกิดขึ้นหลังการปรับปรุงได้

6.1.3 ด้านการประเมินความสามารถในการบำรุงรักษาหลังทำการปรับปรุง

ผลจากการปรับปรุงโค้ดชุดเริ่มต้น ซึ่งมีตัวอย่างทั้งหมด 60 ชุดตัวอย่าง ซึ่งประกอบไปด้วยกลุ่มซับเซตคลีนโค้ด 20 ชุดตัวอย่าง กลุ่มโค้ดที่มีความคลุมเครือ 20 ชุดตัวอย่าง และกลุ่มร่องรอยไม่ดี 20 ชุดตัวอย่าง ผู้วิจัยพบว่า

1) กลุ่มซับเซตคลีนโค้ด เป็นกลุ่มโค้ดที่ไม่ได้มีการปรับปรุงใดๆ และได้ถูกวัดค่าดัชนีความสามารถในการบำรุงรักษา ซึ่งได้ผลลัพธ์ค่าดัชนีอยู่ในเกณฑ์ระดับสูง ถือว่าเป็นไปตามวิทยานิพนธ์นี้ที่ได้ศึกษาไว้

2) กลุ่มโค้ดที่มีความคลุมเครือพบว่า มี 1 3 ชุด มีค่าดัชนีความสามารถในการบำรุงรักษาสูงขึ้น คิดเป็นร้อยละ 65 ของกลุ่มทดสอบ ความคลุมเครือ เป็นไปตามวิธีการที่ออกแบบไว้ มีตัวอย่างโค้ด 3 ชุดที่ค่าดัชนีความสามารถในการบำรุงรักษาลดลง คิดเป็นร้อยละ 15 และมี 4 ชุดที่ค่าดัชนีความสามารถในการบำรุงรักษาไม่เปลี่ยนแปลง คิดเป็นร้อยละ 20 หากนับรวมชุดตัวอย่างที่มีค่าดัชนีความสามารถในการบำรุงรักษาลดลงและไม่เปลี่ยนแปลงคิดเป็นร้อยละ 35 ถือเป็นข้อผิดพลาดของการปรับปรุงคุณภาพโค้ดที่ไม่เป็นไปตามวิธีการที่ออกแบบไว้

3) กลุ่มร่องรอยไม่ดีมีค่าดัชนีความสามารถในการบำรุงรักษาเพิ่มขึ้น 12 ชุดคิดเป็นร้อยละ 60 ของกลุ่มทดสอบร่องรอยไม่ดี เป็นไปตามวิธีการที่ออกแบบไว้ มีโค้ดตัวอย่าง 7 ชุดที่ค่าดัชนีความสามารถในการบำรุงรักษาลดลงคิดเป็นร้อยละ35 และ 1 ชุดตัวอย่างที่ค่าดัชนีความสามารถในการบำรุงรักษาไม่เปลี่ยนแปลง คิดเป็นร้อยละ 5 หากนับรวมชุดตัวอย่างที่มีค่าดัชนีในการบำรุงรักษาลดลงและไม่เปลี่ยนแปลงคิดเป็นร้อยละ 40 ถือเป็นข้อผิดพลาดของการปรับปรุงคุณภาพโค้ดที่ไม่เป็นไปตามวิธีการที่ออกแบบไว้

ดังนั้นความถูกต้องของวิธีการปรับปรุงคิดจากผลรวมของ 12 ชุดตัวอย่างของ ร่องรอยไม่ดี และ 13 ชุดตัวอย่างของความคลุมเครือ จำนวนทั้งสิ้น 25 ชุดตัวอย่างคิดเป็นร้อยละ 62.5 ของชุดตัวอย่างทั้งหมด มี 10 ชุดตัวอย่างที่เกิดปัญหาเรื่องค่าดัชนีบำรุงรักษามีค่าลดลงคิดเป็นร้อยละ 25 และมี 5 ชุดตัวอย่างที่ ค่าดัชนีบำรุงรักษา ไม่มีเปลี่ยนแปลงคิดเป็นร้อยละ 12.5 ทั้งนี้ไม่นับรวม 20 ชุดตัวอย่าง ซับเซตคลีนโค้ดที่ไม่ได้ถูกปรับปรุงและแต่ได้ทดสอบแล้วมีค่าดัชนีบำรุงรักษา

ในระดับสูงซึ่งเป็นไปตามวิธีการที่ออกแบบซึ่งผู้วิจัยวิเคราะห์ได้ว่า เกิดจากปัญหาที่พบระหว่างการทดสอบซึ่งจะอธิบายรายละเอียดในหัวข้อปัญหาและข้อจำกัดในงานวิทยาพนธ์นี้

ส่วนการเปรียบเทียบค่าคัปปลิงและโคฮีชันโดยใช้กลุ่มตัวอย่างที่ผ่านการ ปรับปรุงซึ่งแบ่งออกเป็นโค้ดกลุ่มร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือ พบว่า

1) ค่าคัปปลิงและโคฮีชัน ในกลุ่มของร่องรอยไม่ดีและความคลุมเครือหลังจากทำการปรับปรุงทั้งหมด 40 ชุด พบว่าค่าคัปปลิงและโคฮีชันไม่มีการเปลี่ยนแปลง มีจำนวนทั้งสิ้น 27 ชุด คิดเป็นร้อยละ 67.5 การที่ค่าไม่เปลี่ยนแปลงเนื่องจากวิธีการปฏิบัติที่ออกแบบไว้มุ่งเน้นปรับปรุงโค้ดโดยยึดหลักผลกระทบที่ได้ออกแบบไว้ หากพิจารณาวิธีการแก้ไขความคลุมเครือ จะเห็นได้ว่าส่วนใหญ่จะเป็นการแก้ไขปัญหาที่เกิดขึ้นในส่วนย่อยๆ ภายในคลาสก่อนเป็นลำดับแรก และส่วนที่อยู่ภายนอกที่เกี่ยวข้องตามลำดับ

ส่วนกลุ่มร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือมี ค่าคัปปลิง เพิ่มขึ้น มีจำนวนทั้งสิ้น 6 ชุด คิดเป็นร้อยละ 15 การแก้ไขโดยใช้วิธีการตามชนิดของปัญหาที่เกิดขึ้น เช่น การแบ่งคลาสออกเป็นส่วนย่อยทำให้เพิ่ม ค่าคัปปลิง ในส่วนของคลาสใหม่ที่มาแทน แท้จริงแล้วหากพิจารณาจากเฉลี่ยตามจำนวนคลาสจะพบว่าไม่ได้เพิ่มขึ้นแต่อย่างใด

ส่วนรายการร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือมี ค่าคัปปลิง ลดลง ซึ่งมีจำนวนทั้งหมด 7 ชุดคิดเป็นร้อยละ 17.5 เนื่องจากการยุบรวมของคลาสหรือการย้ายการทำงานเมท็อดหรือแอตทริบิวต์ระหว่างคลาสนั้น ด้วยวิธีการแก้ไขเป็นสาเหตุให้ ค่าคัปปลิง ของคลาสลดลง ซึ่งแสดงให้เห็นถึงความสามารถในการบำรุงรักษาของโค้ดดีขึ้น

2) ค่าโคฮีชัน ในกลุ่มร่องรอยไม่ดีและความคลุมเครือ หลังจากทำการปรับปรุงทั้งหมด 40 ชุด ค่าโคฮีชันมีการเปลี่ยนแปลง จะขึ้นกับชนิดร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือที่ถูกแก้ไข หากการแก้ไขปัญหาที่การทำงานภายในคลาสหรือการทำงานร่วมกันระหว่างส่วนต่างๆในคลาส จะทำให้ค่าโคฮีชันเพิ่มขึ้นเพราะเป็นเกี่ยวข้องกับระบบภายในและทุกส่วนย่อยของคลาส ซึ่งมีจำนวน 16 ชุด คิดเป็นร้อยละ 40

ส่วนรายการร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือที่มี ค่าโคฮีชันลดลงหลังจากทำการปรับปรุง มีจำนวน 4 ชุด คิดเป็นร้อยละ 10 เพราะเกิดจากการแก้ไขร่องรอยไม่ดีบางวิธีที่ทำให้ต้องรวมเมท็อดหรือคลาส ทำให้จำนวนของการทำงานหรือหน้าที่รับผิดชอบของคลาสเพิ่มขึ้น ส่วนรายการที่ค่าโคฮีชันไม่มีการเปลี่ยนมีทั้งหมด 20 ชุด คิดเป็นร้อยละ 50

6.2 ปัญหาและข้อจำกัดในงานวิทยานิพนธ์

ข้อจำกัดของวิทยานิพนธ์ที่เกิดขึ้นระหว่างการทดลอง มีดังต่อไปนี้

1) เนื่องจากโค้ดที่ใช้ในในการทดสอบนั้นมีหลากหลายประเภท ดังนั้นอาจจะมีโค้ดบางส่วนที่มีพฤติกรรมเหมือนร่องรอยไม่ดี เช่นโค้ดที่ถูกสร้างขึ้นอัตโนมัติโดยเครื่องมือของภาษา หรือโค้ดในส่วนหลักของโปรแกรมประยุกต์(Main Application) ที่จำเป็นต้องมีการเรียกใช้ทุกโมดูลเพื่อกำหนดค่าเริ่มต้นในการใช้ งาน หรือการบังคับสืบทอดพฤติกรรมที่บังคับให้คลาสที่รับการสืบทอดจำเป็นต้องนำไปปรับใช้ กับเมท็อดหรือแอตทริบิวต์ที่กำหนด จึงทำให้คลาสมีขนาดใหญ่ และโค้ดบางประเภทเป็นโค้ดที่ถูกสร้างเพื่อใช้ทำการทดสอบการทำงานของโค้ดหรือการทำงานภายในโมดูลต่างๆ ทั้งนี้ จำเป็นต้องกำหนดข้อจำกัดให้แก่โค้ดที่มีพฤติกรรมเหมือนร่องรอยไม่ดีโดยการปรับแก้ผลลัพธ์ที่ผ่านขั้นตอนการวัดด้วยมาตรวัดซอฟต์แวร์ให้มีค่าเหมาะสมก่อนที่จะเข้าสู่ขั้นตอนการแปลความด้วยกฎฟัซซี่ต่อไป

2) เมื่อวิเคราะห์ผลลัพธ์ของการทวนสอบพบว่า มีบางผลลัพธ์ที่จำแนกเป็นซับเซตคลีนโค้ด แต่ไม่ใช่โค้ดที่ถูกออกแบบและพัฒนาโดยผู้พัฒนาโปรแกรม เป็นโค้ดที่ถูกสร้างโดยโปรแกรมอัตโนมัติ ตามเกณฑ์การคัดเลือกที่สร้างไว้จึงถือว่าไม่ผ่านข้อกำหนดพื้นฐานของซับเซตคลีนโค้ด ถือว่าเป็นช่องว่างที่เป็นความเสี่ยงหากนำวิธีการจำแนกไปใช้งานโดยไม่มีการตรวจทานอีกครั้งด้วยผู้วิจัยอาจทำให้เกิดข้อผิดพลาดขึ้นได้

3) เนื่องจากการประเมินคุณภาพของความสามารถในการบำรุงรักษา พบว่ามีกลุ่มโค้ดที่มีค่าดัชนีบำรุงรักษาลดลงหลังจากที่ผ่านกระบวนการปรับปรุง เมื่อพิจารณากลุ่มตัวอย่าง เห็นได้ว่าไม่ได้เกิดจากการเลือกวิธีการแก้ไขผิดวิธี แต่เป็นการสะท้อนค่าดัชนีความสามารถในการบำรุงรักษาที่แท้จริงของกลุ่มการทดลอง อาทิ เช่น การแก้ไขร่องรอยไม่ดีประเภท Lazy Class เป็นตัวอย่างที่มีการมีจำนวนของเมท็อดหรือแอตทริบิวต์น้อยซึ่งทำให้หน้าที่รับผิดชอบที่เกิดขึ้นในการทำงานของคลาสน้อยอาจเป็นสาเหตุที่เกิดร่องรอยไม่ดีประเภทนี้ เมื่อทำการแก้ไขโดยการตัดลดหรือยุบรวมคลาส ทำให้ค่าเฉลี่ยของจำนวนบรรทัดเพิ่มขึ้นหรือเมื่อมีการยุบรวมกันของคลาสและทำให้จำนวนคลาสที่เฉลี่ยลดลง อาจเป็นสาเหตุทำให้ค่าดัชนีความสามารถในการบำรุงรักษานั้นลดลง ทั้งนี้ถ้าพิจารณาการเรียกใช้งานจริง การตัดคลาสที่ไม่ได้ใช้งานออกทำให้ค่าดัชนีบำรุงรักษามีค่าลดลง ซึ่งไม่ใช่วิธีการแก้ไขร่องรอยไม่ดีที่ผิด แต่เป็นการสะท้อนค่าดัชนีบำรุงรักษาจริงที่ควรจะเป็นของโค้ดนั้นออกมา

4) การคำนวณหาค่าดัชนีความสามารถในการบำรุงรักษานั้น เมื่อมีการแก้ไขโค้ดประเภทร่องรอยไม่ดีและความคลุมเครือบางประเภท ซึ่งวิธีปฏิบัติที่เลือกใช้ในการแก้ไขไม่ได้เพิ่มหรือ

ลดจำนวนบรรทัด จำนวนชุดคำสั่ง และคำอธิบายของโปรแกรม ทำให้ค่าดัชนีบำรุงรักษานั้นไม่มีผลต่อการเปลี่ยนแปลงแต่อย่างใด เนื่องจากค่าดัชนีความสามารถในการบำรุงรักษาที่เลือกใช้สำหรับงานวิทยานิพนธ์นี้ ครอบคลุมเฉพาะจำนวนบรรทัด จำนวนของชุดคำสั่งและคำอธิบายของโปรแกรมที่มีการเปลี่ยนแปลง อาจไม่ครอบคลุมถึงส่วนที่ได้รับการแก้ไขทุกส่วน แต่อีกนัยหนึ่ง การแก้ไขและปรับปรุงอาจมีผลกระทบต่อคุณภาพโค้ดส่วนอื่นๆ ที่ไม่ได้ระบุในขอบเขตของวิทยานิพนธ์ก็เป็นได้

5) เกณฑ์มาตรวัดที่ใช้พิจารณาร่องรอยไม่ดีประเภทคำอธิบายการทำงานที่เลือกใช้ในงานวิทยานิพนธ์ นี้จะกำหนดเพียงการตรวจพบหรือไม่พบคำอธิบายของโปรแกรม กรณีอื่นที่นอกเหนือจากนี้ เช่น คำอธิบายที่ไม่มีคุณภาพหรือคำอธิบายที่ไม่เกี่ยวกับการทำงานของโปรแกรม ซึ่งอาจนำมาสู่ข้อถกเถียงไม่สิ้นสุด สำหรับงานวิทยานิพนธ์นี้จะไม่ครอบคลุมถึงส่วนนี้ ซึ่งถือเป็นข้อจำกัดของการวัดร่องรอยไม่ดีประเภทคำอธิบายโปรแกรม

6.3 **ข้อเสนอแนะ**

ผู้วิจัยมีข้อเสนอแนะ ผู้ที่ต้องการนำ กระบวนการนี้ไปทำการปรับปรุงและ ประยุกต์ใช้ดังต่อไปนี้

1) ข้อกำหนดที่ผู้จัยสร้างขึ้นนั้น ถูกสร้างขึ้นโดยการพิจารณาจากกลุ่มตัวอย่างทดสอบ ทั้งนี้อาจมีข้อกำหนดอื่นๆ เพิ่มเติมภายหลังจากที่ทดสอบกับกลุ่มตัวอย่างอื่นๆ

2) ในการกระบวนการ จำแนกโค้ด ไปจนถึงขั้นตอนปฏิบัติในการปรับปรุงในวิทยานิพนธ์นี้ มีขั้นตอนที่ซ้ำซ้อน จำเป็นต้องมีการออกแบบเครื่องมือที่ช่วยในผู้ที่ต้องการนำไปใช้งานสามารถประยุกต์และลดเวลาในขั้นตอนการจำแนกและระบุ ตลอดจนขั้นตอนการเลือกวิธีปฏิบัติในการแก้ไขปัญหา ที่ออกแบบให้สามารถเลือกวิธีที่สามารถช่วยให้ผู้ที่ต้องการแก้ไขโค้ดได้ด้วยตนเองโดยพึ่งพาผู้เชี่ยวชาญให้น้อยที่สุด

3) ในงาน วิทยานิพนธ์ นี้ได้เลือกมาตรวัดดัชนีคุณภาพด้านการบำรุงรักษาเพียงตัวเดียวในการทดสอบ ซึ่งวิธีการที่ผู้วิจัยสร้างขึ้นสามารถขยายผลไปสู่กลุ่มการวัดคุณภาพด้านอื่นในอนาคต

รายการอ้างอิง

[1] Crosby, P. Quality Is Free: The Art of Making Quality Certain. Mentor, 1980.

[2] Drucker, P. Innovation and Entrepreneurship. First edition. Collins, 1993.

[3] McCabe Software Inc. Using Code Quality Metrics in Management of Outsourced Development and Maintenance [Computer file] Available from http://www.mccabe.com/pdf/McCabeCodeQualityMetrics-OutsourcedDev.pdf

[4] International Organization for Standardization. ISO 9126 Software Product Quality. [Computer file]. Available from : http://www.iso.org/iso/catalogue_detail.htm?csnumber=22749 [2011, March 1]

[5] Boehm, B., and Basili, V.R. Software Defect Reduction Top 10 List. Computer 34 (January 2001) :135-137.

[6] Fowler, M. Refactoring: Improving the Design of Existing Code. First edition. Addison-Wesley Professional, 1999.

[7] Stroggylos, K. and Spinellis, D. Refactoring-Does It Improve Software Quality?. The 5th international Workshop on Software Quality 10 (May 2007) : 10-15.

[8] Mäntylä, M., Vanhanen, J., and Lassenius, C. A Taxonomy and an Initial Empirical Study of Bad Smells in Code. International Conference on Software Maintenance, pp. 381-384. Netherlands, 2003.

[9] Martin, R.C. Clean Code: A Handbook of Agile Software Craftsmanship. First edition. Prentice Hall, 2008.

[10] IEEE Computer Society. IEEE Std. 610.12-1990 - Glossary of Software Engineering Terminology. Software Engineering Standards Collection, 2002.

[11] Software Engineering Institute. Maintainability Index Technique for Measuring Program Maintainability SEI Software Technology Review. [Computer file] Available from : http://www.sei.cmu.edu [2002]

[12] Kay, A. C. The early history of Smalltalk. The second ACM SIGPLAN conference on History of programming languages (HOPL-II). 28 (March 1993) : 69-95.

[13] Feldman, S. A Conversation with Alan Kay. Queue 2,9 (December 2004) : 20-30.

[14] Wikiquote. Alan Curtis Kay.[Online]. Available from : http://en.wikiquote.org/wiki /Alan_Kay [2011, February 19]

[15] Wikipedia. Code smell.[Online]. Available from : http://en.wikipedia.org/wiki /Code_smell [2011, May 8]

[16] DeMarco, T. Controlling Software Projects: Management, Measurement, and Estimates. First Edition. Prentice Hall, 1986.

[17] Fenton, N.E. Software Metrics, A Rigorous Approach. First edition.London: Chapman London:Chapman & Hall, 1991.

[18] Halstead, M.H. Elements of Software Science. Operating and programming systems series vol.7. New York:Elsevier Science Ltd, 1977.

[19] Chidamber, S.R. and Kemerer, C.F. A Metrics Suite for Object-Oriented Design. IEEE Transaction. Software Engineering 20 (June 1994) : 476-493.

[20] Chidamber, S.R. and Kemerer, C.F. Towards a Metrics Suite for Object-Oriented Design. OOPSLA '91 Conference proceedings on Object-oriented programming systems, languages, and applications, pp.197-211. New York, 1991.

[21] McCabe, T.J. A complexity measure. IEEE Transaction on Software Engineering 2 (December 1976) : 308-320.

[22] Jones, M.P. Fundamentals of Object-Oriented Design in UML. First edition. Addison-Wesley Professional, 1999.

[23] Meesad, P. Fuzzy systems and Neural Networks Lecture: Fuzzy Logic. Bangkok: Faculty of Information Technology, King Mongkut's University of Technology North, May 2009. (Mimeographed)

[24] Zadeh, L.A., Klir, G.J., and Yuan, B. Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems. First Edition edition. Advances in Fuzzy Systems - Applications and Theory vol.6. World Scientific Pub Co Inc, 1996.

[25] Wikipedia. Fuzzy Logic.[Online]. Available from : http://en.wikipedia.org/wiki /Fuzzy_logic [2011, May 5]

[26] Zadeh, L.A. Fuzzy sets. Information and Control 8 (June 1965) : 338-353.

[27] Coleman, D., Ash D., Lowther B., and Oman P. Using Metrics to Evaluate Software System Maintainability. Computer 27 (August 1994) : 44-49.

[28] Oman, P. and Hagemeister, J. Metrics for Assessing Software System Maintainability. International Conference on Software Maintenance, pp. 337-344. Los Alamitos CA:IEEE Computer Society, 1992.

[29] Pearse, T. and Oman, P. Maintainability measurements on industrial source code maintenance activities. International Conference on Software Maintenance, pp. 295-303. Washington,DC:IEEE Computer Society, 1995.

[30] Mens, T. and Tourwé, T. A Survey of Software Refactoring. IEEE Transaction on Software Engineering 30 (February 2004) : 126-139.

[31] Ruhroth, T., Voigt, H., and Wehrheim, H. Measure, Diagnose, Refactor: A Formal Quality Cycle for Software Models. 35th Euromicro Conference on Software Engineering and Advanced Applications, pp.360-367. Washington,DC :IEEE Computer Society, 2009.

[32] Wikipedia. Software Quality. [Online]. Available from: http://en.wikipedia.org/wiki /Software_quality. [2011, May 9]

[33] Khaled, E.E. Object-Oriented Metrics: A Review of Theory and Practice. Advances in software engineering, pp.23-50. New York : Springer-Verlag New York Inc., 2001.

[34] Zadeh, L.A. Is There a Need for Fuzzy Logic?. Information Sciences: an International Journal 178 (July 2008) : 2751-2779.

[35] Zadeh, L.A. Toward a perception-based theory of probabilistic reasoning with imprecise probabilities. Journal of Statistical Planning and Inference 105 (2002) : 233-264.

[36] Zadeh, L.A. Toward a theory of fuzzy information granulation and its centrality in human reasoning and fuzzy logic. Fuzzy Sets Systems 90(September 1977) : 111-127.

[37] Chandra, E. and Linda, P.E. Class Break Point Determination Using CK MetricsThresholds. Global Journal of Computer Science and Technology 10 (November 2010) : 73-77.

[38] Benlarbi, S., El-Emam, K., Goel, N., and Ra, S.N. Thresholds for Object-Oriented Measures. The 11th International Symposium on Software Reliability Engineering, pp.24-38. San Jose:IEEE Computer Society, 2000.

ภาคผนวก

ภาคผนวก ก
ร่องรอยไม่ดี (Code Smell หรือ Bad Smell)

ก.1 รายการปัญหาที่เกิดขึ้นในโปรแกรม (Smell Taxonomy) [6, 8, 15]

ลักษณะของร่องรอยไม่ดี มีดังนี้

1) Long Method

ฟังก์ชันหรือโมดูลที่มีขนาดใหญ่และซ้ำซ้อน หรือมีการทำงานมากภายในฟังก์ชัน ทำให้เมท็อดมีการรับภาระมากเกินไป ไม่เป็นอิสระ และไม่ยืดหยุ่น ส่งผลให้การปรับปรุงพัฒนาทำได้ยากให้ต่อไปในอนาคต

2) Large Class

ไม่ใช่คลาสที่มีขนาดใหญ่ แต่เป็นคลาสที่มีทำงานรับภาระมากเกินไป โดยสังเกตจากจำนวนของตัวแปร (Variable) และการทำเป็นคู่ (Duplication) ของโค้ด ไม่สามารถที่จะซ่อนไว้ได้ ซึ่งในการทำงานของคลาสในแต่ละครั้งจะพบว่ามีหลายๆส่วนของคลาสไม่ได้ถูกเรียกใช้งาน

3) Primitive Obsession

มีข้อมูล(Data) ที่ใช้ชื่อเหมือนกันแต่ละชนิดมีความแตกต่างกันเนื่องจากวัตถุประสงค์ การทำงานแตกต่างกัน ทำให้มีความสับสนในการนำไปใช้งาน ซึ่งตัวแปรหนึ่งๆ ควรมีสถานะเดียวเท่านั้น

4) Long Parameter List

จำนวนของพารามิเตอร์ที่ถูกใช้งานในแต่ละฟังก์ชันหรือเมท็อด มีจำนวนมากทำให้เกิด Dependency ระหว่างกันมากเกินไป เพราะในการทำงานอย่างหนึ่ง อาจมีความเป็นไปได้แค่ใช้เพียงแค่ตัวแปรเดียว แต่จำเป็น ต้องใส่ค่าที่เหลือด้วยในบางครั้ง ถ้าเมท็อดมีพารามิเตอร์จำนวนมากจริงและใช้งานทุกตัว ก็จะพบว่า วิธีการมีทำงานรับภาระหนักเกินไป

5) Data Clumps

มีข้อมูลแบบเดียวกัน ถูกดึงข้อมูลเข้าไปไว้ใช้งานในหลายๆ คลาส หรือ หลายๆ วิธีการทำให้ไม่สามารถตัดสินได้ว่า แท้จริงแล้วคลาสหรือเมท็อดใดเป็นเจ้าของตัวแปรหรือข้อมูลนั้น

6) Switch Statements

เป็นโค้ดที่มีความซ้ำซ้อนกัน ภายในเงื่อนไขของ Switch statement ภายในคลาส ฟังก์ชันหรือเมท็อดเดียวกัน โดยส่วนใหญ่จะแสดงให้เห็นถึงความซ้ำซ้อน

7) Temporary Field

มีการจัดเก็บข้อมูลชั่วคราวเป็นจำนวนมากในโปรแกรม ส่งผลให้เกิดความไม่เป็นระเบียบ และไม่สามารถบ่งชี้ได้ว่า ส่วนไหนสำคัญหรือถูกใช้งานในปัจจุบัน

8) Refused Bequest

คลาสหนึ่งที่ Inheritance จาก Parent Class แต่ไม่ดึงความสามารถของคลาสแม่มาใช้ แต่กลับเขียนและนิยามตัวเองใหม่

9) Alternative Classes with Different Interfaces

เกิดในกรณีที่คลาสไม่ทำหน้าที่เดียวกัน จำเป็นต้องพึ่งพา ส่วนต่อประสาน แต่ก็ไม่ได้มีวัตถุประสงค์เดียวกัน ทำให้เกิดความสับสนในการทำงาน

10) Parallel Inheritance Hierarchies

การเรียกใช้คลาสย่อย (Subclass) มากกว่า และมีการสืบทอดคลาสเข้ามาเป็นจำนวนมาก ทำให้ไม่สามารถบอกได้ว่า แท้จริงแล้วมีความสามารถจากคลาสแม่ (Parent class) ใด ที่ทำหน้าที่สืบทอด

11) Lazy Class

คลาสที่สร้างขึ้น แต่ไม่มีการทำงานใดๆ เกิดขึ้นเลย ไม่มีประโยชน์ต่อโปรแกรม ทำให้เสียเวลาในการบำรุงรักษา

12) Data Class

คลาสที่จัดเก็บแต่ข้อมูลอย่างเดียว ไม่ทำงานใดๆอีก ตามหลักแล้วควรจะเป็นเพียงแค่วัตถุที่จัดเก็บข้อมูลในโปรแกรมก็พอ ไม่จำเป็นต้องกำหนดให้เป็นคลาส

13) Duplicate Code

โค้ดที่ซ้ำซ้อนกันซึ่งปรากฏมากกว่าหนึ่งที่ในคลาสเดียวกัน ซึ่งอาจมีได้หลายรูปแบบ เช่น โค้ดที่มี นิพจน์เหมือนกันทุกประการ หรือทำงานแบบเดียวกันแต่ต่างอัลกอริทึมก็ถือว่าเป็นโค้ดที่ซ้ำซ้อนกัน

14) Speculative Generality & Dead code

โค้ดที่สร้างขึ้น แต่ไม่ถูกใช้งาน ถึงแม้ว่าจะเป็นการเขียนเพื่อใช้งานในอนาคตก็ตาม ก็ถือว่าเป็นร่องรอยไม่ดีและเรียกว่า Dead code

15) Message Chains

การส่งข้อมูลเป็นทอดหรือเป็นลูกโซ่ ไม่จำเป็นต้องมีคนที่ทำหน้าที่ส่งต่อ เพราะเป็นการสร้างความซับซ้อนให้กับโปรแกรมโดยไม่จำเป็น โปรแกรมควรมีเพียงแค่ผู้ส่งและผู้รับก็เพียงพอแล้ว

16) Middle Man

สถานะที่ทำหน้าที่เป็นคนกลางในโปรแกรม เมื่อต้องการส่วนสำคัญ(Feature) ใด ๆ ไม่สามารถทำได้โดยตรง ต้องขอผ่านคนกลางก่อน ทำให้การทำงานไม่สะดวก เสียเวลา และส่งผลเสียให้กับโปรแกรมมากกว่าผลดี

17) Feature Envy

เป็นการวางเมท็อดผิดตำแหน่ง ส่งผลให้เมื่อมีการเปลี่ยนแปลงพฤติกรรม (Behavior) ต้องมีการดึงข้อมูลจากหลายๆ วัตถุเข้ามา และมีการเรียกใช้มากครั้งเกินไป

18) Inappropriate Intimacy

ภายใต้แต่ละคลาส มีการสร้างเมท็อดที่ซับซ้อน เวลาเรียกใช้งานจึงใช้เวลานาน

19) Divergent Change

การเปลี่ยนแปลงใดๆ จะมีผลกระทบต่อกระบวนการการทำงานอยู่ตลอดเวลา

20) Shotgun Surgery

เมื่อมีการแก้ไขเปลี่ยนแปลงสิ่งใดเล็กน้อย ก็จำเป็นต้องแก้ไขในทุกๆคลาสที่เกี่ยวข้อง ถ้าจำนวนคลาสที่เกี่ยวข้องมีจำนวนมาก ก็เป็นเรื่องยากที่จะต้องตามเข้าไปแก้ไข และบางครั้งอาจจะเกิดข้อบกพร่องได้ง่ายเนื่องจากอาจจะมีข้อผิดพลาดในบางจุดที่แก้ไขไม่ครบ

21) Incomplete Library Class

การเรียกใช้ Library ที่ยังสร้างไม่เสร็จ ทำให้เกิดปัญหา เช่น คาดเดาปัญหาที่จะเกิดขึ้นได้ยาก เมื่อเกิดสิ่งผิดปกติใดๆ ต้องใช้เวลาในการตรวจสอบว่ามาตรงจุดใด

22) Comments

การเขียนคำอธิบายในโปรแกรม เมื่อมีการเปลี่ยนแปลงใดๆ หรือข้อพึงระวังต่างๆ ในโปรแกรม

ภาคผนวก ข
ตัวอย่างรายละเอียดซับเซตคลีนโค้ดพร้อมมาตรวัดซอฟต์แวร์และค่าพิจารณา

ตารางนี้แสดงถึงนิยามและรายละเอียดของซับเซตคลีนโค้ด โดยมีมาตรวัดซอฟ ต์แวร์พร้อมค่าพิจารณาที่ใช้ในการทดสอบกับแนวทางในการปรับปรุงคุณภาพโค้ดด้วยวิธีการจำแนกและระบุหลังจากที่แสดงตัวอย่างซับเซตคลีนโค้ดและมาตรวัดที่ได้ จะนำไปใช้สร้างอินพุตเซตสำหรับฟัซซี่โลจิก

ตารางที่ ข.1 ตัวอย่างนิยามซับเซตคลีนโค้ดและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา Comment

กลุ่มของซับเซตคลีนโค้ด	Comment
ลำดับที่	1.1
นิยาม	Good Comment
แรงจูงใจ	การเขียนคำอธิบายเพื่อบรรยายการทำงานในแต่ละส่วนของโปรแกรม เพื่อให้เข้าใจลักษณะการทำงานได้ง่ายขึ้น
มาตรวัดที่ใช้ในการอธิบาย	PCC คือ การวัดคำอธิบายเพื่อบรรยายการทำงานภายในคลาสที่พิจารณา
ค่าที่ใช้ในการพิจารณา	PCC ค่าที่ใช้พิจารณาควรสูงกว่า 20

ตารางที่ ข.2 ตัวอย่าง นิยามซับเซตคลีนโค้ดและมาตรวัด ซอฟต์แวร์ พร้อมค่าพิจารณา Small Functions

กลุ่มของซับเซตคลีนโค้ด	Function
ลำดับที่	1.2
นิยาม	Small Functions
แรงจูงใจ	ฟังก์ชันควรมีขนาดเล็กให้มากที่สุดเพื่อความสะดวกในการใช้งาน
มาตรวัดที่ใช้ในการอธิบาย	1. NLOC คือ จำนวนของบรรทัดของโค้ดที่ใช้ในการทำงานภายในฟังก์ชันหนึ่งๆ 2. NILI คือ จำนวนชุดคำสั่งในแต่ละฟังก์ชัน ถ้ามีจำนวนมากเกินไป จะทำให้การตรวจสอบความถูกต้องของโปรแกรมและการ

ตารางที่ ข.2 ตัวอย่าง นิยามซับเซตคลีนโค้ดและมาตรวัด ซอฟต์แวร์ พร้อมค่าพิจารณา Small Functions (ต่อ)

มาตรวัดที่ใช้ในการอธิบาย	บำรุงรักษาเป็นไปได้ยาก 3. CC คือ การนับจำนวนของเส้นทางทั้งหมดที่สามารถเกิดได้ในการทำงานภายในฟังก์ชันหรือเมท็อดที่พิจารณา 4. ILCC คือ การนับจำนวนเส้นทาง ทั้งหมดภายในฟังก์ชันหรือเมท็อดที่ถูกพิจารณา เพื่อดูความเป็นไปได้ของชุดคำสั่ง ที่มีการใช้งาน
ค่าที่ใช้ในการพิจารณา	1. NLOC ค่าที่ใช้พิจารณาควรต่ำกว่า 22 2. NILI ค่าที่ใช้พิจารณาควรต่ำกว่า 50 3. CC ค่าที่ใช้พิจารณาควรต่ำกว่า 10 4. ILCC ค่าที่ใช้พิจารณาควรต่ำกว่า 20

ตารางที่ ข.3 ตัวอย่าง นิยามซับเซตคลีนโค้ดและมาตรวัด ซอฟต์แวร์ พร้อมค่าพิจารณา Do One Thing

กลุ่มของซับเซตคลีนโค้ด	Function
ลำดับที่	1.3
นิยาม	Do One Thing
แรงจูงใจ	ฟังก์ชันควรมีการทำงานเพียงแค่หนึ่งอย่างหรือตอบสนอง เพียงเรื่องเดียวเท่านั้น
มาตรวัดที่ใช้ในการอธิบาย	RC คือค่าเฉลี่ยของจำนวนความสัมพันธ์ภายในของการทำงานของทุกส่วน สามารถคำนวณได้โดย H = (R + 1)/N R คือจำนวนของชนิดข้อมูลทั้งหมดที่มีความสัมพันธ์กับการทำงานทั้งหมดที่เกี่ยวข้อง และ N คือจำนวนชนิดที่ถูกใช้ภายในการทำงานที่ตรวจสอบ โดยบวกค่า 1 เพื่อป้องกันไม่ให้ H = 0
ค่าที่ใช้ในการพิจารณา	1. RC ควรอยู่ในช่วงระหว่าง 1.5 ถึง 4.0

ตารางที่ ข.4 ตัวอย่างนิยามซับเซตคลีนโค้ดและมาตรวัด ซอฟต์แวร์พร้อมค่าพิจารณา Nesting Depth

กลุ่มของซับเซตคลีนโค้ด	Function
ลำดับที่	1.4
นิยาม	Nesting Depth
แรงจูงใจ	ระดับความลึกของเมท็อดหรือขอบเขตการทำงาน ไม่ควรมีระดับชั้นความลึกมากจนเกินไปเพราะทำให้ยากต่อการทดสอบ
มาตรวัดที่ใช้ในการอธิบาย	ILND คือ ความลึกของขอบเขตที่ถูกปกปิดไว้ภายในการทำงานของฟังก์ชันหนึ่ง
ค่าที่ใช้ในการพิจารณา	ILND ค่าที่ใช้พิจารณาควรต่ำกว่า 4

ตารางที่ ข.5 ตัวอย่างนิยามซับเซตคลีนโค้ดและมาตรวัด ซอฟต์แวร์พร้อมค่าพิจารณา Function Arguments

กลุ่มของซับเซตคลีนโค้ด	Function
ลำดับที่	1.5
นิยาม	Function Arguments
แรงจูงใจ	พารามิเตอร์ในแต่ละฟังก์ชันไม่ควรมี จำนวน มากกว่า 5 พารามิเตอร์ เพราะต้องจัดการข้อมูลที่รับเข้ามา อาจเป็นสาเหตุทำให้การทำงานของคลาสมากและซับซ้อนยิ่งขึ้น
มาตรวัดที่ใช้ในการอธิบาย	NOP คือ จำนวนของพารามิเตอร์ที่ถูกส่งมาที่ฟังก์ชัน เพื่อการทำงานไม่ว่าจะเป็นพารามิเตอร์ที่ถูกอ้างถึงหรือถูกส่งออกก็ตาม
ค่าที่ใช้ในการพิจารณา	NOP ค่าที่ใช้พิจารณาควรต่ำกว่า 5

ตารางที่ ข.6 ตัวอย่างนิยามซับเซตคลีนโค้ดและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา Structured Programming

กลุ่มของซับเซตคลีนโค้ด	Function
ลำดับที่	1.6
นิยาม	Structured Programming

ตารางที่ ข.6 ตัวอย่างนิยามซับเซตคลีนโค้ดและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา Structured Programming (ต่อ)

แรงจูงใจ	การเขียนโปรแกรมต้องมีการทำงานในรูปแบบ Sequential State เพื่อป้องกันไม่ให้เกิดการทำงานที่ซ้ำซ้อนกัน
มาตรวัดที่ใช้ในการอธิบาย	ISS คือการทำงานของฟังก์ชันนั้นๆ สามารถเขียนอยู่ในรูปของกราฟ โดยแสดงลำดับขั้นตอนการทำงานที่มีจุดเริ่มต้นและสิ้นสุดเพียงที่เดียว
ค่าที่ใช้ในการพิจารณา	ISS ต้องมีค่าเป็นจริงเท่านั้น

ตารางที่ ข.7 ตัวอย่างนิยามซับเซตคลีนโค้ดและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา Class Organization

กลุ่มของซับเซตคลีนโค้ด	Class
ลำดับที่	1.7
นิยาม	Class Organization
แรงจูงใจ	คลาสจำเป็นต้องมีการวางแผนและออกแบบการทำงานเพื่อให้ง่ายต่อการนำไปใช้งาน
มาตรวัดที่ใช้ในการอธิบาย	1. DIT คือ การนับจำนวนของระดับชั้นของการสืบทอดคุณสมบัติจากคลาสที่พิจารณา 2. NOC คือ การนับจำนวนคลาสลูกทั้งหมดที่ทำการสืบทอดคุณสมบัติจากคลาสที่พิจารณา 3. NOI คือ การนับจำนวนส่วนต่อประสานที่ถูกประกาศในคลาสที่ถูกพิจารณา 4. NOIM คือ การนับจำนวนของคลาสที่ไม่ได้ถูกนำไปใช้ในการสืบทอดคุณสมบัติ จึงต้องมีการปกปิดเมท็อดหรือข้อมูลไว้
ค่าที่ใช้ในการพิจารณา	DIT ค่าที่ใช้พิจารณาควรต่ำกว่า 6 NOC ค่าที่ใช้พิจารณาควรต่ำกว่า 6 NOI ค่าที่ใช้พิจารณาควรต่ำกว่า 20 NOIM ค่าที่ใช้พิจารณาควรต่ำกว่า 1

ตารางที่ ข.8 ตัวอย่างนิยามซับเซตคลีนโค้ดและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา Encapsulation

กลุ่มของซับเซตคลีนโค้ด	Class
ลำดับที่	1.8
นิยาม	Encapsulation
แรงจูงใจ	คลาสจำเป็นต้องมีการปกปิดข้อมูลและรายละเอียดของการทำงานไว้ภายใน
มาตรวัดที่ใช้ในการอธิบาย	NPF คือ การนับจำนวนของฟิลด์ที่ถูกประกาศไว้ในคลาส และสามารถเรียกใช้งานได้จากภายนอก
ค่าที่ใช้ในการพิจารณา	NPF ค่าที่ใช้พิจารณาควรต่ำกว่า 1

ตารางที่ ข.9 ตัวอย่างนิยามซับเซตคลีนโค้ดและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา Classes Should Be Small

กลุ่มของซับเซตคลีนโค้ด	Class
ลำดับที่	1.9
นิยาม	Classes Should Be Small
แรงจูงใจ	คลาสจำเป็นต้องมีขนาดเล็กเพื่อความสะดวกใช้งานและง่ายต่อการบำรุงรักษา
มาตรวัดที่ใช้ในการอธิบาย	1. NOP คือ การนับจำนวนของพารามิเตอร์ที่ถูกส่งผ่านมาในฟังก์ชันหรือเมท็อดในการทำงานไม่ว่าจะเป็นพารามิเตอร์ที่ถูกอ้างถึงหรือส่งออกไปก็ตาม
มาตรวัดที่ใช้ในการอธิบาย	2. NOV คือ การนับจำนวนของตัวแปรที่ถูกประกาศไว้ในฟังก์ชันหรือเมท็อดที่ถูกพิจารณา 3. NOO คือ การนับจำนวนของเมท็อดที่ถูกโอเวอร์โหลดในคลาสที่พิจารณา 4. NOM คือ การนับจำนวนของเมท็อดที่ถูกประกาศไว้เพื่อเรียกใช้งานภายในหรือภายนอกคลาส 5. NFD คือ การนับจำนวนของฟิลด์ที่ถูกประกาศไว้ในคลาสที่พิจารณา

ตารางที่ ข.9 ตัวอย่างนิยามซับเซตคลีนโค้ดและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา Classes Should Be Small (ต่อ)

ค่าที่ใช้ในการพิจารณา	NOP ค่าที่ใช้พิจารณาควรต่ำกว่าหรือเท่ากับ 5 NOV ค่าที่ใช้พิจารณาควรต่ำกว่าหรือเท่ากับ 8 NOO ค่าที่ใช้พิจารณาควรต่ำกว่าหรือเท่ากับ 6 NOM ค่าที่ใช้พิจารณาควรต่ำกว่าหรือเท่ากับ 15 NFD ค่าที่ใช้พิจารณาควรต่ำกว่าหรือเท่ากับ 10

ตารางที่ ข.10 ตัวอย่างนิยามซับเซตคลีนโค้ดและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา Maintaining Cohesion

กลุ่มของซับเซตคลีนโค้ด	Class
ลำดับที่	1.10
นิยาม	Maintaining Cohesion
แรงจูงใจ	คลาสจำเป็นต้องมีการทำงานร่วมกันระหว่างข้อมูลและเม ท็อดที่อยู่ภายใน
มาตรวัดที่ใช้ในการอธิบาย	1. LCOM1 คือ การวัดระดับความสัมพันธ์ของการทำงานร่วมระหว่างเมท็อดหรือตัวแปรภายในคลาส โดยมีค่าระหว่าง 0-1 2. LCOM3 คือ การวัดระดับความสัมพันธ์ของการทำงานร่วมระหว่างเมท็อดหรือตัวแปรภายในคลาส เพื่อป้องกันปัญหาการหาค่าโคฮีชันผิดพลาดจากสูตรแรก โดยมีค่าระหว่าง 0-2
ค่าที่ใช้ในการพิจารณา	LCOM1 ค่าที่ใช้พิจารณาควรต่ำกว่าหรือเท่ากับ 0.5 LCOM3 ค่าที่ใช้พิจารณาควรต่ำกว่าหรือเท่ากับ 0.8

ตารางที่ ข.11 ตัวอย่างนิยามซับเซตคลีนโค้ดและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา Organizing for Change

กลุ่มของซับเซตคลีนโค้ด	Class
ลำดับที่	1.11
นิยาม	Organizing for Change

ตารางที่ ข.11 ตัวอย่างนิยามซับเซตคลีนโค้ดและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา Organizing for Change (ต่อ)

แรงจูงใจ	คลาสจำเป็นต้องมีการวางแผนและออกแบบการทำงานเพื่อให้สามารถรองรับการเปลี่ยนแปลงได้
มาตรวัดที่ใช้ในการอธิบาย	1. ACML คือ การวัดจำนวนการพึ่งพาการทำงานของเมท็อดที่พิจารณาภายในคลาส 2. ECM คือ การวัดจำนวนการเรียกใช้งานของเมท็อดที่พิจารณาภายในคลาส
ค่าที่ใช้ในการพิจารณา	ACML ค่าที่ใช้พิจารณาควรมากกว่า 0 ECML ค่าที่ใช้พิจารณาควรต่ำกว่าหรือเท่ากับ 50

ตารางที่ ข.12 ตัวอย่างนิยามซับเซตคลีนโค้ดและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา The Law of Demeter

กลุ่มของซับเซตคลีนโค้ด	Class
ลำดับที่	1.12
นิยาม	The Law of Demeter
แรงจูงใจ	คลาสจำเป็นต้องมีการวางแผนและออกแบบการทำงานเพื่อเรียกใช้งานเมท็อดและข้อมูลต่างๆเพื่อไม่ให้มีการทำงานความซ้ำซ้อน
มาตรวัดที่ใช้ในการอธิบาย	DMS คือ การวัดระยะห่างของการทำงานที่เกิดขึ้นเทียบกับส่วนการทำงานหลักของโปรแกรม
ค่าที่ใช้ในการพิจารณา	DMS ค่าที่ใช้พิจารณาควรต่ำกว่าหรือเท่ากับ 0.5

ตารางที่ ข.13 ตัวอย่างนิยามซับเซตคลีนโค้ดและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา Clean Boundary

กลุ่มของซับเซตคลีนโค้ด	Class
ลำดับที่	1.13
นิยาม	Clean Boundary
แรงจูงใจ	คลาสจำเป็นต้องมีการวางแผนและออกแบบขอบเขตการทำงานภายในคลาสและการติดต่อระหว่างคลาสให้ชัดเจน

ตารางที่ ข.13 ตัวอย่างนิยามซับเซตคลีนโค้ดและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา Clean Boundary (ต่อ)

มาตรวัดที่ใช้ในการอธิบาย	CBO คือ การวัดความสัมพันธ์ในการทำงานของวัตถุว่าต้องพึ่งพาการเรียกใช้งานจากฟิลด์หรือเมท็อดของวัตถุอื่น
ค่าที่ใช้ในการพิจารณา	CBO ค่าที่ใช้พิจารณาควรต่ำกว่าหรือเท่ากับ 6

ภาคผนวก ค
ตัวอย่างรายละเอียดร่องรอยไม่ดีพร้อมมาตรวัดซอฟต์แวร์และค่าพิจารณา

จากตารางได้นำนิยามร่องรอยไม่ดีมาจาก Mäntylä [16] โดยที่ Mäntylä มีการตั้งเกณฑ์ความซับซ้อนไว้ตั้งแต่ ค่า 0 – ค่า 5 ค่า 0 คือ ไม่มีความซับซ้อน สามารถวัดด้วยมาตรวัดซอฟต์แวร์พื้นฐาน ค่า 5 คือ มีความซับซ้อนมาก ต้องใช้มาตรวัดซอฟต์แวร์หลายชนิดวัดค่า ในงานวิทยานิพนธ์นี้จะเลือกเฉพาะค่าที่ถูกจัดอยู่ในระดับ 3-5 เท่านั้น รวมทั้งได้นำมาตรวัดที่ Mäntylä แนะนำไว้มาใช้วัดกลุ่มโค้ดร่องรอยไม่ดีด้วย เนื่องจากเป็นค่าที่ผ่านการสำรวจและเก็บข้อมูลจากผู้เชี่ยวชาญได้ในด้านการออกแบบและเขียนโปรแกรม ใช้เป็นอินพุตเซตสำหรับฟัซซี่ดังตัวอย่างตารางที่ ค.1 – ค.8

ตารางที่ ค.1 ตัวอย่างนิยามร่องรอยไม่ดีและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา Long Method

กลุ่มของร่องรอยไม่ดี	The Bloaters
ลำดับที่	1
นิยาม	Long Method
แรงจูงใจ	ฟังก์ชันหรือเมท็อดที่มีขนาดใหญ่และมีความซ้ำซ้อนก่อให้เกิดความไม่อิสระในการเรียกใช้งานและยากต่อการบำรุงรักษา
มาตรวัดที่ใช้ในการอธิบาย	1. NLOC คือ การนับจำนวนของบรรทัดของโค้ดที่ประกาศไว้ในการทำงานของฟังก์ชันหรือเมท็อดที่ถูกพิจารณา 2. NILI คือ การนับจำนวนของชุดคำสั่งที่ถูกเรียกใช้งานในฟังก์ชันที่หรือเมท็อดที่พิจารณา 3. CC คือ การนับจำนวนของเส้นทางทั้งหมดที่สามารถเกิดได้ในการทำงานภายในฟังก์ชันหรือเมท็อดที่พิจารณา 4. ILCC คือ การนับจำนวนเส้นทางทั้งหมดที่สามารถเป็นไปได้ของชุดคำสั่งในการทำงานภายในฟังก์ชันหรือเมท็อดที่พิจารณา
ค่าที่ใช้ในการพิจารณา	1. NLOC ค่าที่ใช้พิจารณามากกว่า 60 2. NILI ค่าที่ใช้พิจารณามากกว่า 200 3. CC ค่าที่ใช้พิจารณามากกว่า 30 4. ILCC ค่าที่ใช้พิจารณามากกว่า 60

ตารางที่ ค.2 ตัวอย่างนิยามร่องรอยไม่ดีและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา Large Class

กลุ่มของร่องรอยไม่ดี	The Bloaters
ลำดับที่	2
นิยาม	Large Class
แรงจูงใจ	คลาสที่มีหน้าที่การทำงานมากเกินไปและมีความซ้ำซ้อนของการทำงานภายในคลาสส่งผลให้ยากต่อการบำรุงรักษา
มาตรวัดที่ใช้ในการอธิบาย	1. LCOM1 คือ การวัดระดับความสัมพันธ์ของการทำงานร่วมระหว่างเมท็อดหรือตัวแปรภายในคลาส โดยมีค่าระหว่าง 0-1 2. LCOM3 คือ การวัดระดับความสัมพันธ์ของการทำงานร่วมระหว่างเมท็อดหรือตัวแปรภายในคลาส เพื่อป้องกันปัญหาการหาค่าโคฮีชันผิดพลาดจากสูตรแรก โดยมีค่าระหว่าง 0-2 3. NFD คือ การนับจำนวนของแอตทริบิวต์ที่ถูกประกาศไว้ในคลาสที่พิจารณา การนับจำนวนแอตทริบิวต์ แต่จะไม่นับจำนวนแอตทริบิวต์ที่รวมอยู่ในกรอบงาน 4. NOM คือ การนับจำนวนของเมท็อดที่ถูกประกาศไว้เพื่อเรียกใช้งานภายในหรือภายนอกคลาสที่ใช้พิจารณา
ค่าที่ใช้ในการพิจารณา	1. LCOM1 ค่าที่ใช้พิจารณามากกว่า 0.8 2. LCOM3 ค่าที่ใช้พิจารณามากกว่า 1.0 3. NFD ค่าที่ใช้พิจารณามากกว่า 20 4. NOM ค่าที่ใช้พิจารณามากกว่า 20

ตารางที่ ค.3 ตัวอย่าง นิยามร่องรอยไม่ดีและมาตรวัด ซอฟต์แวร์ พร้อมค่าพิจารณา: Long Parameter List

กลุ่มของร่องรอยไม่ดี	The Bloaters
ลำดับที่	3
นิยาม	Long Parameter List
แรงจูงใจ	จำนวนของพารามิเตอร์ที่ถูกส่งค่าผ่านฟังก์ชันหรือเมท็อดมากเกินไปก่อเกิดความไม่สอดคล้องในการทำงานและทำความเข้าใจยาก

ตารางที่ ค.3 ตัวอย่าง นิยามร่องรอยไม่ดีและมาตรวัด ซอฟต์แวร์ พร้อมค่าพิจารณา: Long Parameter List (ต่อ)

มาตรวัดที่ใช้ในการอธิบาย	NOP คือ การนับจำนวนของพารามิเตอร์ที่ถูกส่งผ่านมาในฟังก์ชันหรือเมท็อดในการทำงานไม่ว่าจะเป็นพารามิเตอร์ที่ถูกอ้างถึงหรือส่งออกไปก็ตาม
ค่าที่ใช้ในการพิจารณา	NOP ค่าที่ใช้พิจารณามากกว่า 5

ตารางที่ ค.4 ตัวอย่าง นิยามร่องรอยไม่ดีและมาตรวัด ซอฟต์แวร์ พร้อมค่าพิจารณา : Temporary Field

กลุ่มของร่องรอยไม่ดี	The Object-Orientation Abusers
ลำดับที่	4
นิยาม	Temporary Field
แรงจูงใจ	การจัดเก็บผลลัพธ์ชั่วคราวในระหว่างการทำงานของโปรแกรมเป็นจำนวนมาก ส่งผลให้เกิดความไม่เป็นระเบียบและไม่สามารถบ่งชี้ได้ว่า ส่วนไหนสำคัญหรือถูกใช้งานในสถานะปัจจุบัน
มาตรวัดที่ใช้ในการอธิบาย	AC-NPF คือ การวัดการพึงพาการทำงานของฟิลด์ที่สามารถเรียกใช้งานได้จากภายนอก
ค่าที่ใช้ในการพิจารณา	AC-NPF ค่าที่ใช้พิจารณาน้อยกว่าหรือเท่ากับ 3

ตารางที่ ค.5 ตัวอย่างนิยามร่องรอยไม่ดีและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา: Feature Envy

กลุ่มของร่องรอยไม่ดี	The Couplers
ลำดับที่	5
นิยาม	Feature Envy
แรงจูงใจ	การเรียกใช้งานเมท็อดหรือข้อมูลจากคลาสอื่นมากกว่าคลาสที่เป็นเจ้าของเมท็อดนั้นเป็นการออกแบบที่ไม่เหมาะสมและไม่เป็นอิสระในการทำงาน
มาตรวัดที่ใช้ในการอธิบาย	SAC-OMF คือ การวัดความผิดปกติของการพึ่งพาการทำงานจากการเรียกใช้งานเมท็อดและฟิลด์จากภายนอก

ตารางที่ ค.5 ตัวอย่างนิยามร่องรอยไม่ดีและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา:Feature Envy (ต่อ)

กลุ่มของร่องรอยไม่ดี	The Couplers
ลำดับที่	5
นิยาม	Feature Envy
แรงจูงใจ	การเรียกใช้งานเมท็อดหรือข้อมูลจากคลาสอื่นมากกว่าคลาสที่เป็นเจ้าของเมท็อดนั้นเป็นการออกแบบที่ไม่เหมาะสมและไม่เป็นอิสระในการทำงาน
มาตรวัดที่ใช้ในการอธิบาย	SAC-OMF คือ การวัดความผิดปกติของการพึ่งพาการทำงานจากการเรียกใช้งานเมท็อดและฟิลด์จากภายนอก
ค่าที่ใช้ในการพิจารณา	SAC-OMF ค่าที่ใช้พิจารณามากกว่า 0

ตารางที่ ค.6 ตัวอย่างนิยามร่องรอยไม่ดีและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา: Inappropriate Intimacy

กลุ่มของร่องรอยไม่ดี	The Couplers
ลำดับที่	6
นิยาม	Inappropriate Intimacy
แรงจูงใจ	การเรียกใช้งานเมท็อดภายในแต่ละคลาสที่มีโครงสร้างการเรียกใช้งานที่ซับซ้อนจากการออกแบบที่ไม่เหมาะสมทำให้เรียกใช้งานยุ่งยากและใช้เวลานาน
มาตรวัดที่ใช้ในการอธิบาย	SEC-IM คือ การวัดความผิดปกติของการเรียกใช้งานเมท็อดที่อยู่ภายในคลาส.
ค่าที่ใช้ในการพิจารณา	SEC-IM ค่าที่ใช้พิจารณามากกว่า 5

ตารางที่ ค.7 ตัวอย่างนิยามร่องรอยไม่ดีและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา: Lazy Class

กลุ่มของร่องรอยไม่ดี	The Dispensables
ลำดับที่	7
นิยาม	Lazy Class
แรงจูงใจ	คลาสที่มีการทำงานที่ไม่สำคัญหรือไม่มีประโยชน์ต่อโปรแกรม ส่งผลให้เสียเวลาในการบำรุงรักษา

ตารางที่ ค.7 ตัวอย่างนิยามร่องรอยไม่ดีและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา:Lazy Class (ต่อ)

มาตรวัดที่ใช้ในการอธิบาย	LMFC คือ การวัดปริมาณของเมท็อดและฟิลด์ภายในคลาสที่ประกาศให้ถูกเรียกใช้งาน
ค่าที่ใช้ในการพิจารณา	LMFC ค่าที่ใช้พิจารณามากกว่า 0

ตารางที่ ค.8 ตัวอย่างนิยามร่องรอยไม่ดีและมาตรวัดซอฟต์แวร์พร้อมค่าพิจารณา: Bad Comment

กลุ่มของร่องรอยไม่ดี	Others
ลำดับที่	8
นิยาม	Bad Comment
แรงจูงใจ	โปรแกรมที่ไม่มีคำอธิบายการทำงานของโปรแกรม ทำให้เป็นเรื่องยากที่จะทำความเข้าใจการทำงานของโปรแกรม และส่วนการทำงานที่มีความซับซ้อนหรือข้อจำกัดบางอย่างในการทำงาน ทำให้ส่งผลกระทบต่อเวลาในการบำรุงรักษาโปรแกรม
มาตรวัดที่ใช้ในการอธิบาย	PCC คือ การวัด อัตราร้อยละของคำอธิบายการทำงานภายในโปรแกรม
ค่าที่ใช้ในการพิจารณา	PCC ค่าที่ใช้พิจารณาเท่ากับ 0

ภาคผนวก ง
วิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ

งานวิทยานิพนธ์นี้ได้ศึกษาวิธีการแก้ไขร่องรอยไม่ดีจาก Martin Fowler [6, 8] และเลือกแก้ไขร่องรอยไม่ดีทั้งหมด 9 รายการ ซึ่งในแต่ละรายการผู้วิจัยได้เรียบเรียงและสร้างวิธีแก้ไขร่องรอยไม่ดีและความคลุมเครือไว้หลายวิธีตามลักษณะของปัญหาที่เกิด มีรายละเอียดตามตารางดังนี้

ตารางที่ ง.1 วิธีการแก้ไขร่องรอยไม่ดี

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการร่องรอยไม่ดี							เงื่อนไขในการเลือกวิธีการปรับปรุง
		Long Method	Large Class	Long Parameter List	Temporary Field	Feature Envy	Inappropriate Intimacy	Lazy Class	
1	Add Parameter				✓				1. ปัญหาเกิดจากการประกาศแอตทริบิวต์ เพื่อใช้งานอย่างไม่เหมาะสม โดยทั่วไป ไม่มีความจำเป็นต้องนำค่าของแอตทริบิวต์ไปใช้งาน ดังนั้นควรยุบแอตทริบิวต์ให้อยู่ในรูปของพารามิเตอร์ในแต่ละเมท็อดแทน
2	Change Bidirectional Association to Unidirectional Association						✓		1. ปัญหาเกิดจากแอตทริบิวต์ที่เก็บตัวแปรเพื่อรับ และส่งค่าระหว่างกัน โดยทำหน้าที่ เป็นทั้งผู้รับและผู้ส่ง ซึ่งทำให้เกิดความสับสน ควรที่จะกำหนดทิศทางการทำงานให้เป็นทิศทางเดียวกันเพื่อสะดวกต่อการแก้ไขปัญหาต่างๆ
3	Collapse Hierarchy							✓	1. ปัญหาที่เกิดจากการสืบทอดคุณสมบัติ โดยที่คลาสที่สืบทอดมามีคุณสมบัติการทำงานที่น้อยมากหรือแทบไม่แตกต่างจากคลาสแม่ที่สืบทอดเลย ควรยุบกลับไปรวมกับคลาส แม่เพื่อสามารถแก้ไขได้สะดวกกว่า

ตารางที่ ง.1 วิธีการแก้ไขร่องรอยไม่ดี (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการร่องรอยไม่ดี							เงื่อนไขในการเลือกวิธีการปรับปรุง
		Long Method	Large Class	Long Parameter List	Temporary Field	Feature Envy	Inappropriate Intimacy	Lazy Class	
4	Decompose Conditional	✓							1. ปัญหาที่เกิดจากนิพจน์ที่สร้างซ้ำซ้อนๆ กัน สามารถแก้ไขได้ด้วยการ แปลงให้อยู่ในรูปฟังก์ชันแบบง่าย แล้วจึงแยกย่อยออกไป จะทำให้ความซับซ้อนที่เกิดขึ้นลดน้อยลง
5	Duplicate observed data		✓						1. ปัญหาเกิดจากจำนวนของตัวแอตทริบิวต์มีจำนวนมาก ถ้ามีการเปลี่ยนแปลงหรือแก้ไขค่าแอตทริบิวต์หนึ่ง จะมีผลกระทบต่อแอตทริบิวต์อื่นๆ ควรตัดโค้ดในส่วนที่ซ้ำซ้อนออกเป็นคลาสใหม่หรือรวมเป็นฟังก์ชัน เพื่อลดความซ้ำซ้อนของหน้าที่งานที่เกิดขึ้น
6	Encapsulate Field				✓			✓	1. ปัญหาเกิดจากการประกาศแอตทริบิวต์ที่ถูกใช้ เพื่อ เก็บค่าชั่วคราว โดยแอตทริบิวต์ ที่ถูกประกาศ ค่าไว้ อาจถูกเรียกใช้งาน ไม่กี่ครั้ง หรือเพียงครั้งเดียว ที่เป็นการเรียกใช้งานจากภายนอก ควรแก้ไขให้มีการปกปิดข้อมูลหรือ การกำหนด ลำดับขั้นในการเข้าถึงข้อมูล ด้วย การเพิ่มเงื่อนไ ขเพื่อตรวจสอบค่า เพื่อทราบว่ามี หน้า ที่รับผิดชอบหรือลำดับขั้นการทำงานอย่างไร เพื่อแก้ไขร่องรอยไม่ดีที่เกิดขึ้นได้ง่ายขึ้น

ตารางที่ ง.1 วิธีการแก้ไขร่องรอยไม่ดี (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการร่องรอยไม่ดี							เงื่อนไขในการเลือกวิธีการปรับปรุง
		Long Method	Large Class	Long Parameter List	Temporary Field	Feature Envy	Inappropriate Intimacy	Lazy Class	
7	Extract Class		✓				✓		1. ปัญหาเกิดจากจำนวนของเมท็อดหรือแอตทริบิวต์ที่มีจำนวนมาก สามารถแบ่งคลาสออกเป็นคลาสย่อยตามหน้าที่รับผิดชอบหรือตามขอบเขตการทำงาน เพื่อไม่ให้เกิดความซ้ำซ้อน 2. ปัญหาเกิดจากการประกาศแอตทริบิวต์ในการใช้ งานที่ไม่เหมาะสม อาจเป็นการประกาศใช้ งาน แค่ชั่วคราวหรือมีเงื่อนไขซ้ำซ้อนไม่สอดคล้องกับการทำงานในคลาสหลัก ควรแบ่งส่วนที่เกี่ยวข้องกับแอตทริบิวต์นั้นๆไปยังคลาสใหม่ 3. ปัญหาเกิดจากเรียกใช้งานเมท็อดในคลาส ที่จำเป็นต้องเรียกพร้อม กันหรือควบคู่กันเสมอ ควรแบ่งเมท็อดส่วนนี้ออกเป็นคลาสย่อยเพื่อความสะดวกในการเรียกใช้งาน

ตารางที่ ง.1 วิธีการแก้ไขร่องรอยไม่ดี (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการร่องรอยไม่ดี							เงื่อนไขในการเลือกวิธีการปรับปรุง
		Long Method	Large Class	Long Parameter List	Temporary Field	Feature Envy	Inappropriate Intimacy	Lazy Class	
8	Extract Interface		✓						1. ปัญหาเกิดจากจำนวนของส่วนต่อประสานที่เชื่อมกันมีจำนวนมาก สามารถแบ่งส่วนต่อประสานที่ถูกเชื่อมต่อไว้ออกเป็นส่วนย่อยๆ ตามพฤติกรรมของส่วนเชื่อมต่อว่ามีหน้าที่ทำงานอะไร
9	Extract Method	✓				✓			1. ปัญหาเกิดจากจำนวนบรรทัดของโค้ดมีมากเกินไป สามารถแก้ไขได้โดยแบ่งโค้ดให้อยู่ในรูปของเมท็อดย่อยๆ ตามลักษณะการทำงาน 2. ปัญหาเกิดจากการลำดับการเรียกใช้งานเมท็อดในแต่ละคลาส อาจมีบางกรณีที่ไม่มีความจำเป็นต้องเรียกเมท็อดนี้มาทำงาน แต่มีบางส่วนจำเป็นต้องเรียกใช้งานหรือลำดับการทำงาน จึงเกิดการเรียกใช้งานที่ไม่เหมาะสม ควรแยกส่วนการทำงานให้เป็นอิสระ

ตารางที่ ง.1 วิธีการแก้ไขร่องรอยไม่ดี (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการร่องรอยไม่ดี							เงื่อนไขในการเลือกวิธีการปรับปรุง
		Long Method	Large Class	Long Parameter List	Temporary Field	Feature Envy	Inappropriate Intimacy	Lazy Class	
10	Extract Subclass		✓						1. ปัญหาเกิดจากจำนวนของเมท็อดหรือแอตทริบิวต์ที่มีจำนวนมาก และสามารถแบ่งคลาสออกเป็นคลาสลูกเพื่อสืบทอดการทำงานจากคลาสแม่ โดยมีคลาสลูกที่มีคุณสมบัติเพิ่มเติมจากคลาสแม่ได้
11	Hide Delegate						✓		1. ปัญหาเกิดจากเรียกใช้งานคลาสหลายตัวที่อยู่ภายในคลาส เมื่อวิเคราะห์การทำงานแล้ว สามารถเรียกใช้ข้อมูลที่ต้องการจากคลาสเดียวกันได้ ทำให้สามารถลดจำนวนการเรียกใช้งานลงได้
12	Inline Class							✓	1. เป็นคลาสที่มีลักษณะและหน้าที่การทำงานน้อย ในกรณีที่คลาสมีลักษณะการทำงานที่ใกล้เคียงกัน สามารถยุบรวมเพื่อให้การทำงานประเภทเดียวกันอยู่ด้วยกัน ทำให้สะดวกแก่การเรียกใช้งาน

ตารางที่ ง.1 วิธีการแก้ไขร่องรอยไม่ดี (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการร่องรอยไม่ดี							เงื่อนไขในการเลือกวิธีการปรับปรุง
		Long Method	Large Class	Long Parameter List	Temporary Field	Feature Envy	Inappropriate Intimacy	Lazy Class	
13	Inline Method					✓			1. ปัญหาเกิดจากการทำงานของเมท็อดที่อยู่ในตำแหน่งไม่เหมาะสม ทำให้ระหว่างการทำงานอาจถูกเรียกจากเมท็อดอื่นๆ นอกจากนี้มีการทำงานบางส่วนถูกจำกัดอยู่ภายในเมท็อดที่ถูกเรียกบ่อยครั้ง ทั้งที่ไม่มีหน้าที่เกี่ยวข้องโดยตรง
14	Introduce Null Object				✓				1. ปัญหาเกิดจากการประกาศแอตทริบิวต์ ที่ไม่มีค่าเริ่มต้น ในการทำงาน ทำให้ไม่สามารถระบุสถานะได้ ดังนั้นควรระบุค่าเริ่มต้นสถานะให้กับแอตทริบิวต์ ซึ่งแอตทริบิวต์ที่สามารถแก้ไขได้โดยวิธีนี้จำเป็นต้องมีข้อมูลเป็นประเภทวัตถุ หากกรณีที่แอตทริบิวต์เป็นประเภท ตัวเลข จะไม่สามารถกำหนด ค่าว่างได้ เนื่องจากชนิดข้อมูลไม่สอดคล้องกัน

ตารางที่ ง.1 วิธีการแก้ไขร่องรอยไม่ดี (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการร่องรอยไม่ดี							เงื่อนไขในการเลือกวิธีการปรับปรุง
		Long Method	Large Class	Long Parameter List	Temporary Field	Feature Envy	Inappropriate Intimacy	Lazy Class	
15	Introduce Parameter Object	✓		✓					1. ปัญหาเกิดจากจำนวนของพารามิเตอร์ ถูกรับเข้าทำงานในเมท็อดมีจำนวนมาก เมื่อยุบรวมให้อยู่ในรูปของวัตถุแล้ว อาจจะทำให้การทดสอบเงื่อนไขต่างๆ และการเรียกใช้งานอื่นๆ ลดลงได้ 2. ปัญหาเกิดจากจำนวนของพารามิเตอร์ถูกรับเข้าทำงานในเมท็อดมีจำนวนมาก และมีกลุ่มพารามิเตอร์ที่ถูกใช้งานควบคู่กันเสมอ ควรนำมารวมกันให้อยู่ในรูปของวัตถุ เพื่อลดจำนวนพารามิเตอร์นำเข้า
16	Move Field						✓		1. ปัญหาเกิดจากแอตทริบิวต์ที่ถูกเรียกใช้งานบ่อยๆ จึง ควรย้ายแอตทริบิวต์ให้อยู่ใกล้หรืออยู่ในขอบเขตเดียวกับผู้เรียกใช้งาน เพื่อสะดวกในการทำงานและแก้ไขปัญหา

ตารางที่ ง.1 วิธีการแก้ไขร่องรอยไม่ดี (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการร่องรอยไม่ดี							เงื่อนไขในการเลือกวิธีการปรับปรุง
		Long Method	Large Class	Long Parameter List	Temporary Field	Feature Envy	Inappropriate Intimacy	Lazy Class	
17	Move Method					✓			1. ปัญหาเกิดจากการทำงานของเมท็อดอยู่ในตำแหน่งที่ไม่เหมาะสม ทำให้ระหว่างการทำงานอาจถูกเรียกจากเมท็อดต่างๆ ที่ไม่เกี่ยวข้องโดยตรง เนื่องจากมีการทำงานบางส่วนถูกจำกัดอยู่ภายในเมท็อดที่ถูกเรียกใช้งาน จึงเป็นเหตุให้เมท็อดนี้ถูกเรียกใช้งานอยู่บ่อยๆ ทั้งที่ไม่มีความเกี่ยวข้องกับการทำงานโดยตรง
18	Preserve Whole Object	✓		✓					1. ปัญหาเกิด จาก การรับค่าไปเก็บไว้ในตัวแปรชั่วคราวหลายตัวก่อนที่จะส่งค่าไป คำนวณ ต่อไปในฟังก์ชันถัดไ ป ทำให้ทำงานยุ่งยากเนื่องจากต้องมีการส่งค่าไปหลายครั้ง

ตารางที่ ง.1 วิธีการแก้ไขร่องรอยไม่ดี (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการร่องรอยไม่ดี							เงื่อนไขในการเลือกวิธีการปรับปรุง
		Long Method	Large Class	Long Parameter List	Temporary Field	Feature Envy	Inappropriate Intimacy	Lazy Class	
18	Preserve Whole Object								2. ปัญหาเกิดจากจำนวนของพารามิเตอร์ ถูกรับเข้าทำงานในเมท็อดจำนวนมาก ซึ่งพารามิเตอร์บางตัวไม่จำเป็นต้องนำเข้า เพราะสามารถคำนวณหาค่าได้จากพารามิเตอร์ตัวอื่นในคลาสหรือจากแอตทริบิวต์ตัวอื่นที่รับค่าเข้ามา ดังนั้นสามารถตัดแอตทริบิวต์ ตัวนั้นออกและคำนวณหลังจากรับค่าเข้ามาใช้งานในฟังก์ชัน
19	Rename Method						✓		1. ปัญหาเกิดจากชื่อเมท็อดที่ประกาศไว้มีชื่อซ้ำซ้อนกันหรือไม่สัมพันธ์กับการทำงาน จึงจำเป็นต้องกำหนดชื่อที่ใช้อ้างถึง ให้ใหม่ เพื่อให้ เรียกใช้ได้สะดวก เพื่อไม่ให้เกิดปัญหาการเรียกใช้งานซ้ำซ้อน

ตารางที่ ง.1 วิธีการแก้ไขร่องรอยไม่ดี (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการร่องรอยไม่ดี							เงื่อนไขในการเลือกวิธีการปรับปรุง
		Long Method	Large Class	Long Parameter List	Temporary Field	Feature Envy	Inappropriate Intimacy	Lazy Class	
20	Replace Delegation with Inheritance						✓		1. ปัญหาเกิดจากเมท็อดที่ถูกสร้างขึ้นมาอย่างอิสระในขณะการดำเนินงาน หากสามารถแก้ไขให้อยู่ในของการสืบทอดได้ จะทำให้ลดขั้นตอนวิธีการสร้างเมท็อดนั้นลงและสามารถเรียกใช้ได้ตามคุณสมบัติของคลาสที่รับการสืบทอดมา
21	Replace Method with Method Object	✓							1. ปัญหาเกิดจากการทำงานภายในฟังก์ชัน ที่มีส่วนเรียกใช้ข้อมูลหรือการคำนวณหาค่าที่ต้องการ ซึ่งทำให้เกิดความ ซับซ้อนหรือมีปริมาณข้อมูลที่ ต้องคำนวณมาก ควรตัดแบ่งส่วนออกเป็นเมท็อดใหม่ ที่ส่งคืนผลลัพธ์เป็นชนิดของข้อมูล ที่คำนวนเสร็จเรียบร้อยแล้ว

ตารางที่ ง.1 วิธีการแก้ไขร่องรอยไม่ดี (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการร่องรอยไม่ดี							เงื่อนไขในการเลือกวิธีการปรับปรุง
		Long Method	Large Class	Long Parameter List	Temporary Field	Feature Envy	Inappropriate Intimacy	Lazy Class	
22	Replace Parameter with Method			✓					1. ปัญหาเกิดจากจำนวนของพารามิเตอร์ ที่รับเข้าทำงานในเมท็อดมีจำนวนมาก ซึ่งพารามิเตอร์บางตัวไม่จำเป็นต้องนำเข้า เพราะสามารถคำนวณหาค่าจากเมท็อดอื่นได้ ทั้งนี้ยังสามารถตัดพารามิเตอร์ออกและคำนวณค่าใหม่ภายในฟังก์ชันได้
23	Replace Temp with Query	✓							1. ปัญหาเกิดจากการนำตัวแปรชั่วคราวไปใช้ งานในหลายๆที่ภายในฟังก์ชัน ควรแปลงตัวแปรชั่วคราวนี้ให้อยู่ในรูปของเมท็อดใหม่ เพื่อนำไปใช้งานอย่างถาวร

ตารางที่ ง.2 วิธีการแก้ไขโค้ดที่มีความคลุมเครือ

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการความคลุมเครือ												เงื่อนไขในการเลือกวิธีการปรับปรุง
		Ambiguity in Comment	Ambiguity in Method	Ambiguity in One Thing	Ambiguity in Nesting Depth	Ambiguity in Arguments	Ambiguity in Structured Programming	Ambiguity in Class Organization	Ambiguity in Encapsulation	Ambiguity in Classes	Ambiguity in Cohesion	Ambiguity for Change	Ambiguity of Demeter	Ambiguity in Boundary	
1	Add Comment	✓													1. จำนวนรายละเอียดข้อมูลการทำงานของโปรแกรมมีไม่เพียงพอทำให้ใช้เวลาทำความเข้าใจการทำงานต่างๆ ในคลาสมาก
2	Change Bidirectional Association to Unidirectional Association													✓	1. ปัญหาเกิดจากแอตทริบิวต์และเมท็อดสามารถรับและส่งค่าระหว่างกันได้ โดยเป็นทั้งผู้รับและผู้ส่งซึ่งอาจทำให้เกิดความสับสนในการทำงาน จึงควรกำหนดทิศทางการทำงานให้เป็นไปในทางเดียวเพื่อความสะดวกเมื่อต้องการแก้ไขปัญหาต่างๆ
3	Collapse Hierarchy			✓				✓			✓	✓			1. ปัญหาเกิดจากการทำงานของคลาสที่สืบทอดคุณสมบัติ มีลักษณะการทำงานที่คล้ายกันหรือใกล้เคียงกันอย่างมาก ควรปรับแก้โดยการยุบรวมกันเพื่อไม่ให้เกิดความซ้ำซ้อน

ตารางที่ ง.2 วิธีการแก้ไขโค้ดที่มีความคลุมเครือ (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการความคลุมเครือ													เงื่อนไขในการเลือกวิธีการปรับปรุง
		Ambiguity in Comment	Ambiguity in Method	Ambiguity in One Thing	Ambiguity in Nesting Depth	Ambiguity in Arguments	Ambiguity in Structured Programming	Ambiguity in Class Organization	Ambiguity in Encapsulation	Ambiguity in Classes	Ambiguity in Cohesion	Ambiguity for Change	Ambiguity of Demeter	Ambiguity in Boundary	
3	Collapse Hierarchy														2. ปัญหาเกิดจากคลาสที่ได้รับการสืบทอดคุณสมบัติมีจำนวนมาก และมีคลาสลูกบางตัวที่มีคุณสมบัติใกล้เคียงกับคลาสแม่ที่ได้รับการสืบทอด ควรยุบคลาสลูกกลับไปรวมกับคลาสแม่
4	Decompose Conditional		✓		✓										1. ปัญหาที่เกิดจากนิพจน์ที่สร้างซ้อนๆ กัน สามารถแก้ไขได้โดยแปลงให้อยู่ในรูปของฟังก์ชันง่ายๆ แล้วแยกย่อยออกไป จะทำให้ความซับซ้อนที่เกิดขึ้นลดน้อยลง
5	Encapsulate Field								✓						1. ปัญหาเกิดจากการประกาศใช้งานแอตทริบิวต์ในคลาส โดยที่อาจถูกเรียกใช้งานน้อยหรือเพียงครั้งเดียวซึ่งเป็นการเรียกใช้งาน

ตารางที่ ง.2 วิธีการแก้ไขโค้ดที่มีความคลุมเครือ (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการความคลุมเครือ													เงื่อนไขในการเลือกวิธีการปรับปรุง
		Ambiguity in Comment	Ambiguity in Method	Ambiguity in One Thing	Ambiguity in Nesting Depth	Ambiguity in Arguments	Ambiguity in Structured Programming	Ambiguity in Class Organization	Ambiguity in Encapsulation	Ambiguity in Classes	Ambiguity in Cohesion	Ambiguity for Change	Ambiguity of Demeter	Ambiguity in Boundary	
5	Encapsulate Field														จากภายนอกและแก้ไขให้มีการปกปิดข้อมูลหรือมีลำดับชั้นในการเข้าถึงข้อมูลหรือเพิ่มเงื่อนไขสำหรับตรวจสอบค่าในการทำงาน
6	Extract Class			✓						✓	✓	✓	✓	✓	1. ปัญหาเกิดจากจำนวนของเมท็อดหรือแอตทริบิวต์ที่มีจำนวนมาก ควรแบ่งคลาสออกเป็นคลาสย่อยตามหน้าที่รับผิดชอบหรือตามขอบเขตการทำงาน เพื่อไม่ให้เกิดความซ้ำซ้อน 2. ปัญหาเกิดจากเรียกใช้งานผ่านคลาสต่างๆ จำนวนมากและบางคลาสมีการทำงานที่สามารถแบ่งแยกย่อยออกได้หลังจากแยกคลาสนี้ออกเป็นคลาสย่อยๆ แล้ว จะทำให้ลำดับการทำงานสั้นลง

ตารางที่ ง.2 วิธีการแก้ไขโค้ดที่มีความคลุมเครือ (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการความคลุมเครือ												เงื่อนไขในการเลือกวิธีการปรับปรุง
		Ambiguity in Comment	Ambiguity in Method	Ambiguity in One Thing	Ambiguity in Nesting Depth	Ambiguity in Arguments	Ambiguity in Structured Programming	Ambiguity in Class Organization	Ambiguity in Encapsulation	Ambiguity in Classes	Ambiguity in Cohesion	Ambiguity for Change	Ambiguity of Demeter	Ambiguity in Boundary	
6	Extract Class														3.ปัญหาเกิดจากคลาสเมท็อดและแอตทริบิวต์ทำงานไม่สอดคล้องกัน หากวิเคราะห์ให้ดีจะพบว่า ส่วนของเมท็อดและแอตทริบิวต์สามารถแบ่งออกตามประเภทหรือลักษณะการทำงานได้ จึงควรแบ่งเป็นอีกหนึ่งคลาสเพื่อง่ายต่อการเรียกใช้งาน
7	Extract Interface							✓							1. ปัญหาเกิดจากจำนวนส่วนต่อประสานที่เชื่อมต่อกันมีจำนวนมาก ควรแบ่งส่วนต่อประสานออกเป็นส่วนย่อยๆ ตามพฤติกรรมของการทำงาน

ตารางที่ ง.2 วิธีการแก้ไขโค้ดที่มีความคลุมเครือ (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการความคลุมเครือ												เงื่อนไขในการเลือกวิธีการปรับปรุง
		Ambiguity in Comment	Ambiguity in Method	Ambiguity in One Thing	Ambiguity in Nesting Depth	Ambiguity in Arguments	Ambiguity in Structured Programming	Ambiguity in Class Organization	Ambiguity in Encapsulation	Ambiguity in Classes	Ambiguity in Cohesion	Ambiguity for Change	Ambiguity of Demeter	Ambiguity in Boundary	
8	Extract Method				✓							✓	✓		1. ปัญหาเกิดจากการทำงานของโค้ดที่มีรูปแบบซับซ้อนในฟังก์ชัน ซึ่งการทำงานในแต่ละส่วน สามารถแก้ไขได้โดยแบ่งโค้ดให้อยู่ในรูปของเมท็อดย่อยๆ เนื่องจากมีลำดับขั้นตอนทำงานที่ชัดเจน
															2. ปัญหาเกิดจากจำนวนบรรทัดของปริมาณโค้ดที่มีมากเกินไป สามารถแก้ไขได้โดยแบ่งโค้ดให้อยู่ในรูปของเม็ทอดย่อยๆ ตามลักษณะการทำงาน
9	Extract Subclass									✓		✓			1. ปัญหาเกิดจากจำนวนของเม็ทอดหรือแอตทริบิวต์ที่มีจำนวนมาก ควรแบ่งคลาสออกเป็นคลาสลูกเพื่อสืบทอดการทำงานจากคลาสแม่ได้ โดยคลาสลูกสามารถมีคุณสมบัติเพิ่มเติมจากคลาสแม่ได้

ตารางที่ ง.2 วิธีการแก้ไขโค้ดที่มีความคลุมเครือ (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการความคลุมเครือ													เงื่อนไขในการเลือกวิธีการปรับปรุง
		Ambiguity in Comment	Ambiguity in Method	Ambiguity in One Thing	Ambiguity in Nesting Depth	Ambiguity in Arguments	Ambiguity in Structured Programming	Ambiguity in Class Organization	Ambiguity in Encapsulation	Ambiguity in Classes	Ambiguity in Cohesion	Ambiguity for Change	Ambiguity of Demeter	Ambiguity in Boundary	
10	Inline Class			✓				✓			✓	✓			1. เป็นคลาสมีลักษณะการทำงานที่น้อย ในกรณีที่คลาสมีลักษณะการทำงานที่ใกล้เคียงกัน สามารถยุบกลับไปรวมเพื่อให้คลาสทำงานประเภทเดียวกันอยู่ด้วยกัน สะดวกต่อการเรียกใช้งาน 2. ปัญหาเกิดจากคลาสที่มีลักษณะการทำงานที่คล้ายกันหรือสามารถเป็นคลาสย่อยได้ ควรปรับแก้โดยการยุบรวมเพื่อไม่ให้เกิดความซ้ำซ้อนกัน 3. เป็นคลาสที่ได้รับการสืบทอดคุณสมบัติ มีหน้าที่รับผิดชอบหรือภาระในการทำงานน้อย ถ้ามีคลาสที่มีลักษณะการทำงานใกล้เคียงกัน สามารถยุบรวมกัน ทำให้สะดวกต่อการเรียกใช้งาน

ตารางที่ ง.2 วิธีการแก้ไขโค้ดที่มีความคลุมเครือ (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการความคลุมเครือ													เงื่อนไขในการเลือกวิธีการปรับปรุง
		Ambiguity in Comment	Ambiguity in Method	Ambiguity in One Thing	Ambiguity in Nesting Depth	Ambiguity in Arguments	Ambiguity in Structured Programming	Ambiguity in Class Organization	Ambiguity in Encapsulation	Ambiguity in Classes	Ambiguity in Cohesion	Ambiguity for Change	Ambiguity of Demeter	Ambiguity in Boundary	
11	Inline Method		✓	✓							✓				1. ปัญหาเกิดจากโค้ดที่มีการทำงานซ้ำซ้อนกัน ควรรวมกันและแยกเป็นเมท็อดเดียว ให้สามารถเรียกใช้งานได้ง่ายขึ้น 2. ปัญหาเกิดจากการทำงานของโค้ดที่อยู่ภายในฟังก์ชัน มีการทำงานมากกว่าหนึ่งอย่างหรือหนึ่งประเภท ควรรวมกันและแยกออกเป็นเมท็อดย่อยๆ ให้สามารถเรียกใช้งานได้ง่าย
12	Introduce assertion						✓								1. ปัญหาเกิดจากข้อผิดพลาดการทำงานในโปรแกรม อาจมีบางกรณีไม่เป็นไปตามวัตถุประสงค์ ดังนั้นควรใช้การยืนยันค่าเพื่อตรวจสอบค่าก่อนที่จะเกิดปัญหาที่คาดไม่ถึงเกิดขึ้น เป็นวิธีการป้องกันและพร้อมสำหรับการแก้ไข

ตารางที่ ง.2 วิธีการแก้ไขโค้ดที่มีความคลุมเครือ (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการความคลุมเครือ												เงื่อนไขในการเลือกวิธีการปรับปรุง
		Ambiguity in Comment	Ambiguity in Method	Ambiguity in One Thing	Ambiguity in Nesting Depth	Ambiguity in Arguments	Ambiguity in Structured Programming	Ambiguity in Class Organization	Ambiguity in Encapsulation	Ambiguity in Classes	Ambiguity in Cohesion	Ambiguity for Change	Ambiguity of Demeter	Ambiguity in Boundary	
13	Introduce local extension												✓		1. ปัญหาเกิดจากเรียกใช้งานคลาสที่อยู่ภายนอกหรือต้องเรียกผ่านคลาสอื่นๆ ทำให้ลำดับชั้นหรือความลึกของการทำงานมีมาก ควรแก้ไขโดยนำค่ามาเก็บไว้ภายในคลาสเพื่อสะดวกในการเรียกใช้งาน
14	Introduce Null Object						✓								1. ปัญหาเกิดจากการประกาศแอตทริบิวต์ในการใช้งานโดยไม่มีค่าเริ่มต้นให้แก่แอตทริบิวต์ อาจเป็นสาเหตุที่ทำให้เกิดความผิดปกติในการทำงานของโปรแกรมไม่เป็นไปตามโครงสร้างที่ออกแบบไว้

ตารางที่ ง.2 วิธีการแก้ไขโค้ดที่มีความคลุมเครือ (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการความคลุมเครือ													เงื่อนไขในการเลือกวิธีการปรับปรุง
		Ambiguity in Comment	Ambiguity in Method	Ambiguity in One Thing	Ambiguity in Nesting Depth	Ambiguity in Arguments	Ambiguity in Structured Programming	Ambiguity in Class Organization	Ambiguity in Encapsulation	Ambiguity in Classes	Ambiguity in Cohesion	Ambiguity for Change	Ambiguity of Demeter	Ambiguity in Boundary	
15	Introduce Parameter Object		✓		✓										1. ปัญหาเกิดจากจำนวนของพารามิเตอร์รับเข้าทำงานในเมท็อดมีจำนวนมาก หากยุบรวมให้อยู่ในรูปของวัตถุ เป็นผลให้สามารถลดการทดสอบเงื่อนไข และการเรียกใช้งานอื่นๆ ลงได้ 2. ปัญหาเกิดจากจำนวนของพารามิเตอร์รับเข้าทำงานในเม็ทอดมีจำนวนมาก และมีกลุ่มพารามิเตอร์ที่ถูกใช้งานควบคู่กันเสมอ ควรนำมารวมกันให้อยู่ในรูปของวัตถุเพื่อลดจำนวนพารามิเตอร์นำเข้า

ตารางที่ ง.2 วิธีการแก้ไขโค้ดที่มีความคลุมเครือ (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการความคลุมเครือ													เงื่อนไขในการเลือกวิธีการปรับปรุง
		Ambiguity in Comment	Ambiguity in Method	Ambiguity in One Thing	Ambiguity in Nesting Depth	Ambiguity in Arguments	Ambiguity in Structured Programming	Ambiguity in Class Organization	Ambiguity in Encapsulation	Ambiguity in Classes	Ambiguity in Cohesion	Ambiguity for Change	Ambiguity of Demeter	Ambiguity in Boundary	
16	Move Field			✓						✓	✓	✓			1. ปัญหาเกิดจำนวนของแอตทริบิวต์มีจำนวนมาก แอตทริบิวต์มีการทำงานที่คล้ายกับการทำงานของคลาสที่เกี่ยวข้อง เพื่อลดจำนวนควรย้ายแอตทริบิวต์นี้ไปยังคลาสที่มีการทำงานคล้ายกัน 2. ปัญหาเกิดจากแอตทริบิวต์ที่ถูกเรียกใช้งานบ่อยๆ ควรย้ายแอตทริบิวต์ให้อยู่ใกล้หรืออยู่ในขอบเขตเดียวกัน เพื่อให้เรียกใช้งานหรือแก้ไขปัญหาได้สะดวก 3. ปัญหาเกิดจากแอตทริบิวต์ที่อยู่ในตำแหน่งไม่เหมาะสม จึงไม่ค่อยถูกเรียกใช้งานและไม่มีความเกี่ยวข้องโดยตรงกับการทำงานในคลาส ควรย้ายแอตทริบิวต์ให้ไปอยู่ในส่วนที่ทำงานสอดคล้องกัน

ตารางที่ ง.2 วิธีการแก้ไขโค้ดที่มีความคลุมเครือ (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการความคลุมเครือ													เงื่อนไขในการเลือกวิธีการปรับปรุง
		Ambiguity in Comment	Ambiguity in Method	Ambiguity in One Thing	Ambiguity in Nesting Depth	Ambiguity in Arguments	Ambiguity in Structured Programming	Ambiguity in Class Organization	Ambiguity in Encapsulation	Ambiguity in Classes	Ambiguity in Cohesion	Ambiguity for Change	Ambiguity of Demeter	Ambiguity in Boundary	
17	Move Method			✓						✓	✓	✓	✓		1. ปัญหาเกิดจากการเรียกใช้งานเมท็อดไม่เหมาะสม เพราะการทำงานของเมท็อดไม่สอดคล้องกับคลาสที่เมท็อดนี้ถูกประกาศไว้ ทำให้ระหว่างการทำงานอาจเรียกใช้งานเมท็อดนี้เพื่อส่งข้อมูลไปกลับเกินความจำเป็นทั้งๆที่ไม่มีความเกี่ยวข้องกับการทำงานโดยตรงกับคลาสที่เป็นเจ้าของ ควรย้ายมาให้อยู่ในขอบเขตของคลาสที่ใช้งานจริง เพื่อลดการติดต่อสื่อสารที่ไม่จำเป็น 2. ปัญหาเกิดจำนวนของเมท็อดมีจำนวนมาก และมีบางเมท็อดมีการทำงานที่คล้ายคลึงกับคลาสที่เกี่ยวข้อง ควรย้ายเมท็อดนี้ไปยังคลาสที่มีการทำงานคล้ายคลึงกัน เพื่อลดจำนวนของเมท็อด

ตารางที่ ง.2 วิธีการแก้ไขโค้ดที่มีความคลุมเครือ (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการความคลุมเครือ													เงื่อนไขในการเลือกวิธีการปรับปรุง
		Ambiguity in Comment	Ambiguity in Method	Ambiguity in One Thing	Ambiguity in Nesting Depth	Ambiguity in Arguments	Ambiguity in Structured Programming	Ambiguity in Class Organization	Ambiguity in Encapsulation	Ambiguity in Classes	Ambiguity in Cohesion	Ambiguity for Change	Ambiguity of Demeter	Ambiguity in Boundary	
18	Parameterize method									✓					1. ปัญหาจากการเรียกใช้งานของเมท็อดบางตัวที่มีการทำงานเหมือนกันแต่แตกต่างที่ค่าในการคำนวณ แก้ไขโดยยุบเมท็อดและสร้างเมท็อดใหม่ที่มีพารามิเตอร์นำเข้าเป็นค่าที่ต้องการใช้งานแทน
19	Preserve Whole Object		✓		✓	✓									1. ปัญหาเกิดจากการกำหนดตัวแปรชั่วคราวหลายตัวเพื่อเก็บค่าในการคำนวณทำให้มีการทำงานซับซ้อนจึงควรแยกออกไปให้อยู่ภายในเมท็อดที่ทำงานแบบเดียวกัน 2. ปัญหาเกิดจากจำนวนของพารามิเตอร์รับเข้าทำงานภายในเมท็อดมีจำนวนมากและพารามิเตอร์ที่นำเข้าส่วนมากถูกเก็บเป็นค่าชั่วคราวหรือใช้งานเพียงเมท็อดนี้เท่านั้นควรยุบรวมส่วนนี้

ตารางที่ ง.2 วิธีการแก้ไขโค้ดที่มีความคลุมเครือ (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการความคลุมเครือ													เงื่อนไขในการเลือกวิธีการปรับปรุง
		Ambiguity in Comment	Ambiguity in Method	Ambiguity in One Thing	Ambiguity in Nesting Depth	Ambiguity in Arguments	Ambiguity in Structured Programming	Ambiguity in Class Organization	Ambiguity in Encapsulation	Ambiguity in Classes	Ambiguity in Cohesion	Ambiguity for Change	Ambiguity of Demeter	Ambiguity in Boundary	
19	Preserve Whole Object														3. ปัญหาเกิดการรับค่าไปเก็บไว้ในตัวแปรชั่วคราวหลายตัวก่อนจะส่งค่าไปคำนวณในฟังก์ชันถัดไป ทำให้การทำงานยุ่งยาก เพราะต้องส่งค่าไปหลายครั้ง จึงควรแยกออกเพื่อทำงานให้เบ็ดเสร็จภายในเมท็อดเดียวกัน
20	Remove control flag											✓			1. ปัญหาเกิดจากตัวแปรที่คอยเก็บสถานะการทำงาน การทำงานในแต่ละครั้งจำเป็นต้องมีการแก้ไขสถานะตัวแปรนี้เสมอ ทำให้ยากต่อการแก้ไขเปลี่ยนแปลง จึงควรลดหรือแยกสถานะออกไปเก็บไว้ในแต่ส่วน เพื่อความสะดวกในการแก้ไขปรับปรุง

ตารางที่ ง.2 วิธีการแก้ไขโค้ดที่มีความคลุมเครือ (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการความคลุมเครือ													เงื่อนไขในการเลือกวิธีการปรับปรุง
		Ambiguity in Comment	Ambiguity in Method	Ambiguity in One Thing	Ambiguity in Nesting Depth	Ambiguity in Arguments	Ambiguity in Structured Programming	Ambiguity in Class Organization	Ambiguity in Encapsulation	Ambiguity in Classes	Ambiguity in Cohesion	Ambiguity for Change	Ambiguity of Demeter	Ambiguity in Boundary	
21	Remove middle man											✓			1. ปัญหาเกิดจากต้องมีตัวกลางในการทำงาน ซึ่งไม่มีความจำเป็นที่ต้องเรียกใช้งานผ่านตัวกลางทุกครั้ง ควรแก้ไขด้วยการเรียกใช้งานผ่านตัวกลางแค่ครั้งแรกครั้งเดียว
22	Remove Parameter					✓				✓					1. ปัญหาเกิดจากจำนวนของพารามิเตอร์รับเข้าทำงานในเมท็อดมีจำนวนมาก โดยพารามิเตอร์บางตัวสามารถคำนวณหาค่าได้จากพารามิเตอร์ตัวอื่นที่รับเข้าหรือจากแอตทริบิวต์ในคลาส สามารถตัดออกและคำนวณค่าที่รับเข้ามาใช้งานในฟังก์ชัน

ตารางที่ ง.2 วิธีการแก้ไขโค้ดที่มีความคลุมเครือ (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการความคลุมเครือ													เงื่อนไขในการเลือกวิธีการปรับปรุง
		Ambiguity in Comment	Ambiguity in Method	Ambiguity in One Thing	Ambiguity in Nesting Depth	Ambiguity in Arguments	Ambiguity in Structured Programming	Ambiguity in Class Organization	Ambiguity in Encapsulation	Ambiguity in Classes	Ambiguity in Cohesion	Ambiguity for Change	Ambiguity of Demeter	Ambiguity in Boundary	
22	Remove Parameter														2. ปัญหาเกิดจากจำนวนของพารามิเตอร์รับเข้าทำงานในเมท็อดมีจำนวนมาก โดยมีพารามิเตอร์บางตัวที่ไม่ถูกใช้งานในเมท็อด ควรลบพารามิเตอร์นำเข้านี้ออก 3. ปัญหาเกิดจากจำนวนของพารามิเตอร์รับเข้าทำงานในเมท็อดมีจำนวนมาก ซึ่งมีพารามิเตอร์บางตัวไม่ถูกเรียกใช้งาน สามารถตัดพารามิเตอร์ออกจากการนำเข้าของเมท็อดนี้ได้
23	Replace exception with test						✓								1. ปัญหาเกิดจากวิธีการดักจับข้อผิดพลาดในการทำงานของโปรแกรม หากสามารถคาดการณ์ข้อผิดพลาดที่เกิดขึ้นได้ ควรแก้ไขด้วยการทดสอบค่าต่างๆ เพื่อป้องกันไม่ให้เกิดข้อผิดพลาด

ตารางที่ ง.2 วิธีการแก้ไขโค้ดที่มีความคลุมเครือ (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการความคลุมเครือ													เงื่อนไขในการเลือกวิธีการปรับปรุง
		Ambiguity in Comment	Ambiguity in Method	Ambiguity in One Thing	Ambiguity in Nesting Depth	Ambiguity in Arguments	Ambiguity in Structured Programming	Ambiguity in Class Organization	Ambiguity in Encapsulation	Ambiguity in Classes	Ambiguity in Cohesion	Ambiguity for Change	Ambiguity of Demeter	Ambiguity in Boundary	
23	Replace exception with test														ขึ้นในระหว่างการทำงานแทน เพื่อสามารถจัดการหรือวางแผนแก้ไขเมื่อพบข้อผิดพลาดเกิดขึ้น
24	Replace Method with Method Object		✓												1. ปัญหาเกิดจากการทำงานภายในฟังก์ชันที่เรียกใช้ข้อมูลหรือต้องการคำนวณหาค่าที่ซับซ้อน หรือมีปริมาณข้อมูลที่ต้องคำนวณมาก ควรจะตัดแบ่งส่วนนี้ออกเป็นเมท็อดใหม่ เพื่อส่งคืนผลลัพธ์เป็นชนิดของข้อมูล
25	Replace parameter with explicit method					✓									1. ปัญหาเกิดจากพารามิเตอร์ที่รับเข้าทำงานในเมท็อดมีจำนวนมาก ซึ่งพารามิเตอร์แต่ละตัวมีขอบเขตการทำงานที่อิสระ ควรแยกออกพารามิเตอร์นี้ออกมาเป็นเมท็อดเดียวเพื่อง่ายต่อการแก้ไข

ตารางที่ ง.2 วิธีการแก้ไขโค้ดที่มีความคลุมเครือ (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการความคลุมเครือ													เงื่อนไขในการเลือกวิธีการปรับปรุง
		Ambiguity in Comment	Ambiguity in Method	Ambiguity in One Thing	Ambiguity in Nesting Depth	Ambiguity in Arguments	Ambiguity in Structured Programming	Ambiguity in Class Organization	Ambiguity in Encapsulation	Ambiguity in Classes	Ambiguity in Cohesion	Ambiguity for Change	Ambiguity of Demeter	Ambiguity in Boundary	
26	Replace Parameter with Method					✓				✓					1. ปัญหาเกิดจากจำนวนของพารามิเตอร์รับเข้าทำงานในเมท็อดมีจำนวนมาก โดยมีพารามิเตอร์บางตัวไม่จำเป็นต้องนำเข้า เพราะสามารถคำนวณหาค่าจากเมท็อดอื่นได้และสามารถตัดพารามิเตอร์ออก เพื่อคำนวณค่าใหม่ภายในในฟังก์ชัน
27	Replace Temp with Query		✓												1. ปัญหาเกิดจากการนำตัวแปรชั่วคราวไปใช้หลายๆ ที่ภายในฟังก์ชัน ควรแปลงตัวแปรชั่วคราวนี้ให้อยู่ในรูปของเมท็อดใหม่เพื่อเรียกใช้งานได้อย่างถาวร

ตารางที่ ง.2 วิธีการแก้ไขโค้ดที่มีความคลุมเครือ (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการความคลุมเครือ													เงื่อนไขในการเลือกวิธีการปรับปรุง
		Ambiguity in Comment	Ambiguity in Method	Ambiguity in One Thing	Ambiguity in Nesting Depth	Ambiguity in Arguments	Ambiguity in Structured Programming	Ambiguity in Class Organization	Ambiguity in Encapsulation	Ambiguity in Classes	Ambiguity in Cohesion	Ambiguity for Change	Ambiguity of Demeter	Ambiguity in Boundary	
28	Replace type code with state/strategy						✓								1. ปัญหาเกิดจากข้อผิดพลาดของโปรแกรมทำงานไม่เป็นไปตามโครงสร้างที่กำหนด เนื่องจากมีโค้ดบางประเภทมีการอ้างถึงประเภทข้อมูลที่ต้องการผิดพลาด ควรแก้ไขโดยใช้วิธีการเป็นข้อมูลสถานะหรือออกแบบวิธีการที่มีระเบียบแบบแผนในการทำงาน
29	Seal Classes							✓							1. เป็นการป้องกันไม่ให้เกิดการแก้ไขคลาสหรือสืบทอดคลาส เนื่องจากการทำงานของคลาสอาจขึ้นอยู่กับโครงสร้างของข้อมูลที่ออกแบบเฉพาะด้าน เพราะในกรณีที่มีส่วนเชื่อมต่อกับผู้ใช้งานไม่สามารถแก้ไขได้

ตารางที่ ง.2 วิธีการแก้ไขโค้ดที่มีความคลุมเครือ (ต่อ)

ลำดับที่	เทคนิครีแฟคทอริ่งที่ใช้ปรับปรุง	รายการความคลุมเครือ													เงื่อนไขในการเลือกวิธีการปรับปรุง
		Ambiguity in Comment	Ambiguity in Method	Ambiguity in One Thing	Ambiguity in Nesting Depth	Ambiguity in Arguments	Ambiguity in Structured Programming	Ambiguity in Class Organization	Ambiguity in Encapsulation	Ambiguity in Classes	Ambiguity in Cohesion	Ambiguity for Change	Ambiguity of Demeter	Ambiguity in Boundary	
30	Substitute algorithm		✓	✓	✓						✓				1. ปัญหาเกิดจากจำนวนเงื่อนไขในการทำงาน หากมีเงื่อนไขในแต่ละขั้นตอนทำงานอยู่เสมอและแต่ละเงื่อนไขถูกแบ่งตามชนิดหรือพฤติกรรมการทำงาน ก็สามารถที่จะแก้ไขให้อยู่รูปของการใช้งานแบบทดแทนกันได้

ตารางที่ ง.3 รายละเอียดขั้นตอนวิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ

ลำดับที่	วิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ	ขั้นตอนที่	รายละเอียดขั้นตอน
1	Add Comment	1	ตรวจหาคลาสที่ไม่มีคำอธิบายโปรแกรมหรือมีคำอธิบายน้อยเกินไป
		2	เพิ่มเติมคำอธิบายโปรแกรมลงในส่วนของเมท็อด โดยพิจารณาจากขนาดหรือความซับซ้อนของเมท็อด
		3	แปลชุดคำสั่งเพื่อทดสอบว่าโปรแกรมใช้งานได้หรือไม่
2	Add Parameter	1	วิเคราะห์แอตทริบิวต์ที่ถูกเรียกใช้งานในเมท็อดที่กำลังพิจารณา
		2	ทำการกำหนดค่าพารามิเตอร์ให้แก่เมท็อดโดย ให้มีชนิดเดียวกันกับแอตทริบิวต์ที่เรียกใช้งาน
		3	ทำการแทนค่าแอตทริบิวต์ที่เรียกใช้งานภายในเมท็อดด้วยพารามิเตอร์ที่ถูกสร้างขึ้น
		4	ตรวจสอบแอตทริบิวต์ที่ถูกแทนค่าในเมท็อด หากไม่มีการอ้างถึงหรือเรียกใช้งานให้ลบทิ้ง
		5	ทดสอบเพื่อยืนยันความถูกต้องหลังจากแก้ไขเรียบร้อย
3	Change Bidirectional Association to Unidirectional Association	1	วิเคราะห์การ สื่อสาร ระหว่างคลาสเพื่อหาคลาสหลักและ คลาส รอง ที่ใช้ติดต่อเพื่อแก้ไขให้สื่อสารไปในทิศทางเดียวกัน
		2	ติดตั้ง observer ไปยังแอตทริบิวต์ของคลาสหลักแทนการติดต่อสื่อสารเดิม

ตารางที่ ง.3 รายละเอียดขั้นตอนวิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ (ต่อ)

ลำดับที่	วิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ	ขั้นตอนที่	รายละเอียดขั้นตอน
3	Change Bidirectional Association to Unidirectional Association	3	ลบแอตทริบิวต์ในส่วนคลาสที่ทำการติดต่อสื่อสาร
		4	ตรวจสอบความถูกต้องในการทำงาน หลังจากที่แก้ไขเสร็จสิ้น
4	Collapse Hierarchy	1	ตรวจหาคลาสแม่และคลาสลูกที่มีคุณสมบัติการทำงานที่น้อยมากหรือ อาจ ไม่มีความแตกต่างจากคลาสแม่ที่ถูกสืบทอด
		2	ยุบกลับไปรวมกับคลาสแม่ที่เรียกใช้งาน
		3	แก้ไขการเรียกใช้งานคลาสลูกไปเป็นการเรียกใช้งานคลาสแม่แทน
		4	ตรวจสอบความถูกต้องในการทำงาน หลังจากที่แก้ไขเสร็จสิ้น
5	Decompose Conditional	1	วิเคราะห์นิพจน์ที่สร้างซ้อนทับกันในเมท็อด
		2	แก้ไขได้โดยแปลงให้อยู่ในรูปของเมท็อดใหม่ แล้วแยกย่อยออกไปเป็นเมท็อดใหม่
		3	ตรวจสอบความถูกต้อง หลังจากแยกออกจากการเรียกใช้งานให้เป็นเมท็อดใหม่
6	Duplicate observed data	1	ตรวจสอบหาจำนวนของแอตทริบิวต์ที่มีจำนวนมากในคลาส
		2	วิเคราะห์เมื่อมีการเปลี่ยนแปลงหลังจาก แก้ไขค่าแอตทริบิวต์และผลกระทบไปยังส่วนต่างๆ ของโปรแกรม

ตารางที่ ง.3 รายละเอียดขั้นตอนวิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ (ต่อ)

ลำดับที่	วิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ	ขั้นตอนที่	รายละเอียดขั้นตอน
6	Duplicate observed data	3	ติดตั้ง observer ให้แก่แอตทริบิวต์และส่งค่าไปยังแอตทริบิวต์ต่างๆที่มีส่วนเกี่ยวข้อง
		4	ตัดโค้ดที่มีส่วนซ้ำซ้อนออกเป็นคลาสใหม่หรือรวมเป็นฟังก์ชัน
		5	ตรวจสอบความถูกต้อง หลังจากแก้ไขการเรียกใช้งาน
7	Encapsulate Field	1	ตรวจสอบหาการประกาศแอตทริบิวต์ที่ถูกใช้ เพื่อเก็บค่าชั่วคราว
		2	แก้ไขให้มีการปกปิดข้อมูลหรือการกำหนดลำดับชั้นในการเข้าถึงข้อมูล
		3	แก้ไขส่วนต่างๆที่ทำการเรียกใช้งานแอตทริบิวต์นี้
		4	ตรวจสอบความถูกต้องจากการเรียกใช้งาน หลังจากแก้ไขเรียบร้อย
8	Extract Class	1	ตรวจสอบหาคลาสที่มีจำนวนเมท็อดหรือแอตทริบิวต์มาก
		2	แบ่งคลาสออกเป็นคลาสย่อยตามหน้าที่รับผิดชอบหรือตามขอบเขตการทำงาน
		3	แก้ไขการเรียกใช้งานหลังจากแบ่งคลาสออกเป็นคลาสย่อยๆ
		4	ตรวจสอบความถูกต้องของผลลัพธ์ต่างๆ หลังจากแก้ไขเสร็จสิ้น
9	Extract Interface	1	ตรวจสอบหาส่วนต่อประสานที่เชื่อมกันมีจำนวนมาก
		2	แบ่งส่วนต่อประสานที่ถูกเชื่อมไว้ออกเป็นส่วนย่อยๆ ตามพฤติกรรมของการทำงาน

ตารางที่ ง.3 รายละเอียดขั้นตอนวิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ (ต่อ)

ลำดับที่	วิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ	ขั้นตอนที่	รายละเอียดขั้นตอน
9	Extract Interface	3	นำส่วนต่อประสานที่ถูกแบ่งออกให้ประกาศแทนส่วนต่อประสานเก่าในคลาสต่างๆ
		4	แปลชุดคำสั่งเพื่อทดสอบว่าโปรแกรมใช้งานได้
10	Extract Method	1	ตรวจสอบหาคลาสที่มีจำนวนบรรทัดในการทำงานมากหรือมีลำดับการ ทำงานที่ซับซ้อนเกินไป
		2	วิเคราะห์เพื่อแบ่งเมท็อดออกเป็นส่วนย่อยๆ ตามลักษณะการทำงาน
		3	แก้ไขการเรียกใช้งานภายในเมท็อดที่ได้รับการแก้ไข
		4	ทดสอบหลังจากที่แก้ไขการทำงานเสร็จสิ้น
11	Extract Subclass	1	วิเคราะห์หาคลาสที่มีจำนวนเมท็อดหรือแอตทริบิวต์ที่มีจำนวนมาก
		2	แบ่งคลาสออกเป็นคลาสลูก เพื่อสืบทอดการทำงานจากคลาสแม่
		3	แก้ไขการเรียกใช้งานในส่วนๆ ต่างของโปรแกรมจากคลาสแม่มาเป็นคลาสลูกแทน
		4	ตรวจสอบความถูกต้องในการทำงาน หลังจากที่แก้ไขเสร็จสิ้น
12	Hide Delegate	1	วิเคราะห์การเรียกใช้งานข้อมูลจากหลายคลาส เช่นจากคลาสหนึ่งไปอีกคลาสหนึ่งเป็นลำดับขั้น

ตารางที่ ง.3 รายละเอียดขั้นตอนวิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ (ต่อ)

ลำดับที่	วิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ	ขั้นตอนที่	รายละเอียดขั้นตอน
12	Hide Delegate	3	สร้างเมท็อดใหม่เพื่อเรียกใช้ข้อมูลที่ต้องการให้อยู่ภายในคลาสเดียวกันเพื่อลดลำดับการใช้งาน
		4	ตรวจสอบการเรียกใช้งานเพื่อความถูกต้อง หลังจากที่แก้ไขให้เรียกใช้งานในเมท็อดเดียวกัน
13	Inline Class	1	ตรวจหาคลาสที่มีการทำงานน้อยและสามารถยุบรวมกับคลาสอื่นได้ เนื่องจากเป็นคลาสลูกหรือมีการทำงานคล้ายคลึงกัน
		2	ยุบรวมกับคลาสที่ทำงานคล้ายกัน
		3	ตรวจสอบการเรียกใช้งานเพื่อความถูกต้อง หลังจากที่นำคลาสมายุบรวมกัน
14	Inline Method	1	ตรวจหาการทำงานของเมท็อดที่อยู่ในตำแหน่งที่ไม่เหมาะสม ที่ถูกเรียกบ่อยครั้ง ทั้งที่ไม่มีหน้าที่เกี่ยวข้องโดยตรง
		2	วิเคราะห์ส่วนที่ต้องการใช้งานอยู่ในเมท็อดที่ถูกเรียกใช้งาน แล้วจึงเพิ่มเติมส่วนนี้เข้ามาเป็นเมท็อดใหม่ที่อยู่ภายใน ทำให้ไม่ต้องเรียกใช้งานจากภายนอก
		3	ตรวจสอบเมท็อดหลังจากแก้ไขเสร็จแล้วเพื่อความถูกต้อง

ตารางที่ ง.3 รายละเอียดขั้นตอนวิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ (ต่อ)

ลำดับที่	วิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ	ขั้นตอนที่	รายละเอียดขั้นตอน
15	Introduce assertion	1	ตรวจหาส่วนที่น่าสงสัยที่ไม่มีการป้องกันหรือตรวจสอบข้อผิดพลาด
		2	เพิ่มขั้นตอนการตรวจสอบค่าก่อนที่จะเข้าทำงานในส่วนที่สงสัย
		3	ทำการทดสอบหลังจากเพิ่มการตรวจสอบเสร็จสิ้น
16	Introduce local extension	1	ตรวจหาการประกาศแอตทริบิวต์ในการใช้งานคลาสที่อยู่ภายนอกหรือต้องเรียกผ่านคลาสอื่นๆ ทำให้ลำดับชั้นหรือความลึกของการทำงานมีมาก
		2	แก้ไขโดยนำค่ามาเก็บไว้ภายในคลาสเพื่อสะดวกในการเรียกใช้งาน
		3	ลบแอตทริบิวต์ที่ไม่ใช้งานออก
		4	ทดสอบเรียกใช้งานแอตทริบิวต์และการทำงานหลังจากแก้ไขเสร็จสิ้น
17	Introduce Null Object	1	ตรวจหาการประกาศแอตทริบิวต์ในการใช้งานโดยที่ไม่มีค่าเริ่มต้นให้กับแอตทริบิวต์
		2	หากแอตทริบิวต์มีความเป็นวัตถุ ให้กำหนดค่าเริ่มต้นให้แก่แอตทริบิวต์
		3	ทดสอบเรียกใช้งานแอตทริบิวต์และการทำงาน หลังจากทำการแก้ไขเสร็จสิ้น
18	Introduce Parameter Object	1	วิเคราะห์พารามิเตอร์ที่ถูกนำเข้า หากพบกรณีที่พารามิเตอร์มีการขึ้นต่อการทำงานของพารามิเตอร์อีกตัวสามารถยุบรวมในรูปของวัตถุได้

ตารางที่ ง.3 รายละเอียดขั้นตอนวิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ (ต่อ)

ลำดับที่	วิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ	ขั้นตอนที่	รายละเอียดขั้นตอน
18	Introduce Parameter Object	2	ทำการยุบรวมให้อยู่ในรูปของวัตถุแล้วจึงลบพารามิเตอร์ที่ไม่ได้ใช้งานออก
		3	ทดสอบเงื่อนไขต่างๆ และการเรียกใช้งานอื่นๆ เพื่อตรวจสอบความถูกต้อง
19	Move Field	1	ค้นหากลุ่มการทำงานของแอตทริบิวต์ที่อยู่ในตำแหน่งไม่เหมาะสม
		2	ย้ายแอตทริบิวต์ไปยังที่ตำแหน่งเหมาะสมในคลาสที่มีความเกี่ยวข้องกัน
		3	ตรวจแก้ไขการเรียกใช้งานแอตทริบิวต์ที่ถูกย้ายไป
		4	ทดสอบเรียกใช้งานแอตทริบิวต์และตรวจสอบการทำงานหลังจากที่แก้ไขเสร็จสิ้น
20	Move Method	1	ค้นหากลุ่มของการทำงานของเมท็อดที่อยู่ในตำแหน่งไม่เหมาะสม
		2	ย้ายเมท็อดไปยังตำแหน่งที่เหมาะสมในคลาสที่มีความเกี่ยวข้องกัน
		3	ตรวจแก้ไขการเรียกใช้งานเมท็อดที่ถูกย้ายไป
		4	ทดสอบเรียกใช้งานเมท็อดและตรวจสอบการทำงานหลังจากที่แก้ไขเสร็จสิ้น
21	Parameterize method	1	ค้นหากลุ่มของการใช้งานของเมท็อดที่มีการทำงานเหมือนกัน
		2	ยุบเมท็อดรวมกันเพื่อสร้างเมท็อดใหม่ที่มีพารามิเตอร์นำเข้าเป็นค่าที่ต้องการใช้ในการทำงาน
		3	แก้ไขคลาสหรือเมท็อดที่มีการเรียกใช้งานหรือส่วนที่เกี่ยวข้อง

ตารางที่ ง.3 รายละเอียดขั้นตอนวิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ (ต่อ)

ลำดับที่	วิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ	ขั้นตอนที่	รายละเอียดขั้นตอน
21	Parameterize method	4	ทดสอบการเรียกใช้งานเมท็อดและการทำงานหลังจากที่แก้ไขเสร็จสิ้น
22	Preserve Whole Object	1	วิเคราะห์หากลุ่มตัวแปรชั่วคราวสำหรับรับส่งค่าเพื่อคำนวณในฟังก์ชันถัดไป
		2	ประกาศเมท็อดใหม่แล้วนำกลุ่มการทำงานที่ตรวจพบ ย้ายมาไว้ในเมท็อดที่ประกาศขึ้น
		3	ทดสอบการทำงานหลังจากที่แก้ไขเสร็จสิ้น
23	Remove control flag	1	ค้นหาตัวแปรที่ ทำหน้าที่ เก็บสถานะการทำงาน โดยการทำงานแต่ละครั้งจำเป็นต้องมีการแก้ไขสถานะตัวแปรนี้เสมอ
		2	แยกตัวแปรนี้มาแปลงเป็นสถานะภายในแต่ละวัตถุ เพื่อแทนวิธีการทำงานแบบเดิม
		3	ลบตัวแปรที่เก็บค่าสถานะการทำงานออกจากโปรแกรม
		4	ทดสอบการทำงานหลังจากที่แก้ไขเสร็จสิ้น
24	Remove middle man	1	ค้นหาลักษณะการทำงานที่มีตัวกลางในการทำงาน ซึ่งไม่มีความจำเป็นที่ต้องเรียกใช้งานผ่านตัวกลางในการทำงาน

ตารางที่ ง.3 รายละเอียดขั้นตอนวิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ (ต่อ)

ลำดับที่	วิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ	ขั้นตอนที่	รายละเอียดขั้นตอน
24	Remove middle man	2	กำหนดเมท็อดใหม่เพื่อส่งคืนค่าวัตถุที่ต้องการเรียกใช้งาน เพื่อให้เกิดการเรียกใช้งานผ่านตัวกลางแค่ครั้งแรกเพียงครั้งเดียว
		3	ทดสอบการทำงานหลังจากที่แก้ไขเสร็จสิ้น
25	Remove Parameter	1	ตรวจหาพารามิเตอร์บางตัวที่ไม่ถูกใช้งานในเมท็อด
		2	ลบพารามิเตอร์นำเข้านี้ออก
		3	ทดสอบการทำงานหลังจากที่ลบพารามิเตอร์นี้ออกแล้ว
26	Rename Method	1	ตรวจหาเมท็อดที่ประกาศไว้และมีชื่อซ้ำซ้อน
		2	แก้ไขโดยกำหนดชื่อที่ใช้อ้างถึงใหม่เพื่อให้เรียกใช้งานได้สะดวก
		3	ทดสอบการทำงานหลังจากแก้ไขการเรียกงานเมท็อด
27	Replace Delegation with Inheritance	1	ตรวจสอบหาเมท็อดที่สร้างแบบชั่วคราวในการทำงาน
		2	นำเมท็อดที่ตรวจพบมาประกาศไว้ที่คลาสแม่ เพื่อทำการสืบทอด
		3	ลบเมท็อดเดิมออก หลังจากทำการสืบทอดเสร็จ และแก้ไขทุกส่วนที่มีการเรียกใช้ แบบเดิมมาเรียกใช้แบบสืบทอด
		4	ทดสอบการทำงานหลังจากแก้ไขเพื่อเรียกเมท็อดจากการสืบทอด

ตารางที่ ง.3 รายละเอียดขั้นตอนวิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ (ต่อ)

ลำดับที่	วิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ	ขั้นตอนที่	รายละเอียดขั้นตอน
28	Replace exception with test	1	ตรวจหาการดักจับข้อผิดพลาดในการทำงานของโปรแกรม
		2	หากสามารถคาดการณ์ข้อผิดพลาดต่างๆที่เกิดขึ้นได้ ควรสร้างเงื่อนไขเพื่อทดสอบค่าก่อนทำงานในลำดับต่อไป
		3	ในกรณีที่สร้างการตรวจสอบได้ครบทุกๆ กรณี ในลบการดักจับข้อผิดพลาดออก
		4	ทดสอบด้วยการส่งค่าผิดพลาดต่างๆ เพื่อให้ครอบคลุมทุกกรณี
29	Replace Method with Method Object	1	วิเคราะห์การทำงานของเมท็อด หากพบส่วนที่มีการเรียกใช้ข้อมูลหรือคำนวณหาค่าที่มีความซับซ้อน
		2	ตัดแบ่งเมท็อด ออกเป็นเมท็อด ใหม่ย่อยๆ ในแต่ละส่วน ซึ่งทำงานส่งข้อมูลที่คำนว ณเสร็จเรียบร้อยเพื่อทำงานต่อ
		3	ทดสอบการทำงานหลังจากที่แยกเมท็อดออกเป็นส่วนย่อยๆ เพื่อความถูกต้องของการทำงาน
30	Replace parameter with explicit method	1	วิเคราะห์พารามิเตอร์ที่รับเข้าทำงานในเมท็อด หากตรวจพบพารามิเตอร์แต่ละตัวมีขอบเขตการทำงานที่อิสระไม่เกี่ยวข้องระหว่างกัน
		2	แยกพารามิเตอร์นี้ออกมาเป็นเมท็อดใหม่ เพื่อง่ายต่อการแก้ไขและเรียกใช้งาน

ตารางที่ ง.3 รายละเอียดขั้นตอนวิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ (ต่อ)

ลำดับที่	วิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ	ขั้นตอนที่	รายละเอียดขั้นตอน
30	Replace parameter with explicit method	3	ลบพารามิเตอร์ออก หลังจากแก้ไขให้เรียกใช้งานจากเมท็อด
		4	ทดสอบการทำงานหลังจากที่ลบพารามิเตอร์ออกเพื่อตรวจความถูกต้องของการทำงาน
31	Replace Parameter with Method	1	วิเคราะห์พารามิเตอร์ที่รับเข้าทำงานในเมท็อด หากตรวจพบพารามิเตอร์ที่ไม่จำเป็นเนื่องจากสามารถคำนวณหาค่าจากเมท็อดอื่นได้
		2	ลบพารามิเตอร์ออก และแก้ไขให้เรียกใช้งานจากเมท็อดอื่นแทน
		3	ทดสอบการทำงานหลังจากที่ลบพารามิเตอร์ออกเพื่อความถูกต้องของการทำงาน
32	Replace Temp with Query	1	วิเคราะห์ตัวแปรชั่วคราวที่ถูกใช้งานในหลายๆ ที่ภายในเมท็อดของแต่ละคลาส
		2	สร้างเมท็อดใหม่เพื่อเรียกใช้งานตัวแปรชั่วคราวและเกิดการเรียกใช้อย่างถาวร
		3	ลบตัวแปรชั่วคราวที่ถูกแทนที่ด้วยเมท็อดออก
		4	ทดสอบการทำงานหลังจากที่ลบตัวแปรออกเพื่อความถูกต้องของการทำงาน
33	Replace type code with state/strategy	1	ตรวจหาค่าตัวแปรที่กำหนดชนิดของโค้ดซึ่งแบ่งตามลักษณะการทำงานของโปรแกรม
		2	ในกรณีที่การทำงานของโปรแกรมที่ ถูก ตรวจสอบเป็นลักษณะของสถานะของข้อมูล ควรเลือกใช้วิธีแก้ไขแบบ state

ตารางที่ ง.3 รายละเอียดขั้นตอนวิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ (ต่อ)

ลำดับที่	วิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ	ขั้นตอนที่	รายละเอียดขั้นตอน
33	Replace type code with state/strategy	3	แต่ในกรณีที่การทำงานมีความสัมพันธ์กับพฤติกรรมของการทำงาน ควรเลือกใช้วิธีแก้ไขแบบ strategy
		4	ลบชนิดค่าคงที่ต่างๆที่กำหนดลักษณะการทำงาน และนำวิธีการแก้ไขโดย state/strategy เข้ามาทดแทนวิธีเดิม
		5	ทดสอบการทำงานหลังจากที่ลบชนิดข้อมูลออกเพื่อความถูกต้องของการทำงาน
34	Seal Classes	1	ตรวจสอบหาคลาสที่เป็นคลาสลำดับท้ายสุดใน รูปภาพต้นไม้เพื่อตัดสินใจ ทำการกำหนดคุณสมบัติปิด
		2	ทำการเติมคุณลักษณะปิดผนึก เพื่อไม่ให้คลาสนั้นถูกนำไปทำการสืบทอดหรือนำไปใช้งานต่อ
		3	แปลชุดคำสั่งเพื่อทดสอบว่าโปรแกรมมาใช้งานได้
35	Substitute algorithm	1	วิเคราะห์เงื่อนไขที่อยู่ภายในเมท็อด ซึ่งถูกแบ่งแยกตามพฤติกรรมและลักษณะการทำงาน
		2	สร้างคลาสใหม่ตามลักษณะการทำงานหรือประเภทการทำงานที่ได้วิเคราะห์ไว้
		3	ย้ายการทำงานที่อยู่ภายในแต่ละเงื่อนไขไปสู่คลาสใหม่ที่สร้างขึ้น

ตารางที่ ง.3 รายละเอียดขั้นตอนวิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ (ต่อ)

ลำดับที่	วิธีการแก้ไขร่องรอยไม่ดีและความคลุมเครือ	ขั้นตอนที่	รายละเอียดขั้นตอน
35	Substitute algorithm	4	ปรับแต่งการทำงานของคลาสหลังจากย้ายพฤติกรรมต่างๆ ไปไว้ในคลาสใหม่แล้ว โดยลบเงื่อนไขต่างๆออกและเปลี่ยนเป็นการสร้าง instance ของแต่ละประเภทแทน
		5	ทดสอบการทำงานของแต่ละเงื่อนไข หลังจากที่ลบเงื่อนไขเพื่อความถูกต้องของการทำงาน

ภาคผนวก จ
ตัวอย่างวิธีการจำแนกโค้ดตัวอย่างด้วยมาตรวัดซอฟต์แวร์และฟัซซีโลจิก พร้อมแนวทางการปรับปรุงร่องรอยไม่ดีและโค้ดที่มีความคลุมเครือ

ภาคผนวกนี้แสดง ตัวอย่าง วิธีการจำแนกโค้ดและวิธี การปรับปรุง ร่องรอยไม่ดีและ โค้ดที่มีความคลุมเครือเพื่อให้มีคุณสมบัติเป็นซับเซตของคลีนโค้ดตามวิธีการที่ได้ออกแบบไว้ ซึ่งประกอบไปด้วยตารางผลลัพธ์จากการวัดค่าตามมาตรวัดที่ออกแบบ ตารางผลลัพธ์จากการแปลความ และตารางแสดงวิธีปรับปรุงร่องรอยไม่ดีหรือความคลุมเครือที่ตรวจพบจนกระทั่งมีผลลัพธ์ซับเซตของคลีนโค้ด โดยมีตัวอย่างทั้งสิ้น 9 ตัวอย่างแบ่งเป็น วิธีการจำแนกและปรับปรุงประเภทร่องรอยไม่ดี 3 ตัวอย่าง ประเภทโค้ดที่มีความคลุมเครือ 3 ตัวอย่าง และวิธีการจำแนกคลีนโค้ด 3 ตัวอย่าง ดังต่อไปนี้

ตารางที่ จ.1 ผลลัพธ์ที่วัดได้เป็นร่องรอยไม่ดี Hit the Keys

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ	Hit_the_keys																												
จำนวนคลาสที่ใช้ทดสอบ	3				จำนวนเมท็อดที่ใช้ทดสอบ						8							จำนวนแอตทริบิวต์ที่ใช้ทดสอบ						16					
	มาตรวัดที่ใช้ในการทดสอบ																												
ชื่อคลาสที่ใช้ทดสอบ	ACML	CC	ECML	LCC	LLND	NILI	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	RSS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-QMF	SEC-IM	ข้อจำกัด
1) Form1																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
# Constructor	1	1	4	1	0	18	5	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- Form1_KeyDown	1	5	12	6	2	173	23	1	2	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- InitializeComponent	1	1	55	1	0	396	71	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M1
- timer1_Tick	1	2	6	2	1	41	5	1	2	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Dispose	0	3	1	4	1	23	3	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
- components_3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- listBox1_3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- timer1_3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- difficultyProgressBar_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- correctLabel_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- missedLabel_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- accuracyLabel_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- random_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- totalLabel_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- stats_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- statusStrip1_1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- toolStripStatusLabel1_1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้											0	30	7	1	1	1	12	0	0	5	32								C1
2) Program																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
- Main	0	1	4	1	0	10	3	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้											0	3	1	1	0	0	0	0	0	1	50								
3) Stats																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
# Constructor	1	1	1	1	0	16	4	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Update	1	2	0	3	1	74	11	1	8	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
+ Correct	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Accuracy	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Total	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Missed	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้											4	0	1	1	0	0	4	0	0	2	0								
รวมทั้งสิ้น																						0	0	2	1	0	0		

ตารางที่ จ.2 ผลลัพธ์ที่แสดงค่าการแปลความ Hit the Keys

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			Hit_the_keys		
	ชุดรายการ			Defuzzification Output	สรุปค่าแปลความ
เซตที่	คลาส	เมท็อด	แอตทริบิวต์		
1	Form1	Constructor	components 3	0.297	Clean
2		Form1_KeyDown	listBox1 3	0.317	Clean
3		InitializeComponent	timer1 3	0.334	Ambiguous
4		timer1_Tick	difficultyProgressBar 2	0.297	Clean
5		Dispose	correctLabel 2	0.321	Clean
6			missedLabel 2		
7			accuracyLabel 2		
8			random 2		
9			totalLabel 2		
10			stats 2		
11			statusStrip1 1		
12			toolStripStatusLabel1 1		
13	Program	Main		0.263	Clean
14	Stats	Constructor	Correct	0.799	Bad
15		Update	Accuracy	0.799	Bad
16			Total		
17			Missed		

ตารางที่ จ.3 แนวทางการปรับปรุงโค้ดกลุ่มโค้ดร่องรอยไม่ดี Hit_the_keys ครั้งที่ 1

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			Hit_the_keys		
ลำดับที่	**คลาส**	**เมท็อด**	**แอตทริบิวต์**	**ผลลัพธ์การแปลความ**	**สรุปค่าแปลความ**
1	Stats	Constructor	Correct	0.799	Bad
2		Update	Accuracy	0.799	Bad
3			Total		
4			Missed		
ประเภทร่องรอยไม่ดีที่ต้องการแก้ไข			Long Parameter List และ Temporary Field		
วิธีการรีแฟคทอริ่งสำหรับการแก้ไขร่องรอยไม่ดี					
1. Composing Method วิธีย่อยที่ใช้ Introduce Parameter Object 2. Making Method Calls Simpler วิธีย่อยที่ใช้ Preserve Whole Object 3. Making Method Calls Simpler วิธีย่อยที่ใช้ Replace Parameter with Method 4. Moving Features between Objects วิธีย่อยที่ใช้ Extract Class 5. Simplifying Conditional Expression วิธีย่อยที่ใช้ Introduce Null Object					
วิธีที่เลือกใช้ในการปรับปรุงร่องรอยไม่ดี			Making Method Calls Simpler วิธีย่อยที่ใช้ Replace Parameter with Method		

ตารางที่ จ.3 แนวทางการปรับปรุงโค้ดกลุ่มโค้ดร่องรอยไม่ดี Hit_the_keys ครั้งที่ 1 (ต่อ)

เหตุผลที่เลือกใช้
ตัวอย่างมีร่องรอยไม่ดีอยู่สองชนิด ดังนั้นควรจัดการ Long Parameter List ตามลำดับของผลกระทบที่กำหนดไว้ตามวิธีปฏิบัติที่ออกแบบไว้ ในส่วนนี้ค่อนข้างเห็นสาเหตุของปัญหาได้ง่าย โดยสังเกตได้จากจำนวนของพารามิเตอร์ ที่รับเข้าทำงานในเมท็อดมีจำนวนมาก ซึ่งพารามิเตอร์บางตัวไม่จำเป็นต้องนำเข้า ดังนั้นแก้ไขด้วยการลดจำนวนของพารามิเตอร์ลงแล้ว ให้เรียกใช้ค่าจากเมท็อดแทน ด้วยวิธีการ Replace Parameter with Method เพื่อสร้างเมท็อดใหม่ในการเรียกใช้งานภายในเมท็อด สามารถลดจำนวนของพารามิเตอร์ที่รับเข้ามาได้แล้วจึงทำการทดสอบก่อนจะดำเนินการในขั้นตอนต่อไป

ตารางที่ จ.4 ผลลัพธ์หลังจากการปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี Hit the Keys ครั้งที่ 1

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ	Hit_the_keys																												
จำนวนคลาสที่ใช้ทดสอบ	3					จำนวนเมท็อดที่ใช้ทดสอบ						9						จำนวนแอตทริบิวต์ที่ใช้ทดสอบ						16					
	มาตรวัดที่ใช้ในการทดสอบ																												
ชื่อคลาสที่ใช้ทดสอบ	AOML	CC	EOML	ILCC	ILND	NILI	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-OMF	SEC-IM	ข้อจำกัด
1) Form1																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
# Constructor	1	1	4	1	0	18	5	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- Form1_KeyDown	1	5	12	6	2	145	16	1	2	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- InitializeComponent	1	1	55	1	0	396	71	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M1
- timer1_Tick	1	2	6	2	1	41	5	1	2	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Dispose	0	3	1	4	1	23	3	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
- components_3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- listBox1_3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- timer1_3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- difficultyProgressBar_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- correctLabel_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- missedLabel_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- accuracyLabel_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- random_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- totalLabel_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- stats_2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- statusStrip1_1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- toolStripStatusLabel1_1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	30	7	1	1	1	12	0	0	5	32							C1
2) Program																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
- Main	0	1	4	1	0	10	3	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	3	1	1	0	0	0	0	0	1	50							
3) Stats																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
# Constructor	1	1	1	1	0	16	4	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Update	1	2	0	3	1	80	11	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_Current	0	1	0	1	0	8	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
+ Correct	-	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Accuracy	-	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Total	-	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Missed	-	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												4	0	1	1	0	1	5	0	0	3	0							
รวมทั้งสิ้น																							0	0	2	1	0	0	

ตารางที่ จ.5 ผลลัพธ์แสดงค่าการแปลความหลังจากการปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี Hit the Keys ครั้งที่ 1

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			Hit_the_keys		
	ชุดรายการ			Defuzzification Output	สรุปค่าแปลความ
เซตที่	คลาส	เมท็อด	แอตทริบิวต์		
1	Form1	Constructor	components 3	0.303	Clean
2		Form1_KeyDown	listBox1 3	0.324	Clean
3		InitializeComponent	timer1 3	0.339	Ambigous
4		timer1_Tick	difficultyProgressBar 2	0.303	Clean
5		Dispose	correctLabel 2	0.327	Clean
6			missedLabel 2		
7			accuracyLabel 2		
8			random 2		
9			totalLabel 2		
10			stats 2		
11			statusStrip1 1		
12			toolStripStatusLabel1 1		
13	Program	Main		0.263	Clean
14	Stats	Constructor	Correct	0.799	Bad
15		Update	Accuracy	0.799	Bad
16		get_Current	Total	0.799	Bad
17			Missed	0.799	Bad

ตารางที่ จ.6 แนวทางการปรับปรุงโค้ดกลุ่มโค้ดร่องรอยไม่ดี Hit_the_keys ครั้งที่ 2

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			Hit_the_keys		
ลำดับที่	**คลาส**	**เมท็อด**	**แอตทริบิวต์**	**ผลลัพธ์การแปลความ**	**สรุปค่าแปลความ**
1	Stats	Constructor	Correct	0.799	Bad
2		Update	Accuracy	0.799	Bad
3		get_Current	Total	0.799	Bad
4			Missed	0.799	Bad
ประเภทร่องรอยไม่ดีที่ต้องการแก้ไข			Temporary Field		
วิธีการรีแฟคทอริ่งสำหรับการแก้ไขร่องรอยไม่ดี					
1. Moving Features between Objects วิธีย่อยที่ใช้ Extract Class 2. Simplifying Conditional Expression วิธีย่อยที่ใช้ Introduce Null Object 3. Organizing Data วิธีย่อยที่ใช้ Encapsulate Field					
วิธีที่เลือกใช้ในการปรับปรุงร่องรอยไม่ดี			Organizing Data วิธีย่อยที่ใช้ Encapsulate Field		
เหตุผลที่เลือกใช้					
เมื่อแก้ไขปัญหาแรกเสร็จแล้ว จึงแก้ไขปัญหาเรื่องแอตทริบิวต์ที่มีการเรียกใช้ เพราะแอตทริบิวต์ที่เรียกใช้งาน ถูกสร้างขึ้นเพื่อเก็บค่าชั่วคราว เท่านั้น ดังนั้นควรใช้วิธีการ Encapsulate Field เพื่อเป็นการป้องกันการแก้ไขค่าโดยตรงและปกปิดข้อมูลให้กับโปรแกรมวิธีการปฏิบัติอื่น					

ตารางที่ จ.6 แนวทางการปรับปรุงโค้ดกลุ่มโค้ดร่องรอยไม่ดี Hit_the_keys ครั้งที่ 2 (ต่อ)

เหตุผลที่เลือกใช้
เช่น Extract Class ไม่เหมาะสม เนื่องจากคลาสมีหน้าที่รับผิดชอบเพียงอย่างเดียว การแบ่งแยกอาจนำไปสู่ร่องรอยไม่ดีประเภท Lazy Class ได้ และการเลือกใช้ Introduce Null Object ก็ไม่เหมาะสมกับกรณีนี้ เพราะแอตทริบิวต์ภายในคลาสนี้เป็นประเภทตัวเลข

ตารางที่ จ.7 ผลลัพธ์หลังจากการปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี Hit the Keys ครั้งที่ 2

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		Hit_the_keys																												
จำนวนคลาสที่ใช้ทดสอบ		3					จำนวนเมท็อดที่ใช้ทดสอบ					17							จำนวนแอตทริบิวต์ที่ใช้ทดสอบ					17							
		มาตรวัดที่ใช้ในการทดสอบ																												
ชื่อคลาสที่ใช้ทดสอบ		ACML	CC	ECML	ILCC	ILND	NILI	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-OME	SEC-IM	ข้อจำกัด
1)	Form1																													
	ชื่อเมท็อดที่ใช้ทดสอบ																													
	# Constructor	1	1	4	1	0	18	5	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- Form1_KeyDown	1	5	16	6	2	145	16	1	2	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- InitializeComponent	1	1	55	1	0	396	71	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M1
	- timer1_Tick	1	2	6	2	1	41	5	1	2	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ Dispose	0	3	1	4	1	23	3	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
	- components_3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- listBox1_3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- timer1_3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- difficultyProgressBar_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- correctLabel_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- missedLabel_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- accuracyLabel_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- random_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- totalLabel_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- stats_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- statusStrip1_1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- toolStripStatusLabel1_1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ค่าที่วัดได้											0	30	7	1	1	1	12	0	0	7	31								C1
2)	Program																													
	ชื่อเมท็อดที่ใช้ทดสอบ																													
	- Main	0	1	4	1	0	10	3	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ค่าที่วัดได้											0	3	1	1	0	0	0	0	0	1	50								
3)	Stats																													
	ชื่อเมท็อดที่ใช้ทดสอบ																													
	# Constructor	1	1	1	1	0	18	5	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ get_Total	2	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ dumpResult	1	1	1	1	0	41	6	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ Counter	1	1	0	1	0	8	1	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ Update	1	1	3	1	0	24	4	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ isCorrectKey	1	2	0	3	1	23	3	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ get_Missed	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ get_Accuracy	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ get_Correct	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ getCurrentState	0	1	0	1	0	8	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ ResetCounter	0	1	0	1	0	5	1	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
	- _missed_5	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- _correct_5	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- _accuracy_4	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- _total_4	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- _currentState_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ค่าที่วัดได้											0	1	1	1	1	1	5	0	0	11	0								
รวมทั้งสิ้น																							0	0	2	1	0	0		

ตารางที่ จ.8 ผลลัพธ์แสดงค่าการแปลความหลังจากการปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี Hit the Keys ครั้งที่ 2

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		Hit_the_keys			
		ชุดรายการ		Defuzzification Output	สรุปค่าแปลความ
เซ็ตที่	คลาส	เมธอด	ฟิลด์		
1	Form1	Constructor	components 3	0.303	Clean
2		Form1_KeyDown	listBox1 3	0.324	Clean
3		InitializeComponent	timer1 3	0.317	Clean
		InitFirst	difficultyProgressBar 2	0.317	Clean
		InitForm	correctLabel 2	0.321	Clean
4		timer1_Tick	missedLabel 2	0.303	Clean
5		Dispose	accuracyLabel 2	0.327	Clean
6			random 2		
7			totalLabel 2		
8			stats 2		
9			statusStrip1 1		
10			toolStripStatusLabel1 1		
11					
12					
13	Program	Main		0.263	Clean
14	Stats	Constructor	_missed 5	0.306	Clean
15		get_Total	_correct 5	0.306	Clean
16		dumpResult	_accuracy 4	0.306	Clean
17		Counter	_total 4	0.306	Clean
18		Update	_currentState 2	0.306	Clean
19		isCorrectKey		0.306	Clean
20		get_Missed		0.306	Clean
21		get_Accuracy		0.306	Clean
22		get_Correct		0.306	Clean
23		getCurrentState		0.329	Clean
24		ResetCounter		0.329	Clean

ตารางที่ จ.9 ผลลัพธ์ที่วัดได้เป็นร่องรอยไม่ดี Feature Envy

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ	FeatureEnvy																													
จำนวนคลาสที่ใช้ทดสอบ	3					จำนวนเมท็อดที่ใช้ทดสอบ					15								จำนวนแอตทริบิวต์ที่ใช้ทดสอบ					5							
	มาตรวัดที่ใช้ในการทดสอบ																													
ชื่อคลาสที่ใช้ทดสอบ	ACML	CC	ECML	ILCC	ILND	NII	NLOC	NOO	NOP	NOV	ACNPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOAM	RC	SAC-OMF	SEC-IM	ข้อจำกัด	
1) Calendar																														
ชื่อเมท็อดที่ใช้ทดสอบ																														
# Constructor1	2	1	1	1	0	6	1	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
# Constructor2	0	1	2	1	0	4	1	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
+ setTime	1	1	0	1	0	4	1	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
+ get_Year	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
+ get_Month	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
+ getMonth	1	1	0	1	0	9	1	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
+ GetInstance	1	1	1	1	0	6	1	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
+ createCalendar	1	1	2	1	0	12	3	1	1	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
+ getYear	1	1	1	1	0	9	1	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																														
+ _myDate	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
+ instance	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
ค่าที่วัดได้											0	1	1	1	0	0	2	0	0	9	0									
2) InsuranceQuote																														
ชื่อเมท็อดที่ใช้ทดสอบ																														
# Constructor1	1	1	1	1	0	9	2	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
+ calculateMotoristrisk	0	4	2	6	1	39	5	1	0	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																														
+ motorist	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
ค่าที่วัดได้											1	2	1	1	0	0	2	0	0	2	0									
3) Motorist																														
ชื่อเมท็อดที่ใช้ทดสอบ																														
# Constructor1	1	1	1	1	0	12	3	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
+ getAge	1	1	8	1	0	36	6	1	0	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
- adustYearsDownIfNegativeMonthDif	1	2	0	2	1	20	3	1	2	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
+ getPointsOnLicense	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																														
+ dateOfBirth	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
+ pointsOnLicense	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	
ค่าที่วัดได้											0	2	1	1	1	1	1	0	0	2	0									
รวมทั้งสิ้น																						0	1	3	1	1	0			

ตารางที่ จ.10 ผลลัพธ์ที่แสดงค่าการแปลความ Feature Envy

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			FeatureEnvy		
ชุดรายการ				Defuzzification Output	สรุปค่าแปลความ
เซตที่	คลาส	เมท็อด	แอตทริบิวต์		
1	Calendar	Constructor1	_myDate	0.799	bad
2		Constructor2	instance	0.799	bad
3		setTime		0.799	bad
4		get_Year		0.799	bad
5		get_Month		0.799	bad
6		getMonth		0.799	bad
7		GetInstance		0.799	bad
8		createCalendar		0.799	bad
9		getYear		0.799	bad
10	InsuranceQuote	Constructor1	motorist	0.799	bad
11		calculateMotoristrisk		0.799	bad
12	Motorist	Constructor1	dateOfBirth	0.799	bad
13		getAge	pointsOnLicense	0.799	bad
14		adustYearsDownIfNegative MonthDifference		0.799	bad
15		getPointsOnLicense		0.799	bad

ตารางที่ จ.11 แนวทางการปรับปรุงโค้ดกลุ่มโค้ดร่องรอยไม่ดี Feature Envy ครั้งที่ 1

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			Feature Envy		
ลำดับที่	**คลาส**	**เมท็อด**	**แอตทริบิวต์**	**ผลลัพธ์การแปลความ**	**สรุปค่าแปลความ**
1	Calendar	Constructor1	_myDate	0.799	bad
2		Constructor2	instance	0.799	bad
3		setTime		0.799	bad
4		get_Year		0.799	bad
5		get_Month		0.799	bad
6		getMonth		0.799	bad
7		GetInstance		0.799	bad
8		createCalendar		0.799	bad
9		getYear		0.799	bad
10	InsuranceQuote	Constructor1	motorist	0.799	bad
11		calculateMotoristrisk		0.799	bad
12	Motorist	Constructor1	dateOfBirth	0.799	bad
13		getAge	pointsOnLicense	0.799	bad
14		adustYearsDownIfNegativeMonthDifference		0.799	bad
15		getPointsOnLicense		0.799	bad
ประเภทร่องรอยไม่ดีที่ต้องการแก้ไข			Feature Envy และ Lazy Class		

ตารางที่ จ.11 แนวทางการปรับปรุงโค้ดกลุ่มโค้ดร่องรอยไม่ดี Feature Envy ครั้งที่ 1(ต่อ)

วิธีการรีแฟคทอริ่งสำหรับการแก้ไขร่องรอยไม่ดี	
1. Dealing with Generalization วิธีย่อยที่ใช้ Collapse Hierarchy 2. Moving Features between Objects วิธีย่อยที่ใช้ Inline Class 3. Composing Method วิธีย่อยที่ใช้ Extract Method 4. Moving Features between Objects วิธีย่อยที่ใช้ Move Method 5. Composing Method วิธีย่อยที่ใช้ Inline Method	
วิธีที่เลือกใช้ในการปรับปรุงร่องรอยไม่ดี	Moving Features between Objects วิธีย่อยที่ใช้ Move Method
เหตุผลที่เลือกใช้	
ตัวอย่างนี้มีร่องรอยไม่ดีเกิ ดขึ้นซ้ำซ้อนกันสองประเภท ควรเริ่มแก้จากตัวที่เป็นสาเหตุหลักก่อน คือ ร่องรอยไม่ดีประเภท Lazy Class เกิดจากการทำงานของเมท็อด ไม่เหมาะสมระหว่างคลาส ทำให้จำนวนเมท็อดที่ถูกใช้งานมีจำนวนน้อย จากสาเหตุนี้ควรย้ายเมท็อดที่มีการทำงานเกี่ยวข้องกันมาอยู่ภายในคลาสเดียวกันเพื่อแก้ไขปัญหาร่องรอยไม่ดีนี้ จึงใช้ Move Method เพื่อทำการย้ายเมท็อดมาไว้ในขอบเขตความรับผิดชอบของคลาสที่เรียกใช้งาน ทั้งนี้วิธีปฏิบัติ Collapse Hierarchy ไม่สามารถแก้ไขร่องรอยไม่ดีในตัวอย่างโค้ดนี้ได้ เนื่องจากไม่ได้เกิดจากการสืบทอดคุณสมบัติ ส่วนวิธี ปฏิบัติ Inline Method Extract Method และ Move Method ไม่เหมาะสมกั บการแก้ไข เนื่องจากแต่ละคลาสทำงานเพียงหนึ่งอย่าง ไม่เป็นเซตย่อย และไม่มีส่วน การทำงาน ที่ซ้ำซ้อนกัน	

ตารางที่ จ.12 ผลลัพธ์หลังจากการปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี Feature Envy ครั้งที่ 1

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ	FeatureEnvy																													
จำนวนคลาสที่ใช้ทดสอบ	3					จำนวนเมท็อดที่ใช้ทดสอบ					13								จำนวนแอตทริบิวต์ที่ใช้ทดสอบ					5						
	มาตรวัดที่ใช้ในการทดสอบ																													
ชื่อคลาสที่ใช้ทดสอบ	ACML	CC	ECML	ILCC	ILND	NILI	NLOC	NOO	NOP	NOV	ACNPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-OMF	SEC-IM	ข้อจำกัด
1) Calendar																														
ชื่อเมท็อดที่ใช้ทดสอบ																														
# Constructor1	2	1	1	1	0	6	1	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
# Constructor2	0	1	2	1	0	4	1	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ setTime	1	1	0	1	0	4	1	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_Year	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_Month	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ getMonth	1	1	0	1	0	9	1	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ GetInstance	1	1	1	1	0	6	1	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ createCalendar	1	1	2	1	0	12	3	1	1	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ getYear	1	1	1	1	0	9	1	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																														
+ _myDate	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ instance	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	1	1	1	0	0	2	0	0	9	0								
2) InsuranceQuote																														
ชื่อเมท็อดที่ใช้ทดสอบ																														
# Constructor1	1	1	1	1	0	9	2	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ calculateMotoristrisk	0	1	1	1	0	8	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																														
+ motorist	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												1	2	1	1	0	0	1	0	0	2	25								
3) Motorist																														
ชื่อเมท็อดที่ใช้ทดสอบ																														
# Constructor1	1	1	1	1	0	9	2	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ calculateMotoristRisk	0	1	1	1	0	8	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																														
- dateOfBirth 2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- pointsOnLicense 2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												2	9	1	1	1	1	2	0	0	5	14								
รวมทั้งสิ้น																							0	1	3	1	0	0		

ตารางที่ จ.13 ผลลัพธ์แสดงค่าการแปลความหลังจากการปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี Feature Envy ครั้งที่ 1

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		FeatureEnvy			
	ชุดรายการ			Defuzzification Output	สรุปค่าแปลความ
เซตที่	คลาส	เมท็อด	แอตทริบิวต์		
1	Calendar	Constructor1	_myDate	0.25	Clean
2		Constructor2	instance	0.273	Clean
3		setTime		0.25	Clean
4		get_Year		0.25	Clean
5		get_Month		0.25	Clean
6		getMonth		0.25	Clean
7		GetInstance		0.25	Clean
8		createCalendar		0.25	Clean
9		getYear		0.25	Clean
10	InsuranceQuote	Constructor1	motorist	0.799	Bad
11		calculateMotoristrisk		0.799	Bad
12	Motorist	Constructor1	dateOfBirth 2	0.281	Clean
13		calculateMotoristRisk	pointsOnLicense 2	0.306	Clean

ตารางที่ จ.14 แนวทางการปรับปรุงโค้ดกลุ่มโค้ดร่องรอยไม่ดี Feature Envy ครั้งที่ 2

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			Feature Envy		
ลำดับที่	**คลาส**	**เมท็อด**	**แอตทริบิวต์**	**ผลลัพธ์การแปลความ**	**สรุปค่าแปลความ**
1	Calendar	Constructor1	_myDate	0.799	Bad
2		Constructor2	instance	0.799	Bad
3		setTime		0.799	Bad
4		get_Year		0.799	Bad
5		get_Month		0.799	Bad
6		getMonth		0.799	Bad
7		GetInstance		0.799	Bad
8		createCalendar		0.799	Bad
9		getYear		0.799	Bad
10	InsuranceQuote	Constructor1	motorist	0.799	Bad
11		calculateMotoristrisk		0.799	Bad
12	Motorist	Constructor1	dateOfBirth 2	0.799	Bad
13		calculateMotoristRisk	pointsOnLicense 2	0.799	Bad
ประเภทร่องรอยไม่ดีที่ต้องการแก้ไข			Feature Envy		
วิธีการรีแฟคทอริ่งสำหรับการแก้ไขร่องรอยไม่ดี					
1. Dealing with Generalization วิธีย่อยที่ใช้ Collapse Hierarchy 2. Moving Features between Objects วิธีย่อยที่ใช้ Inline Class 3. Organizing Data วิธีย่อยที่ใช้ Encapsulate Field					
วิธีที่เลือกใช้ในการปรับปรุงร่องรอยไม่ดี			Organizing Data วิธีย่อยที่ใช้ Encapsulate Field		
เหตุผลที่เลือกใช้					
เมื่อแก้ไขร่องรอยไม่ดีแล้ว ตัวอย่างนี้ยังมีร่องรอยไม่ดีอีกชนิดคือ Feature Envy ตามวิธีที่แนะนำควรเปลี่ยนมาแก้ไขโดย Encapsulate Field แก้ไขปัญหาการประกาศใช้งานแอตทริบิวต์ในคลาสโดยอาจถูกเรียกใช้งานน้อยหรือเพียงครั้งเดียวซึ่งเป็นการเรียกใช้งานจากภายนอก ให้มีการปกปิดข้อมูลหรือมีลำดับชั้นในการเข้าถึงข้อมูล ทำให้เกิดการใช้งานของแอตทริบิวต์มากขึ้น ในส่วนวิธีปฏิบัติ Collapse Hierarchy ไม่สามารถแก้ไขได้ เพราะคลาสที่มีร่องรอยไม่ดีเกิดขึ้นนั้นไม่ได้ถูกสืบทอดมา และส่วน Inline Class ไม่สามารถแก้ไขได้ เพราะคลาสไม่ได้มีการทำงานแบบเซตย่อยหรือหน้าที่การทำงานที่ไม่ซ้ำซ้อนกัน					

ตารางที่ จ.15 ผลลัพธ์หลังจากการปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี Feature Envy ครั้งที่ 2

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			FeatureEnvy																											
จำนวนคลาสที่ใช้ทดสอบ			3					จำนวนเมท็อดที่ใช้ทดสอบ					15					จำนวนแอตทริบิวต์ที่ใช้ทดสอบ					5							
			มาตรวัดที่ใช้ในการทดสอบ																											
ชื่อคลาสที่ใช้ทดสอบ			AOML	CC	EOML	ILCC	ILND	NILI	NLOC	NOO	NOP	NOV	AC_NPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC_OMF	SEC_IM	ข้อจำกัด
1)	Calendar																													
	ชื่อเมท็อดที่ใช้ทดสอบ																													
	#	Constructor1	2	1	1	1	0	6	1	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	#	Constructor2	0	1	2	1	0	4	1	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+	setTime	1	1	0	1	0	4	1	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+	get_Year	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+	get_Month	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+	getMonth	1	1	0	1	0	9	1	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+	GetInstance	1	1	1	1	0	6	1	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+	createCalendar	1	1	2	1	0	12	3	1	1	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+	getYear	1	1	1	1	0	9	1	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
	+	_myDate	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+	instance	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ค่าที่วัดได้												0	1	1	1	0	0	2	0	0	9	0								
2)	InsuranceQuote																													
	ชื่อเมท็อดที่ใช้ทดสอบ																													
	#	Constructor1	1	1	1	1	0	9	2	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+	calculateMotoristrisk	0	1	1	1	0	8	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+	set_MotorisQuote	0	1	0	1	0	5	1	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+	get_MotorisQuote	0	1	1	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
	-	_motorist	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ค่าที่วัดได้												0	2	1	1	0	0	1	0	0	4	0								
3)	Motorist																													
	ชื่อเมท็อดที่ใช้ทดสอบ																													
	#	Constructor1	1	1	1	1	0	9	2	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+	calculateMotoristRisk	0	1	1	1	0	8	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
	-	dateOfBirth_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	-	pointsOnLicense_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ค่าที่วัดได้												0	3	1	1	1	1	2	0	0	5	0								
รวมทั้งสิ้น																								0	0	3	1	0	0		

ตารางที่ จ.16 ผลลัพธ์แสดงค่าการแปลความหลังจากการปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี Feature Envy ครั้งที่ 2

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		FeatureEnvy			
	ชุดรายการ			Defuzzification Output	สรุปค่าแปลความ
เซตที่	คลาส	เมท็อด	แอตทริบิวต์		
1	Calendar	Constructor1	_myDate	0.25	Clean
2		Constructor2	instance	0.273	Clean
3		setTime		0.25	Clean
4		get_Year		0.25	Clean
5		get_Month		0.25	Clean
6		getMonth		0.25	Clean
7		GetInstance		0.25	Clean
8		createCalendar		0.25	Clean
9		getYear		0.25	Clean
10	InsuranceQuote	Constructor1	_motorist	0.25	Clean
11		calculateMotoristrisk		0.273	Clean
12		set_MotorisQuote		0.273	Clean
13		get_MotorisQuote		0.273	Clean
14	Motorist	Constructor1	dateOfBirth_2	0.281	Clean
15		calculateMotoristRisk	pointsOnLicense_2	0.306	Clean

ตารางที่ จ.17 ผลลัพธ์ที่วัดได้เป็นร่องรอยไม่ดี Party Planner 2

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		Party Planner 2																													
จำนวนคลาสที่ใช้ทดสอบ		4					จำนวนเมท็อดที่ใช้ทดสอบ									26							จำนวนแอตทริบิวต์ที่ใช้ทดสอบ							28			
		มาตรวัดที่ใช้ในการทดสอบ																													
ชื่อคลาสที่ใช้ทดสอบ		ACML	CC	ECML	ILCC	ILND	NILI	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-OMF	SEC-IM	ข้อจำกัด	
1) BirthdayParty																															
ชื่อเมท็อดที่ใช้ทดสอบ																															
#	Constructor	1	1	5	1	0	30	8	1	3	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	CalculateCostOfDecorations	3	2	6	3	1	42	4	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	get_NumberOfPeople	3	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	set_CakeWriting	3	4	4	6	2	60	9	1	1	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	get_CakeWriting	2	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	CalculateCakeSize	2	2	1	3	1	16	3	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	set_NumberOfPeople	1	1	3	1	0	18	4	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	CalculateCost	1	2	8	3	1	57	5	1	0	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																															
-	cakeWriting 4	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	numberOfPeople 3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	CakeSize	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	CakeOfDecorations	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	fancyDecorations 3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												2	2	1	1	1	1	6	0	0	8	16									
2) DinnerParty																															
ชื่อเมท็อดที่ใช้ทดสอบ																															
#	Constructor	1	1	5	1	0	25	6	1	3	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	CalculateCostOfDecorations	3	2	6	3	1	46	4	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	SetHealthyOption	2	2	1	3	1	29	3	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	get_NumberOfPeople	2	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	set_NumberOfPeople	2	1	1	1	0	10	2	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	CalculateCost	1	2	6	3	1	37	4	1	1	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																															
+	CostOfDecorations	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	fancyDecorations 3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	numberOfPeople 2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	CostOfBeveragesPerPerson	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												2	1	1	1	1	1	5	0	0	6	0									
3) Form1																															
ชื่อเมท็อดที่ใช้ทดสอบ																															
#	Constructor	1	1	10	1	0	41	6	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	DisplayBirthdayPartyCost	4	1	4	1	0	20	3	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		Party Planner 2																													
จำนวนคลาสที่ใช้ทดสอบ		4					จำนวนเมท็อดที่ใช้ทดสอบ									26							จำนวนแอตทริบิวต์ที่ใช้ทดสอบ							28			
		มาตรวัดที่ใช้ในการทดสอบ																													
ชื่อคลาสที่ใช้ทดสอบ		ACML	CC	ECML	ILCC	ILND	NILI	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-OMF	SEC-IM	ข้อจำกัด	
-	DisplayDinnerPartyCost	4	1	4	1	0	16	2	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	InitializeComponent	1	1	53	1	0	965	166	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M1
-	fancyBirthday_CheckedChanged	1	1	3	1	0	12	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	cakeWriting_TextChanged	1	1	3	1	0	12	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	numberBirthday_ValueChanged	1	1	4	1	0	13	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	fancybox_CheckedChanged	1	1	3	1	0	12	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	numberUpDown1_ValueChanged	1	1	4	1	0	13	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	healthyBox_CheckedChanged	1	1	3	1	0	12	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	Dispose	0	3	1	4	1	23	3	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																															
-	birthdayParty 5	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	dinnerParty 5	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	healthyBox 4	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	cakeWriting 4	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	numericUpDown1 3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	numberBirthday 3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	fancyBox 3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	fancyBirthday 3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	birthdayCost 2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	costLabel 2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	label5 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	tabPage1 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	label6 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	label4 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	label2 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	tabPage2 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	tabControl1 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	label1 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	components 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	23	7	1	1	1	19	0	0	11	25									C1
4) Program																															
ชื่อเมท็อดที่ใช้ทดสอบ																															
-	Main	0	1	4	1	0	10	3	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	3	1	1	0	0	0	0	0	1	50									
รวมทั้งสิ้น																							0	0	4	1	0	0		

ตารางที่ จ.18 ผลลัพธ์ที่แสดงค่าการแปลความ Party Planner 2

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		Party Planner 2			
ชุดรายการ				Defuzzification Output	สรุปค่าแปลความ
เซตที่	คลาส	เมท็อด	แอตทริบิวต์		
1	BirthdayParty	Constructor	cakeWriting 4	0.313	Clean
2		CalculateCostOfDecorations	numberOfPeople 3	0.313	Clean
3		get_NumberOfPeople	fancyDecorations 3	0.313	Clean
4		set_CakeWriting		0.328	Clean
5		get_CakeWriting		0.313	Clean
6		CalculateCakeSize		0.313	Clean
7		set_NumberOfPeople		0.313	Clean
8		CalculateCost		0.323	Clean
1		Constructor	CakeSize	0.799	Bad
2		CalculateCostOfDecorations	CakeOfDecorations	0.799	Bad
3		get_NumberOfPeople		0.799	Bad
4		set_CakeWriting		0.799	Bad
5		get_CakeWriting		0.799	Bad
6		CalculateCakeSize		0.799	Bad
7		set_NumberOfPeople		0.799	Bad
8		CalculateCost		0.799	Bad
9	DinnerParty	Constructor	fancyDecorations 3	0.321	Clean
10		CalculateCostOfDecorations	numberOfPeople 2	0.321	Clean
11		SetHealthyOption		0.321	Clean
12		get_NumberOfPeople		0.321	Clean
13		set_NumberOfPeople		0.321	Clean
14		CalculateCost		0.321	Clean
15		Constructor	CostOfBeveragesPerPers	0.799	Bad
16		CalculateCostOfDecorations		0.799	Bad
17		SetHealthyOption		0.799	Bad
18		get_NumberOfPeople		0.799	Bad
19		set_NumberOfPeople		0.799	Bad
20		CalculateCost		0.799	Bad
21		Constructor	CostOfDecorations	0.799	Bad
22		CalculateCostOfDecorations		0.799	Bad
23		SetHealthyOption		0.799	Bad
24		get_NumberOfPeople		0.799	Bad
25		set_NumberOfPeople		0.799	Bad
26		CalculateCost		0.799	Bad
27	Form1	Constructor	birthdayParty 5	0.305	Clean
28		DisplayBirthdayPartyCost	dinnerParty 5	0.305	Clean
29		DisplayDinnerPartyCost	healthyBox 4	0.305	Clean
30		InitializeComponent	cakeWriting 4	0.335	Ambiguous
31		fancyBirthday_CheckedChange	numericUpDown1 3	0.305	Clean
32		cakeWriting_TextChanged	numberBirthday 3	0.305	Clean
33		numberBirthday_ValueChange	fancyBox 3	0.305	Clean
34		fancybox_CheckedChanged	fancyBirthday 3	0.305	Clean
35		numberUpDown1_ValueChang	birthdayCost 2	0.305	Clean
36		healthyBox_CheckedChanged	costLabel 2	0.328	Ambiguous
37		Dispose	label5 1	0.328	Ambiguous
38			tabPage1 1		
39			label6 1		
40			label4 1		
41			label2 1		
42			tabPage2 1		
43			tabControl1 1		
44			label1 1		
45			components 1		
46	Program	Main		0.262	Clean

ตารางที่ จ.19 แนวทางการปรับปรุงโค้ดกลุ่มโค้ดร่องรอยไม่ดี Party Planner 2 ครั้งที่ 1

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			Party Planner 2		
ลำดับที่	**คลาส**	**เมท็อด**	**แอตทริบิวต์**	**ผลลัพธ์การแปลความ**	**สรุปค่าแปลความ**
1	BirthdayParty	Constructor	CakeSize	0.799	Bad
2		CalculateCostOfDecorations	CakeOfDecorations	0.799	Bad
3		get_NumberOfPeople		0.799	Bad
4		set_CakeWriting		0.799	Bad
5		get_CakeWriting		0.799	Bad
6		CalculateCakeSize		0.799	Bad
7		set_NumberOfPeople		0.799	Bad
8		CalculateCost		0.799	Bad
9	DinnerParty	Constructor	CostOfDecorations	0.799	Bad
10		CalculateCostOfDecorations		0.799	Bad
11		SetHealthyOption		0.799	Bad
12		get_NumberOfPeople		0.799	Bad
13		set_NumberOfPeople		0.799	Bad
14		CalculateCost		0.799	Bad
15		Constructor	CostOfBeveragesPerPerson	0.799	Bad
16		CalculateCostOfDecorations		0.799	Bad
17		SetHealthyOption		0.799	Bad
18		get_NumberOfPeople		0.799	Bad
19		set_NumberOfPeople		0.799	Bad
20		CalculateCost		0.799	Bad
ประเภทร่องรอยไม่ดีที่ต้องการแก้ไข			Temporary Field		
วิธีการรีแฟคทอริ่งสำหรับการแก้ไขร่องรอยไม่ดี					
1. Moving Features between Objects วิธีย่อยที่ใช้ Extract Class 2. Simplifying Conditional Expression วิธีย่อยที่ใช้ Introduce Null Object 3. Organizing Data วิธีย่อยที่ใช้ Encapsulate Field					
วิธีที่เลือกใช้ในการปรับปรุงร่องรอยไม่ดี			Organizing Data วิธีย่อยที่ใช้ Encapsulate Field		

ตารางที่ จ.19 แนวทางการปรับปรุงโค้ดกลุ่มโค้ดร่องรอยไม่ดี Party Planner 2 ครั้งที่ 1 (ต่อ)

เหตุผลที่เลือกใช้
สังเกตจากค่าแปลความที่ค่อนข้างชี้ไปที่แอตทริบิวต์ที่เกิดร่องรอยไม่ดี ดังนั้นปัญหาไม่ได้เกิดจากเมท็อดหรือเงื่อนไขภายในคลาส แม้ว่าการเลือกใช้วิธี Extract Class ซึ่งเป็นวิธีการแก้ไขแบบกว้างก็สามารถแก้ไขได้เช่นกัน แต่อาจต้องเลือกวิธีการอื่นเพิ่มเติมเพื่อให้การปรับปรุงนั้นมีประสิทธิภาพมากขึ้น เนื่องจากคลาส BirthdayParty และ DinnerParty มีการทำงานในขอบเขตที่ชัดเจน การแบ่งคลาสอาจทำให้เกิดร่องรอยไม่ดีประเภท Lazy Class ได้ และวิธี Introduce Null Object ไม่สามารถนำมาใช้แก้ไข เพราะข้อมูลอยู่ในรูปแบบจำนวนเต็มไม่ใช่การอ้างถึงวัตถุอื่น ทั้งนี้การเลือก Encapsulate Field ตามคำแนะนำของผู้เชี่ยวชาญน่าจะเป็นการแก้ไขที่ต้นเหตุมากกว่า ซึ่งต้องพิจารณาจากค่าการแปลความที่ได้ประกอบไว้

ตารางที่ จ.20 ผลลัพธ์หลังจากการปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี Party Planner 2 ครั้งที่ 1

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		Party Planner 2																											
จำนวนคลาสที่ใช้ทดสอบ		4				จำนวนเมท็อดที่ใช้ทดสอบ					37							จำนวนแอตทริบิวต์ที่ใช้ทดสอบ					30						
		มาตรวัดที่ใช้ในการทดสอบ																											
ชื่อคลาสที่ใช้ทดสอบ		ACoML	CC	ECoML	ILCC	LLND	NILI	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-OMIE	SEC-IM	ข้อจำกัด
1) BirthdayParty																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
#	Constructor	1	1	5	1	0	34	9	1	3	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	get_NumberOfPeople	3	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	set_CakeWriting	3	4	5	6	2	60	9	1	1	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	CalculateCostOfDecorations	3	2	7	3	1	44	4	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	get_CakeWriting	2	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	CalculateCakeSize	2	2	2	3	1	18	3	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	get_CakeSize	2	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	CalculateCost	1	2	11	3	1	58	5	1	0	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	get_CostOfFoodPerPerson	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	set_NumberOfPeople	1	1	3	1	0	18	4	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	get_CostOfDecorations	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	set_CakeSize	1	2	0	2	1	11	2	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	set_CostOfDecorations	1	2	2	2	1	14	2	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	set_CostOfFoodPerPerson	0	2	2	2	1	14	2	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
-	cakeWriting 4	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	_costOfDecorations 3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	numberOfPeople 3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	fancyDecorations 3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	_costofFoodPerPerson 3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	_cakesize 2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	2	1	1	1	1	6	0	0	14	15							
2) DinnerParty																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
#	Constructor	1	1	5	1	0	29	7	1	3	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	CalculateCostOfDecorations	2	2	2	3	1	31	3	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	get_NumberOfPeople	2	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	set_NumberOfPeople	2	1	1	1	0	10	2	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	set_CostOfDecorations	1	2	2	2	1	14	2	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	CalculateCost	1	2	8	3	1	37	4	1	1	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	get_CostOfBeveragePerPerson	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	get_CostOfFoodPerPerson	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	get_CostOfDecorations	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	set_CostOfBeveragePerPerson	1	2	2	2	1	14	2	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	set_CostOfFoodPerPerson	0	2	2	2	1	14	2	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

ตารางที่ จ.20 ผลลัพธ์หลังจากการปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี Party Planner 2 ครั้งที่ 1 (ต่อ)

	ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																														
	- fancyDecorations 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- _costOfDecorations 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- _costofFoodPerPerson 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- numberOfPeople 2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- CostOfBeveragesPerPerson 2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ค่าที่วัดได้												0	1	1	1	1	1	5	0	0	12	0								
3)	Form1																														
	ชื่อเมท็อดที่ใช้ทดสอบ																														
	# Constructor	1	1	10	1	0	41	6	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- DisplayBirthdayPartyCost	4	1	4	1	0	20	3	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- DisplayDinnerPartyCost	4	1	4	1	0	16	2	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- InitializeComponent	1	1	53	1	0	965	166	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M1
	- fancyBirthday_CheckedChanged	1	1	3	1	0	12	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- cakeWriting_TextChanged	1	1	3	1	0	12	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- numberBirthday_ValueChanged	1	1	4	1	0	13	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- fancybox_CheckedChanged	1	1	3	1	0	12	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- numberUpDown1_ValueChanged	1	1	4	1	0	13	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- healthyBox_CheckedChanged	1	1	3	1	0	12	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ Dispose	0	3	1	4	1	23	3	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																														
	- birthdayParty 5	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- dinnerParty 5	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- healthyBox 4	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- cakeWriting 4	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- numericUpDown1 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- numberBirthday 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- fancyBox 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- fancyBirthday 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- birthdayCost 2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- costLabel 2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- label5 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- tabPage1 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- label6 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- label4 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- label2 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- tabPage2 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- tabControl1 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- label1 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- components 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ค่าที่วัดได้												0	23	7	1	1	1	19	0	0	11	25								C1
4)	Program																														
	ชื่อเมท็อดที่ใช้ทดสอบ																														
	- Main	0	1	4	1	0	10	3	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ค่าที่วัดได้												0	3	1	1	0	0	0	0	0	1	50								
รวมทั้งสิ้น																								0	0	4	1	0	0		

ตารางที่ จ.21 ผลลัพธ์แสดงค่าการแปลความหลังจากการปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี Party Planner 2 ครั้งที่ 1

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			Party Planner 2		
	ชุดรายการ			Defuzzification Output	สรุปค่าแปลความ
เซตที่	คลาส	เมท็อด	แอตทริบิวต์		
1	BirthdayParty	Constructor	cakeWriting 4	0.278	Clean
2		get_NumberOfPeople	_costOfDecorations 3	0.278	Clean
3		set_CakeWriting	numberOfPeople 3	0.29	Clean
4		CalculateCostOfDecorations	fancyDecorations 3	0.278	Clean
5		get_CakeWriting	_costofFoodPerPerson 3	0.278	Clean
6		CalculateCakeSize	_cakesize 2	0.278	Clean
7		get_CakeSize		0.278	Clean
8		CalculateCost		0.287	Clean
9		get_CostOfFoodPerPerson		0.278	Clean
10		set_NumberOfPeople		0.278	Clean
11		get_CostOfDecorations		0.278	Clean
12		set_CakeSize		0.278	Clean
13		set_CostOfDecorations		0.278	Clean
14		set_CostOfFoodPerPerson		0.304	Clean
15	DinnerParty	Constructor	fancyDecorations 3	0.294	Clean
16		CalculateCostOfDecorations	_costOfDecorations 3	0.294	Clean
17		get_NumberOfPeople	_costofFoodPerPerson 3	0.294	Clean
18		set_NumberOfPeople	numberOfPeople 2	0.294	Clean
19		set_CostOfDecorations	CostOfBeveragesPerPers	0.294	Clean
20		CalculateCost		0.294	Clean
21		get_CostOfBeveragePerPerson		0.294	Clean
22		get_CostOfFoodPerPerson		0.294	Clean
23		get_CostOfDecorations		0.294	Clean
24		set_CostOfBeveragePerPerson		0.294	Clean
25		set_CostOfFoodPerPerson		0.319	Clean
26	Form1	Constructor	birthdayParty 5	0.305	Clean
27		DisplayBirthdayPartyCost	dinnerParty 5	0.305	Clean
28		DisplayDinnerPartyCost	healthyBox 4	0.305	Clean
29		InitializeComponent	cakeWriting 4	0.335	Ambigous
30		fancyBirthday_CheckedChang	numericUpDown1 3	0.305	Clean
31		cakeWriting_TextChanged	numberBirthday 3	0.305	Clean
32		numberBirthday_ValueChange	fancyBox 3	0.305	Clean
33		fancybox_CheckedChanged	fancyBirthday 3	0.305	Clean
34		numberUpDown1_ValueChan	birthdayCost 2	0.305	Clean
35		healthyBox_CheckedChanged	costLabel 2	0.305	Clean
36		Dispose	label5 1	0.328	Clean
37			tabPage1 1		
38			label6 1		
39			label4 1		
40			label2 1		
41			tabPage2 1		
42			tabControl1 1		
43			label1 1		
44			components 1		
45	Program	Main		0.262	Clean

ตารางที่ จ.22 แนวทางการปรับปรุงโค้ดกลุ่มโค้ดร่องรอยไม่ดี Party Planner 2 ครั้งที่ 2

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			Party Planner 2		
ลำดับที่	**คลาส**	**เมท็อด**	**แอตทริบิวต์**	**ผลลัพธ์การแปลความ**	**สรุปค่าแปลความ**
1	Form1	InitializeComponent	birthdayParty 5	0.335	Ambiguity
2			dinnerParty 5	0.335	Ambiguity
3			healthyBox 4	0.335	Ambiguity
4			cakeWriting 4	0.335	Ambiguity
5			numericUpDown1 3	0.335	Ambiguity
6			numberBirthday 3	0.335	Ambiguity
7			fancyBox 3	0.335	Ambiguity
8			fancyBirthday 3	0.335	Ambiguity
9			birthdayCost 2	0.335	Ambiguity
10			costLabel 2	0.335	Ambiguity
11			label5 1	0.335	Ambiguity
12			tabPage1 1	0.335	Ambiguity
13			label6 1	0.335	Ambiguity
14			label4 1	0.335	Ambiguity
15			label2 1	0.335	Ambiguity
16			tabPage2 1	0.335	Ambiguity
17			tabControl1 1	0.335	Ambiguity
18			label1 1	0.335	Ambiguity
19			components 1	0.335	Ambiguity
ประเภทร่องรอยไม่ดีที่ต้องการแก้ไข			Ambiguity in Method		
วิธีการรีแฟคทอริ่งสำหรับการแก้ไขร่องรอยไม่ดี					
1. Simplifying Conditional Expression วิธีย่อยที่ใช้ Decompose Conditional 2. Composing Method วิธีย่อยที่ใช้ Extract Method 3. Composing Method วิธีย่อยที่ใช้ Introduce Parameter Object 4. Making Method Calls Simpler วิธีย่อยที่ใช้ Preserve Whole Object 5. Composing Method วิธีย่อยที่ใช้ Replace Method with Method Object 6. Composing Method วิธีย่อยที่ใช้ Replace Temp with Query 7. Composing Method วิธีย่อยที่ใช้ Inline Method 8. Composing Method วิธีย่อยที่ใช้ Substitute algorithm					
วิธีที่เลือกใช้ในการปรับปรุงร่องรอยไม่ดี			Composing Method วิธีย่อยที่ใช้ Extract Method		

ตารางที่ จ.22 แนวทางการปรับปรุงโค้ดกลุ่มโค้ดร่องรอยไม่ดี Party Planner 2 ครั้งที่ 2 (ต่อ)

เหตุผลที่เลือกใช้
ค่าแปลความบ่งชี้ได้ว่าเมท็อด InitializeComponent นั้น มีจำนวนของบรรทัดโค้ด มากจนเกินไป เป็นเหตุให้เกิดความคลุมเครือในระดับของเมท็อดขึ้น ดังนี้ผู้เชี่ยวชาญแนะนำให้แก้ไขด้วยวิธีการ Extract Method จะทำให้เมท็อดนี้มีขนาดเล็กลง เพื่อให้ง่ายต่อการบำรุงรักษา ซึ่งวิธีปฏิบัติ Decompose Conditional ไม่สามารถแก้ไขได้ เพราะเมท็อดนี้ไม่มีการทำงานของนิพจน์เข้ามาเกี่ยวข้อง และไม่มีการรับพารามิเตอร์เข้ามาทำให้วิธีปฏิบัติ Introduce Parameter Object ไม่สามารถแก้ไขได้เช่นกัน ส่วนวิธีปฏิบัติที่เหลือไม่สามารถแก้ไขปัญหาที่เกิดขึ้นโดยตรงกับเมท็อดนี้

ตารางที่ จ.23 ผลลัพธ์หลังจากการปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี Party Planner 2 ครั้งที่ 2

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ	Party Planner 2																												
จำนวนคลาสที่ใช้ทดสอบ	4					จำนวนเมท็อดที่ใช้ทดสอบ					42								จำนวนแอตทริบิวต์ที่ใช้ทดสอบ					30						
	มาตรวัดที่ใช้ในการทดสอบ																												
ชื่อคลาสที่ใช้ทดสอบ	ACML	CC	ECML	ILCC	ILND	MILI	NLOC	NOO	NOP	NOV	ACNPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SACOMF	SECIM	ข้อจำกัด
1) BirthdayParty																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
# Constructor	1	1	5	1	0	34	9	1	3	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_NumberOfPeople	3	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ set_CakeWriting	3	4	5	6	2	60	9	1	1	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ CalculateCostOfDecorations	3	2	7	3	1	44	4	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_CakeWriting	2	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- CalculateCakeSize	2	2	2	3	1	18	3	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_CakeSize	2	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ CalculateCost	1	2	11	3	1	58	5	1	0	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_CostOfFoodPerPerson	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ set_NumberOfPeople	1	1	3	1	0	18	4	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_CostOfDecorations	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ set_CakeSize	1	2	0	2	1	11	2	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ set_CostOfDecorations	1	2	2	2	1	14	2	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ set_CostOfFoodPerPerson	0	2	2	2	1	14	2	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
- cakeWriting 4	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- _costOfDecorations 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- numberOfPeople 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- fancyDecorations 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- _costofFoodPerPerson 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- _cakesize 2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	2	1	1	1	1	6	0	0	14	15							
2) DinnerParty																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
# Constructor	1	1	5	1	0	29	7	1	3	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ CalculateCostOfDecorations	2	2	2	3	1	31	3	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_NumberOfPeople	2	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ set_NumberOfPeople	2	1	1	1	0	10	2	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ set_CostOfDecorations	1	2	2	2	1	14	2	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ CalculateCost	1	2	8	3	1	37	4	1	1	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_CostOfBeveragePerPerson	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_CostOfFoodPerPerson	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_CostOfDecorations	1	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ set_CostOfBeveragePerPerson	1	2	2	2	1	14	2	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ set_CostOfFoodPerPerson	0	2	2	2	1	14	2	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

ตารางที่ จ.23 ผลลัพธ์หลังจากการปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี Party Planner 2 ครั้งที่ 2(ต่อ)

	ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																																						
	- fancyDecorations 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- _costOfDecorations 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- _costofFoodPerPerson 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- numberOfPeople 2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- CostOfBeveragesPerPerson 2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ค่าที่วัดได้												0	1	1	1	1	1	5	0	0	12	0																
3)	Form1																																							
	ชื่อเมท็อดที่ใช้ทดสอบ																																							
	# Constructor	1	1	10	1	0	41	6	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- DisplayBirthdayPartyCost	4	1	4	1	0	20	3	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- DisplayDinnerPartyCost	4	1	4	1	0	16	2	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- InitializeComponent	1	1	5	1	0	17	5	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M1
	- InitializeCom1	1	1	7	1	0	65	20	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M1
	- InitializeCom2	1	1	16	1	0	221	35	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M1
	- InitializeCom3	1	1	11	1	0	235	39	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M1
	- InitializeCom4	1	1	20	1	0	184	31	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M1
	- InitializeCom5	1	1	32	1	0	267	41	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M1
	- fancyBirthday_CheckedChanged	1	1	3	1	0	12	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- cakeWriting_TextChanged	1	1	3	1	0	12	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- numberBirthday_ValueChanged	1	1	4	1	0	13	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- fancybox_CheckedChanged	1	1	3	1	0	12	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- numberUpDown1_ValueChanged	1	1	4	1	0	13	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- healthyBox_CheckedChanged	1	1	3	1	0	12	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ Dispose	0	3	1	4	1	23	3	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																																							
	- birthdayParty 5	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- dinnerParty 5	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- healthyBox 4	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- cakeWriting 4	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- numericUpDown1 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- numberBirthday 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- fancyBox 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- fancyBirthday 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- birthdayCost 2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- costLabel 2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- label5 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- tabPage1 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- label6 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- label4 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- label2 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- tabPage2 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- tabControl1 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- label1 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- components 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ค่าที่วัดได้												0	23	7	1	1	1	19	0	0	16	24																C1
4)	Program																																							
	ชื่อเมท็อดที่ใช้ทดสอบ																																							
	- Main	0	1	4	1	0	10	3	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ค่าที่วัดได้												0	3	1	1	0	0	0	0	0	1	50																
รวมทั้งสิ้น																								0	0	4	1	0	0										

ตารางที่ จ.24 ผลลัพธ์แสดงค่าการแปลความหลังจากการปรับปรุงโค้ดกลุ่มร่องรอยไม่ดี ครั้งที่ 2

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		Party Planner 2			
ชุดรายการ				Defuzzification Output	สรุปค่าแปลความ
เซตที่	คลาส	เมท็อด	แอตทริบิวต์		
1	BirthdayParty	Constructor	cakeWriting 4	0.278	Clean
2		get_NumberOfPeople	_costOfDecorations 3	0.278	Clean
3		set_CakeWriting	numberOfPeople 3	0.29	Clean
4		CalculateCostOfDecorations	fancyDecorations 3	0.278	Clean
5		get_CakeWriting	_costofFoodPerPerson 3	0.278	Clean
6		CalculateCakeSize	_cakesize 2	0.278	Clean
7		get_CakeSize		0.278	Clean
8		CalculateCost		0.287	Clean
9		get_CostOfFoodPerPerson		0.278	Clean
10		set_NumberOfPeople		0.278	Clean
11		get_CostOfDecorations		0.278	Clean
12		set_CakeSize		0.278	Clean
13		set_CostOfDecorations		0.278	Clean
14		set_CostOfFoodPerPerson		0.304	Clean
15	DinnerParty	Constructor	fancyDecorations 3	0.294	Clean
16		CalculateCostOfDecorations	_costOfDecorations 3	0.294	Clean
17		get_NumberOfPeople	_costofFoodPerPerson 3	0.294	Clean
18		set_NumberOfPeople	numberOfPeople 2	0.294	Clean
19		set_CostOfDecorations	CostOfBeveragesPerPers	0.294	Clean
20		CalculateCost		0.294	Clean
21		get_CostOfBeveragePerPerson		0.294	Clean
22		get_CostOfFoodPerPerson		0.294	Clean
23		get_CostOfDecorations		0.294	Clean
24		set_CostOfBeveragePerPerson		0.294	Clean
25		set_CostOfFoodPerPerson		0.319	Clean
26	Form1	Constructor	birthdayParty 5	0.305	Clean
27		DisplayBirthdayPartyCost	dinnerParty 5	0.305	Clean
28		DisplayDinnerPartyCost	healthyBox 4	0.305	Clean
29		InitializeComponent	cakeWriting 4	0.305	Clean
30		InitializeCom1	numericUpDown1 3	0.318	Clean
31		InitializeCom2	numberBirthday 3	0.324	Clean
32		InitializeCom3	fancyBox 3	0.326	Clean
33		InitializeCom4	fancyBirthday 3	0.318	Clean
34		InitializeCom5	birthdayCost 2	0.328	Clean
35		fancyBirthday_CheckedChanged	costLabel 2	0.305	Clean
36		cakeWriting_TextChanged	label5 1	0.305	Clean
37		numberBirthday_ValueChanged	tabPage1 1	0.305	Clean
38		fancybox_CheckedChanged	label6 1	0.305	Clean
39		numberUpDown1_ValueChanged	label4 1	0.305	Clean
40		healthyBox_CheckedChanged	label2 1	0.305	Clean
41		Dispose	tabPage2 1	0.328	Clean
42			tabControl1 1		
43			label1 1		
44			components 1		
45	Program	Main		0.262	Clean

ตารางที่ จ.25 ผลลัพธ์วัดได้เป็นโค้ดที่มีความคลุมเครือ TaxApp

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ	TaxApp																													
จำนวนคลาสที่ใช้ทดสอบ	2					จำนวนเมท็อดที่ใช้ทดสอบ								8								จำนวนแอตทริบิวต์ที่ใช้ทดสอบ								9				
	มาตรวัดที่ใช้ในการทดสอบ																													
ชื่อคลาสที่ใช้ทดสอบ	ACML	CC	ECML	ILCC	ILND	NILI	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAcCOMF	SECJIM	ข้อจำกัด
1) Form1																														
ชื่อเมท็อดที่ใช้ทดสอบ																														
# Constructor	1	1	2	1	0	12	3	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- btnSubmit_Click	1	2	10	2	1	91	7	1	2	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- btnClear_Click	1	1	1	1	0	9	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- InitializeComponent	1	1	35	1	0	384	69	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M1
- getTotal	1	1	2	1	0	12	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- getTxtAmount	0	3	1	4	1	12	2	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Dispose	0	3	1	4	1	23	3	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																														
- lblAmount 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- btnClear 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- btnSubmit 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- lblTax 1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- lblResult 2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- nudTax 2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- components 2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Tax 4	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- txtAmount 5	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												1	3	7	1	1	1	8	0	0	5	8								C1
2) Program																														
ชื่อเมท็อดที่ใช้ทดสอบ																														
- Main	0	1	4	1	0	10	3	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	22	1	1	0	0	0	0	0	1	50								
รวมทั้งสิ้น																							0	0	2	1	0	0		

ตารางที่ จ.26 ผลลัพธ์ที่แสดงค่าการแปลความ TaxApp

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		TaxApp			
	ชุดรายการ			Defuzzification Output	สรุปค่าแปลความ
เซตที่	คลาส	เมท็อด	แอตทริบิวต์		
1	Form1	Constructor	lblAmount 1	0.326	Clean
2		btnSubmit_Click	btnClear 1	0.346	Ambiguous
3		btnClear_Click	btnSubmit 1	0.326	Clean
4		InitializeComponent	lblTax 1	0.35	Ambiguous
5		getTotal	lblResult 2	0.326	Clean
6		getTxtAmount	nudTax 2	0.35	Ambiguous
7		Dispose	components 2	0.35	Ambiguous
8			Tax 4		
9			txtAmount 5		
10	Program	Main		0.287	Clean

ตารางที่ จ.27 แนวทางการปรับปรุงโค้ดกลุ่มโค้ดที่มีความคลุมเครือ TaxApp

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			TaxApp		
ลำดับที่	คลาส	เมท็อด	แอตทริบิวต์	ผลลัพธ์การแปลความ	สรุปค่าแปลความ
1	Form1	btnSubmit_Click	btnClear 1	0.35	Ambiguity
3		InitializeComponent	lblTax 1	0.354	Ambiguity
5		getTxtAmount	nudTax 2	0.354	Ambiguity
6		Dispose	components 2	0.354	Ambiguity
7			Tax 4		
8			txtAmount 5		
ประเภทความคลุมเครือที่ต้องการแก้ไข			Ambiguity in Method		

ตารางที่ จ.27 แนวทางการปรับปรุงโค้ดกลุ่มโค้ดที่มีความคลุมเครือ TaxApp (ต่อ)

วิธีการรีแฟคทอริ่งสำหรับการแก้ไขความคลุมเครือ	
1. Simplifying Conditional Expression วิธีย่อยที่ใช้ Decompose Conditional 2. Composing Method วิธีย่อยที่ใช้ Extract Method 3. Composing Method วิธีย่อยที่ใช้ Introduce Parameter Object 4. Making Method Calls Simpler วิธีย่อยที่ใช้ Preserve Whole Object 5. Composing Method วิธีย่อยที่ใช้ Replace Method with Method Object 6. Composing Method วิธีย่อยที่ใช้ Replace Temp with Query 7. Composing Method วิธีย่อยที่ใช้ Inline Method 8. Composing Method วิธีย่อยที่ใช้ Substitute algorithm	
วิธีที่เลือกใช้ในการปรับปรุงความคลุมเครือ	Composing Method วิธีย่อยที่ใช้ Inline Method
เหตุผลที่เลือกใช้	
คลาสมีความคลุมเครือประเภท Ambiguity in Classes เกิดขึ้น เนื่องจากมีปัญหาเกิดจากโค้ดที่มีการทำงานซ้ำซ้อนกัน ควรรวมกันและแยกให้เป็นเมท็อดเดียว ให้สามารถเรียกใช้งานได้ง่ายขึ้น ดังนั้นควรต้องทำการรวมให้อยู่ในเมท็อดเดียวกัน โดยวิธีการ Inline Method เพื่อแก้ไขเมท็อด getTotal รวมเข้ากับ getTxtAmount ทำให้สามารถใช้ งานจบภายในเมท็อดเดียว จากนั้นให้ทดสอบ อีกครั้ง เพื่อให้แน่ใจว่า การรวมเมท็อดไม่ทำให้เกิดความคลุมเครือแบบอื่นๆ เพิ่มเติม ซึ่งสาเหตุของปัญหานี้ไม่เกี่ยวกับจำนวนของโค้ดที่มีปริมาณมากหรือซ้ำซ้อนของเมท็อด ฉะนั้น วิธีการปฏิบัติอื่นจึงไม่เหมาะสมสำหรับการแก้ไขปัญหานี้	

ตารางที่ จ.28 ผลลัพธ์หลังจากการปรับปรุงโค้ดกลุ่มโค้ดที่มีความคลุมเครือ TaxApp ครั้งที่ 1

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			TaxApp																										
จำนวนคลาสที่ใช้ทดสอบ			2					จำนวนเมท็อดที่ใช้ทดสอบ								6						จำนวนแอตทริบิวต์ที่ใช้ทดสอบ							8			
			มาตรวัดที่ใช้ในการทดสอบ																										
ชื่อคลาสที่ใช้ทดสอบ			ACML	CC	ECML	ILCC	ILND	NILI	NLOC	NOO	NOP	NOV	ACNPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-OMF	SEC-IM	ข้อจำกัด
1)	Form1																													
	ชื่อเมท็อดที่ใช้ทดสอบ																													
	#	Constructor	1	1	2	1	0	12	3	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	-	btnClear_Click	1	1	1	1	0	6	1	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	-	btnSubmit_Click	1	2	7	2	1	44	6	1	2	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	-	InitializeComponent	1	1	35	1	0	384	69	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M1
	+	Dispose	0	3	1	4	1	23	3	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	-	getResultString	1	1	4	1	0	55	1	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	-	getAmount	1	1	2	1	0	9	1	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
	-	btnClear_1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	-	btnSubmit_1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	-	lblAmount_1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	-	lblTax_1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	-	nudTax_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	-	components_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	-	lblResult_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	-	txtAmount_3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ค่าที่วัดได้												0	22	7	1	1	1	8	0	0	7	20							C1
2)	Program																													
	ชื่อเมท็อดที่ใช้ทดสอบ																													
	-	Main	0	1	4	1	0	10	3	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ค่าที่วัดได้												0	3	1	1	0	0	0	0	0	1	50							
รวมทั้งสิ้น																								0	0	2	1	0	0	

ตารางที่ จ.29 ผลลัพธ์แสดงค่าการแปลความหลังจากการปรับปรุงโค้ดกลุ่มโค้ดที่มีความคลุมเครือ TaxApp ครั้งที่ 1

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		TaxApp			
	ชุดรายการ			Defuzzification Output	สรุปค่าแปลความ
เซตที่	คลาส	เมท็อด	แอตทริบิวต์		
1	Form1	Constructor	btnClear_1	0.264	Clean
2		btnClear_Click	btnSubmit_1	0.3	Clean
3		btnSubmit_Click	lblAmount_1	0.3	Clean
4		InitializeComponent	lblTax_1	0.324	Clean
5		Dispose	nudTax_2	0.324	Clean
6		getResultString	components_2	0.306	Clean
7		getAmount	lblResult_2	0.3	Clean
8			txtAmount_3		
9	Program	Main		0.264	Clean

ตารางที่ จ.30 ผลลัพธ์วัดได้เป็นโค้ดที่มีความคลุมเครือ ObjectDumper

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		ObjectDumper																												
จำนวนคลาสที่ใช้ทดสอบ		1					จำนวนเมท็อดที่ใช้ทดสอบ						12						จำนวนแอตทริบิวต์ที่ใช้ทดสอบ						4					
		มาตรวัดที่ใช้ในการทดสอบ																												
ชื่อคลาสที่ใช้ทดสอบ		ACML	CC	ECML	ILCC	ILND	NILI	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	ISS	LOOM1	LOOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-OMF	SEC-IM	ข้อจำกัด
1)	ObjectDumper																													
	ชื่อเมท็อดที่ใช้ทดสอบ																													
	# Constructor	1	1	1	1	0	9	2	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- Write1	4	2	2	2	1	22	3	4	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- WriteObject	2	9	9	16	4	151	22	1	2	5	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- WriteValue	2	6	3	8	1	81	9	1	1	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- WriteLine	2	1	1	1	0	9	2	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- WriteIndent	2	2	1	3	1	21	4	1	0	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- writePropertyMember	1	10	12	15	5	125	15	1	2	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- writeMemberElement	1	11	19	19	4	167	25	1	2	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- WriteTab	1	2	1	3	1	22	3	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ Write2	1	1	2	1	0	13	3	4	3	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ Write3	1	1	2	1	0	7	1	4	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ Write4	0	1	1	1	0	6	1	4	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
	- write_4	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- pos_3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- depth_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- level_2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ค่าที่วัดได้											0	10	1	1	1	1	4	0	0	12	0								
รวมทั้งสิ้น																							0	0	1	1	0	0		

ตารางที่ จ.31 ผลลัพธ์ที่แสดงค่าการแปลความ ObjectDumper

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		ObjectDumper			
	ชุดรายการ			Defuzzification Output	สรุปค่าแปลความ
เซตที่	คลาส	เมท็อด	แอตทริบิวต์		
1	ObjectDumper	Constructor	write_4	0.309	Clean
2		Write1	pos_3	0.309	Clean
3		WriteLine	depth_2	0.309	Clean
4		WriteIndent	level_2	0.309	Clean
5		WriteObject		0.338	Ambiguous
6		WriteTab		0.309	Clean
7		Write2		0.309	Clean
8		Write3		0.309	Clean
9		WriteValue		0.328	Clean
10		Write4		0.335	Ambiguous

ตารางที่ จ.32 แนวทางการปรับปรุงโค้ดกลุ่มโค้ดที่มีความคลุมเครือ ObjectDumper ครั้งที่ 1

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			ObjectDumper		
ลำดับที่	**คลาส**	**เมท็อด**	**แอตทริบิวต์**	**ผลลัพธ์การแปลความ**	**สรุปค่าแปลความ**
1	ObjectDumper	WriteObject	write_4	0.338	Ambiguity
			pos_3		
2		Write4	depth_2	0.335	Ambiguity
			level_2		
ประเภทความคลุมเครือที่ต้องการแก้ไข			Ambiguity in Method		
วิธีการรีแฟคทอริ่งสำหรับการแก้ไขความคลุมเครือ					
1. Simplifying Conditional Expression วิธีย่อยที่ใช้ Decompose Conditional 2. Composing Method วิธีย่อยที่ใช้ Extract Method					

ตารางที่ จ.32 แนวทางการปรับปรุงโค้ดกลุ่มโค้ดที่มีความคลุมเครือ ObjectDumper ครั้งที่ 1(ต่อ)

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			ObjectDumper		
ลำดับที่	**คลาส**	**เมท็อด**	**แอตทริบิวต์**	**ผลลัพธ์การแปลความ**	**สรุปค่าแปลความ**
1	ObjectDumper	WriteObject	write 4	0.338	Ambiguity
			pos 3		
2		Write4	depth 2	0.335	Ambiguity
			level 2		
ประเภทความคลุมเครือที่ต้องการแก้ไข			Ambiguity in Method		
วิธีการรีแฟคทอริ่งสำหรับการแก้ไขความคลุมเครือ					
1. Simplifying Conditional Expression วิธีย่อยที่ใช้ Decompose Conditional 2. Composing Method วิธีย่อยที่ใช้ Extract Method 3. Composing Method วิธีย่อยที่ใช้ Introduce Parameter Object 4. Making Method Calls Simpler วิธีย่อยที่ใช้ Preserve Whole Object 5. Composing Method วิธีย่อยที่ใช้ Replace Method with Method Object 6. Composing Method วิธีย่อยที่ใช้ Replace Temp with Query 7. Composing Method วิธีย่อยที่ใช้ Inline Method 8. Composing Method วิธีย่อยที่ใช้ Substitute algorithm					
วิธีที่เลือกใช้ในการปรับปรุงความคลุมเครือ			Composing Method วิธีย่อยที่ใช้ Extract Method		
เหตุผลที่เลือกใช้					
ความคลุมเครือที่เกิดขึ้นในตัวอย่าง เกิดจากเมท็อดที่มีขนาดใหญ่เกินไปและทำงานมากกว่าหนึ่งอย่าง ทำให้เกิดปัญหาประเภท Ambiguity in Method ในการเรียกใช้งานเมท็อดทั้งสองตัว สามารถแก้ไขปัญหา ด้วยวิธีแยกเมท็อดออกเป็นส่วนย่อ ยๆ โดยใช้วิธี Extract Method ทำให้เมท็อดใหม่ที่เกิดขึ้นนั้นไม่ซ้ำซ้อนและทำงานเพียง 1 อย่าง ช่วยให้การบำรุงรักษาเมท็อดนี้ง่ายขึ้น วิธีการปฏิบัติวิธี Decompose Conditional และ Substitute algorithm เป็นการแก้ไขนิพจน์หรือเงื่อนไขในเมท็อดที่เกิดความซับซ้อน จึงไม่เกี่ยวกับปัญหาที่เกิดขึ้นในเมท็อดนี้และไม่เกี่ยวกับพารามิเตอร์หรือตัวแปร ฉะนั้นวิธีปฏิบัติที่เหลือไม่สามารถแก้ไขปัญหาที่เกิดขึ้นได้เช่นกัน					

ตารางที่ จ.33 ผลลัพธ์หลังจากการปรับปรุงโค้ดกลุ่มโค้ดที่มีความคลุมเครือ ObjectDumper ครั้งที่ 1

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		ObjectDumper																													
จำนวนคลาสที่ใช้ทดสอบ		1					จำนวนเมท็อดที่ใช้ทดสอบ									14					จำนวนแอตทริบิวต์ที่ใช้ทดสอบ								4			
		มาตรวัดที่ใช้ในการทดสอบ																													
ชื่อคลาสที่ใช้ทดสอบ		AOML	CC	ECML	ILOC	ILND	NILI	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	ISS	LOOM1	LOOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-OME	SEC-IM	ข้อจำกัด
1)	ObjectDumper																														
	ชื่อเมท็อดที่ใช้ทดสอบ																														
	# Constructor	1	1	1	1	0	9	2	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- Write1	5	2	2	2	1	22	3	4	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- WriteIndent	3	2	1	3	1	21	4	1	0	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- WriteLine	3	1	1	1	0	9	2	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- WriteObject	3	5	3	7	2	53	6	1	2	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- WriteValue	2	6	3	8	1	81	9	1	1	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- writeMemberElement	1	11	19	19	4	167	25	1	2	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- WriteEnumerableElement	1	5	8	10	3	97	14	1	3	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ Write2	1	1	2	1	0	7	1	4	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- writePropertyMember	1	10	12	15	5	125	15	1	2	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ Write3	1	1	2	1	0	13	3	4	3	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- writeNullElement	1	1	4	1	0	16	4	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- WriteTab	1	2	1	3	1	22	3	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	+ Write4	0	1	1	1	0	6	1	4	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																														
	- write_4	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- depth_3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- level_3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- pos_3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ค่าที่วัดได้											0	10	1	1	1	1	4	0	0	14	0									
รวมทั้งสิ้น																							0	0	1	1	0	0		

ตารางที่ จ.34 ผลลัพธ์แสดงค่าการแปลความหลังจากการปรับปรุงโค้ดกลุ่มโค้ดที่มีความคลุมเครือ ObjectDumper ครั้งที่ 1

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		ObjectDumper			
ชุดรายการ				Defuzzification Output	สรุปค่าแปลความ
เซตที่	คลาส	เมท็อด	แอตทริบิวต์		
1	ObjectDumper	Constructor	write_4	0.309	Clean
2		Write1	depth_3	0.309	Clean
3		WriteIndent	level_3	0.309	Clean
4		WriteLine	pos_3	0.309	Clean
5		WriteObject		0.314	Clean
6		WriteValue		0.329	Clean
7		writeMemberElement		0.335	Ambiguous
8		WriteEnumerableElement		0.332	Clean
9		Write2		0.309	Clean
10		writePropertyMember		0.338	Ambiguous
11		Write3		0.309	Clean
12		writeNullElement		0.309	Clean
13		WriteTab		0.309	Clean
14		Write4		0.309	Clean

ตารางที่ จ.35 แนวทางการปรับปรุงโค้ดกลุ่มโค้ดที่มีความคลุมเครือ ObjectDumper ครั้งที่ 2

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			ObjectDumper		
ลำดับที่	คลาส	เมท็อด	แอตทริบิวต์	ผลลัพธ์การแปลความ	สรุปค่าแปลความ
1	ObjectDumper	writeMemberElement	write_4	0.338	Ambiguity
2		writePropertyMember	pos_3	0.338	Ambiguity
			depth_2		
			level_2		

ตารางที่ จ.35 แนวทางการปรับปรุงโค้ดกลุ่มโค้ดที่มีความคลุมเครือ ObjectDumper ครั้งที่ 2 (ต่อ)

ประเภทความคลุมเครือที่ต้องการแก้ไข	Ambiguity in Class Organization
วิธีการรีแฟคทอริ่งสำหรับการแก้ไขความคลุมเครือ	
1. Dealing with Generalization วิธีย่อยที่ใช้ Extract Interface 2. Dealing with Generalization วิธีย่อยที่ใช้ Collapse Hierarchy 3. Moving Features between Objects วิธีย่อยที่ใช้ Inline Class 4. Dealing with Generalization วิธีย่อยที่ใช้ Seal Classes	
วิธีที่เลือกใช้ในการปรับปรุงความคลุมเครือ	Dealing with Generalization วิธีย่อยที่ใช้ Seal Classes
เหตุผลที่เลือกใช้	
การทำงานคลาส ที่ค่อนข้างยึดติดกับลักษณะการใช้งานแบบนี้ จึง ไม่ควรนำไปใช้กับงานอื่นด้วยการสืบทอด เพราะ โครงสร้างและการออกแบบติดกับการทำงานกับโครงสร้างข้อมูลแบบนี้ ควรจะ Seal Classes เพื่อป้องกันไม่ให้เกิดการสืบทอดคลาสไปใช้งานที่ไม่เหมาะสมต่อไป	

ตารางที่ จ.36 ผลลัพธ์หลังจากการปรับปรุงโค้ดกลุ่มโค้ดที่มีความคลุมเครือ ObjectDumper ครั้งที่ 2

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		ObjectDumper																													
จำนวนคลาสที่ใช้ทดสอบ		1					จำนวนเมท็อดที่ใช้ทดสอบ					14								จำนวนแอตทริบิวต์ที่ใช้ทดสอบ				4								
		มาตรวัดที่ใช้ในการทดสอบ																													
ชื่อคลาสที่ใช้ทดสอบ		AOML	CC	ECML	ILCC	ILND	NILI	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-OMF	SEC-IM	ข้อจำกัด	
1) ObjectDumper																															
ชื่อเมท็อดที่ใช้ทดสอบ																															
#	Constructor	1	1	1	1	0	9	2	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	Write1	5	2	2	2	1	22	3	4	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	WriteIndent	3	2	1	3	1	21	4	1	0	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	WriteLine	3	1	1	1	0	9	2	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	WriteObject	3	5	3	7	2	53	6	1	2	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	WriteValue	2	6	3	8	1	81	9	1	1	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	writeMemberElement	1	11	19	19	4	167	25	1	2	9	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	WriteEnumerableElement	1	5	8	10	3	97	14	1	3	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	Write2	1	1	2	1	0	7	1	4	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	writePropertyMember	1	10	12	15	5	125	15	1	2	8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	Write3	1	1	2	1	0	13	3	4	3	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	writeNullElement	1	1	4	1	0	16	4	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	WriteTab	1	2	1	3	1	22	3	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	Write4	0	1	1	1	0	6	1	4	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																															
-	write_4	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	depth_3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	level_3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	pos_3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	10	1	1	1	1	4	0	0	14	0									
รวมทั้งสิ้น																							0	0	0	1	0	0			

ตารางที่ จ.37 ผลลัพธ์แสดงค่าการแปลความหลังจากการปรับปรุงโค้ดกลุ่มโค้ดที่มีความคลุมเครือ ObjectDumper ครั้งที่ 2

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		ObjectDumper			
	ชุดรายการ			Defuzzification Output	สรุปค่าแปลความ
เซตที่	คลาส	เมท็อด	แอตทริบิวต์		
1	ObjectDumper	Constructor	write 4	0.283	Clean
2		Write1	depth 3	0.283	Clean
3		WriteIndent	level 3	0.283	Clean
4		WriteLine	pos 3	0.283	Clean
5		WriteObject		0.287	Clean
6		WriteValue		0.301	Clean
7		writeMemberElement		0.308	Clean
8		WriteEnumerableElement		0.305	Clean
9		Write2		0.283	Clean
10		writePropertyMember		0.312	Clean
11		Write3		0.283	Clean
12		writeNullElement		0.283	Clean
13		WriteTab		0.283	Clean
14		Write4		0.309	Clean

ตารางที่ จ.38 ผลลัพธ์วัดได้เป็นโค้ดที่มีความคลุมเครือ LINQ to XML

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		LINQ to XML																													
จำนวนคลาสที่ใช้ทดสอบ		1					จำนวนเมท็อดที่ใช้ทดสอบ						9			จำนวนแอตทริบิวต์ที่ใช้ทดสอบ				5											
		มาตรวัดที่ใช้ในการทดสอบ																													
ชื่อคลาสที่ใช้ทดสอบ		AOML	CC	EOML	ILCC	ILND	NILI	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-OMF	SEC-IM	ข้อจำกัด
1)	Program																														
	ชื่อเมท็อดที่ใช้ทดสอบ																														
	# Constructor	0	N/A	1	1	0	3	N/A	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- b_1	1	1	3	1	0	12	1	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- b_4	1	1	4	1	0	15	1	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- b_3	1	1	0	1	0	5	1	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- b_2	1	1	3	1	0	9	1	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- b_0	1	1	4	1	0	23	1	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- GetStarBuzzData	1	1	7	1	0	90	2	1	0	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- CreatePerson	1	1	3	1	0	60	1	1	5	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	- Main	0	4	26	18	2	184	25	1	1	13	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																														
	- 9_CachedAnonymousMethodDelegate8 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	F1
	- 9_CachedAnonymousMethodDelegate7 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	F1
	- 9_CachedAnonymousMethodDelegate9 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	F1
	- 9_CachedAnonymousMethodDelegate5 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	F1
	- 9_CachedAnonymousMethodDelegate6 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	F1
	ค่าที่วัดได้											0	16	1	1	0	0	5	0	0	9	0									
รวมทั้งสิ้น																							0	0	1	1	0	0		

ตารางที่ จ.39 ผลลัพธ์ที่แสดงค่าการแปลความ LINQ to XML

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		LINQ to XML			
	ชุดรายการ			Defuzzification Output	สรุปค่าแปลความ
เซตที่	คลาส	เมท็อด	แอตทริบิวต์		
1	Program	Constructor		0.305	Clean
2		b_1		0.281	Clean
3		b_2		0.281	Clean
4		b_3		0.281	Clean
5		b_4		0.281	Clean
6		b_0		0.281	Clean
7		GetStarBuzzData		0.288	Clean
8		Main		0.344	Ambiguous

ตารางที่ จ.40 แนวทางการปรับปรุงโค้ดกลุ่มโค้ดที่มีความคลุมเครือ LINQ to XML

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ			LINQ to XML		
ลำดับที่	**คลาส**	**เมท็อด**	**แอตทริบิวต์**	**ผลลัพธ์การแปลความ**	**สรุปค่าแปลความ**
1	Program	Main		0.344	Ambiguity
ประเภทความคลุมเครือที่ต้องการแก้ไข			Ambiguity in Method		
วิธีการรีแฟคทอริ่งสำหรับการแก้ไขความคลุมเครือ					
1. Simplifying Conditional Expression วิธีย่อยที่ใช้ Decompose Conditional 2. Composing Method วิธีย่อยที่ใช้ Extract Method 3. Composing Method วิธีย่อยที่ใช้ Introduce Parameter Object 4. Making Method Calls Simpler วิธีย่อยที่ใช้ Preserve Whole Object 5. Composing Method วิธีย่อยที่ใช้ Replace Method with Method Object 6. Composing Method วิธีย่อยที่ใช้ Replace Temp with Query 7. Composing Method วิธีย่อยที่ใช้ Inline Method 8. Composing Method วิธีย่อยที่ใช้ Substitute algorithm					
วิธีที่เลือกใช้ในการปรับปรุงความคลุมเครือ			Composing Method วิธีย่อยที่ใช้ Extract Method		
เหตุผลที่เลือกใช้					
เมท็อดในตัวอย่างแสดงลักษณะความคลุมเครือประเภท Ambiguity in Method ซึ่งเกิดจากขนาดของเมท็อด ที่อยู่ภายในนั้น มีขนาดใหญ่ทำให้แก้ไขปรับปรุงจึงเป็นไปได้ยาก ดังนั้นจึงควรแก้ไขเมท็อดนี้ด้วยวิธี Extract Method เพื่อทำให้เมท็อดมีขนาดเล็กและแบ่งงานที่ซ้ำซ้อนกันให้เป็นสัดส่วนมากขึ้น โดยสาเหตุของปัญหาที่เกิดขึ้นไม่เกี่ยวกับนิพจน์ เงื่อนไข พารามิเตอร์หรือตัวแปรรับค่าในเมท็อด ฉะนั้นวิธีการปฏิบัติวิธีอื่นไม่สามารถแก้ไขปัญหาที่เกิดขึ้นได้					

ตารางที่ จ.41 ผลลัพธ์หลังจากการปรับปรุงโค้ดกลุ่มโค้ดที่มีความคลุมเครือ LINQ to XML ครั้งที่ 1

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ	LINQ to XML																													
จำนวนคลาสที่ใช้ทดสอบ	1					จำนวนเมท็อดที่ใช้ทดสอบ									14					จำนวนแอตทริบิวต์ที่ใช้ทดสอบ								5			
	มาตรวัดที่ใช้ในการทดสอบ																													
ชื่อคลาสที่ใช้ทดสอบ	AOML	CC	EOML	ILCC	ILND	NILI	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-OMF	SEC-IM	ข้อจำกัด	
1) Program																														
ชื่อเมท็อดที่ใช้ทดสอบ																														
# Constructor	0	N/A	1	1	0	3	N/A	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- writeEach	2	2	6	5	2	31	5	1	1	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- b_0	1	1	3	1	0	12	1	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- GetStarBuzzData	1	1	7	1	0	90	2	1	0	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- b_1	1	1	3	1	0	9	1	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- b_2	1	1	0	1	0	5	1	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- b_6	1	1	4	1	0	15	1	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- b_8	1	1	4	1	0	23	1	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- writezipcodeGroup	1	2	16	8	2	67	6	1	1	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- writeStarBuzzData	1	1	6	1	0	18	5	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- CreatePerson	1	1	3	1	0	60	1	1	5	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- writePersonData	1	1	7	2	1	23	3	1	1	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- writeOurBlog	1	1	10	2	1	36	4	1	0	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- Main	0	1	3	1	0	8	3	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																														
- 9_CachedAnonymousMethodDelegate8 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	F1
- 9_CachedAnonymousMethodDelegate7 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	F1
- 9_CachedAnonymousMethodDelegate9 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	F1
- 9_CachedAnonymousMethodDelegate5 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	F1
- 9_CachedAnonymousMethodDelegate6 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	F1
ค่าที่วัดได้											0	16	1	1	0	0	5	0	0	14	0									
รวมทั้งสิ้น																						0	0	1	1	0	0			

ตารางที่ จ.42 ผลลัพธ์แสดงค่าการแปลความหลังจากการปรับปรุงโค้ดกลุ่มโค้ดที่มีความคลุมเครือ LINQ to XML ครั้งที่ 1

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ	LINQ to XML				
	ชุดรายการ			Defuzzification Output	สรุปค่าแปลความ
เซตที่	คลาส	เมท็อด	แอตทริบิวต์		
1	Program	Constructor	9_CachedAnonymousMethodDelegate8 1	0.305	Clean
2		writeEach	9_CachedAnonymousMethodDelegate7 1	0.281	Clean
3		b_0	9_CachedAnonymousMethodDelegate9 1	0.281	Clean
4		GetStarBuzzData	9_CachedAnonymousMethodDelegate5 1	0.299	Clean
5		b_1	9_CachedAnonymousMethodDelegate6 1	0.281	Clean
6		b_2		0.281	Clean
7		b_6		0.281	Clean
8		b_8		0.281	Clean
9		writezipcodeGroup		0.294	Clean
10		writeStarBuzzData		0.281	Clean
11		CreatePerson		0.292	Clean
12		writePersonData		0.281	Clean
13		writeOurBlog		0.281	Clean
14		Main		0.305	Clean

ตารางที่ จ.43 ผลลัพธ์ที่วัดได้เป็นคลีนโค้ด Encryption and Decryption

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ	Encryption and Decryption																												
จำนวนคลาสที่ใช้ทดสอบ	2					จำนวนเมท็อดที่ใช้ทดสอบ					6									จำนวนแอตทริบิวต์ที่ใช้ทดสอบ					3				
	มาตรวัดที่ใช้ในการทดสอบ																												
ชื่อคลาสที่ใช้ทดสอบ	ACML	CC	ECML	ILCC	ILND	NILI	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-OMF	SEC-IM	ข้อจำกัด
1) ArrayConverter																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
+ ConvertToBytes	1	2	2	3	1	36	6	1	1	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้											0	2	1	1	0	0	0	0	0	1	0								
2) CryptoHelper																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
+ SetKeyAndInitialVector	2	1	6	1	0	19	2	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ SetAlgorithm	2	1	3	1	0	13	3	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Key	1	3	4	3	1	38	6	1	1	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Encrypt	0	1	12	3	2	52	8	1	1	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Decrypt	0	1	11	3	2	59	9	1	1	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
- initialVector	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- key	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- algorithm	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้											0	9	3	1	0	0	3	0	0	0	N/A								
รวมทั้งสิ้น																						0	0	1	1	0	0		

ตารางที่ จ.44 ผลลัพธ์ที่แสดงค่าการแปลความ Encryption and Decryption

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		Encryption and Decryption			
ชุดรายการ				Defuzzification Output	สรุปค่าแปลความ
เซตที่	คลาส	เมท็อด	แอตทริบิวต์		
1	ArrayConverter	ConvertToBytes	-	0.242	clean
2	CryptoHelper	SetKeyAndInitialVector	initialVector	0.261	clean
3		SetAlgorithm	key	0.261	clean
4		Key	algorithm	0.261	clean
5		Encrypt		0.268	clean
6		Decrypt		0.282	clean

ตารางที่ จ.45 ผลลัพธ์ที่วัดได้เป็นคลีนโค้ด Playing Card

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ	Playing Card																												
จำนวนคลาสที่ใช้ทดสอบ	3					จำนวนเมท็อดที่ใช้ทดสอบ					10									จำนวนแอตทริบิวต์ที่ใช้ทดสอบ					5				
	มาตรวัดที่ใช้ในการทดสอบ																												
ชื่อคลาสที่ใช้ทดสอบ	ACML	CC	ECML	ILCC	ILND	NILI	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-OMF	SEC-IM	ข้อจำกัด
1) Card																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
# Constructor	1	1	3	1	0	14	3	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ set_Value	1	N/A	0	1	0	4	0	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_Name	1	1	4	1	0	15	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_Suit	1	N/A	0	1	0	6	0	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ set_Suit	1	N/A	0	1	0	4	0	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_Value	1	N/A	0	1	0	6	0	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
- k_BackingField1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- k_BackingField2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้											0	2	1	1	0	0	2	0	0	6	0								
2) Form1																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
# Constructor	1	1	3	1	0	15	4	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- InitializeComponent	1	1	20	1	0	83	16	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M1
- button1_Click	1	1	5	1	0	17	2	1	2	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ Dispose	0	3	1	4	1	23	3	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
- components	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- random	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- button1	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้											0	3	7	1	1	1	3	0	0	4	40								C1
3) Program																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
- Main	0	1	4	1	0	10	3	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้											0	16	1	1	0	0	0	0	0	1	50								
รวมทั้งสิ้น																						0	0	2	1	0	0		

ตารางที่ จ.46 ผลลัพธ์ที่แสดงค่าการแปลความ Playing Card

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		Playing Card			
ชุดรายการ				Defuzzification Output	สรุปค่าแปลความ
เซตที่	คลาส	เมท็อด	แอตทริบิวต์		
1	Card	Constructor	k_BackingField1	0.238	clean
2		set_Value	k_BackingField2	0.238	clean
3		get_Name		0.238	clean
4		get_Suit		0.238	clean
5		set_Suit		0.238	clean
6		get_Value		0.238	clean
7	Form1	Constructor	components	0.215	clean
8		InitializeComponent	random	0.216	clean
9		button1_Click	button1	0.215	clean
10		Dispose		0.216	clean
11	Program	Main		0.241	clean

ตารางที่ จ.47 ผลลัพธ์ที่วัดได้เป็นคลีนโค้ด Let's Build a House

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		Let's Build a House																												
จำนวนคลาสที่ใช้ทดสอบ		7					จำนวนเมท็อดที่ใช้ทดสอบ					25					จำนวนแอตทริบิวต์ที่ใช้ทดสอบ					20								
		มาตรวัดที่ใช้ในการทดสอบ																												
ชื่อคลาสที่ใช้ทดสอบ		ACML	CC	ECML	ILCC	ILND	NILI	NLOC	NOO	NOP	NOV	AC-NPF	NPF	CBO	DIT	ISS	LCOM1	LCOM3	NFD	NOC	NOI	NOM	PCC	DMS	LMFC	NOIM	RC	SAC-OME	SEC-IM	ข้อจำกัด
1) Form1																														
ชื่อเมท็อดที่ใช้ทดสอบ																														
#	Constructor	1	1	4	1	0	17	4	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	MoveToANewLocation	3	3	8	5	1	71	11	1	1	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	goThroughTheDoor_Click	1	1	2	1	0	11	2	1	2	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	goHere_Click	1	1	2	1	0	12	1	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	CreateObjects	1	1	6	1	0	152	16	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	InitializeComponents	1	1	30	1	0	229	44	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	M1
+	Dispose	0	3	1	4	1	23	3	1	1	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																														
-	exits 3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	currentLocation 3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	livingRoom 2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	goThroughTheDoor 2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	description 2	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	frontYard 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	garden 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	backYard 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	kitchen 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	goHere 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	components 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	diningRoom 1	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	11	7	1	1	1	12	0	0	7	24								C1
2) Location																														
ชื่อเมท็อดที่ใช้ทดสอบ																														
#	Constructor	3	3	3	4	2	58	9	1	0	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	get_Description	2	1	1	1	0	9	2	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	get_Name	2	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																														
+	Exits	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
-	Name 3	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												1	23	1	1	0	1	2	4	0	3	0								

ตารางที่ จ.47 ผลลัพธ์ที่วัดได้เป็นคลีนโค้ด Let's Build a House (ต่อ)

3) Outside																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
# Constructor	2	1	1	1	0	10	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_Description	1	2	2	2	1	20	4	1	0	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_Hot	0	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
- hot 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	1	2	1	0	0	1	1	0	3	0							
4) OutsideWithDoor																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
# Constructor	1	1	1	1	0	11	2	1	3	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ set_DoorLocation	1	1	0	1	0	5	1	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_Description	0	1	2	1	0	12	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_DoorDescription	0	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_DoorLocation	0	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
- doorDescription 3	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- doorLocation 2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	3	3	1	1	1	2	0	1	5	0							
5) Program																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
- Main	0	1	4	1	0	10	3	1	0	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	3	1	1	0	0	0	0	0	1	50							
6) Room																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
# Constructor	2	1	1	1	0	10	2	1	2	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_Description	0	1	2	1	0	12	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
- decoration 2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	1	2	1	0	0	1	1	0	2	0							
7) RoomWithDoor																													
ชื่อเมท็อดที่ใช้ทดสอบ																													
# Constructor	1	1	1	1	0	11	2	1	3	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ set_DoorLocation	1	1	0	1	0	5	1	1	1	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_DoorDescription	0	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+ get_DoorLocation	0	1	0	1	0	7	1	1	0	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ชื่อแอตทริบิวต์ที่ใช้ทดสอบ																													
- doorLocation 2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
- doorDescription 2	-	-	-	-	-	-	-	-	-	-	0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
ค่าที่วัดได้												0	3	3	1	1	1	2	0	1	4	0							
รวมทั้งสิ้น																							0	0	4	2	0	0	

ตารางที่ จ.48 ผลลัพธ์ที่แสดงค่าการแปลความ Let's Build a House

ชื่อกลุ่มข้อมูลที่ใช้ทดสอบ		Let's Build a House			
ชุดรายการ				Defuzzification Output	สรุปค่าแปลความ
เซตที่	คลาส	เมท็อด	แอตทริบิวต์		
1	Form1	Constructor	exits 3	0.291	Clean
2		MoveToANewLocation	currentLocation 3	0.305	Clean
3		goThroughTheDoor_Click	livingRoom 2	0.291	Clean
4		goHere_Click	goThroughTheDoor 2	0.291	Clean
5		CreateObjects	description 2	0.309	Clean
6		InitializeComponents	frontYard 1	0.311	Clean
7		Dispose	garden 1	0.314	Clean
8			backYard 1		
9			kitchen 1		
10			goHere 1		
11			components 1		
12			diningRoom 1		
13	Location	Constructor	Exits	0.3	Clean
14		get_Description	Name 3	0.3	Clean
15		get_Name		0.291	Clean
16				0.291	Clean
17				0.291	Clean
18				0.291	Clean
19	Outside	Constructor	hot 3	0.247	Clean
20		get_Description		0.247	Clean
21		get_Hot		0.269	Clean
22	OutsideWithDoor	Constructor	doorDescription 3	0.247	Clean
23		set_DoorLocation	doorLocation 2	0.247	Clean
24		get_Description		0.269	Clean
25		get_DoorDescription		0.269	Clean
26		get_DoorLocation		0.269	Clean
27	Program	Main		0.247	Clean
28	Room	Constructor	decoration 2	0.247	Clean
29		get_Description		0.269	Clean
30	RoomWithDoor	Constructor	doorLocation 2	0.247	Clean
31		set_DoorLocation	doorDescription 2	0.247	Clean
32		get_DoorDescription		0.269	Clean
33		get_DoorLocation		0.269	Clean

ภาคผนวก ฉ
รายการและลักษณะกลุ่มตัวอย่างโค้ดชุดเริ่มต้นและชุดทวนสอบ

รายการนี้เป็นการอธิบายลักษณะกลุ่มตัวอย่างโค้ดสำหรับชุดเริ่มต้นทำการทดสอบและชุดทวนสอบ ประกอบไปด้วย

1) รายการโค้ด คือ เป็นการอธิบายชื่อของกลุ่มตัวอย่างที่ใช้ในการทดสอบ

2) จำนวนคลาส คือ เป็นผลรวมของคลาสที่ถูกทำการทดสอบ

3) จำนวนเมท็อด คือ ผลรวมของเมท็อดที่รวมจากทุกคลาสที่ถูกมาใช้ในการทดสอบ

4) จำนวนแอตทริบิวต์ คือ ผลรวมของฟิลด์ที่ถูกรวมไว้จากทุกคลาสที่ใช้ในการทดสอบ

5) จุดประสงค์การทำงาน คือ การอธิบายกลุ่มตัวอย่างที่ใช้ทดสอบถูกใช้งานหรือเกี่ยวข้องกับเรื่องใด

6) แหล่งที่มา คือ ที่มาของกลุ่มตัวอย่างทดสอบที่ถูกนำมาใช้งานวิทยานิพนธ์

ตารางที่ ฉ.1 รายการกลุ่มโค้ดตัวอย่างชุดเริ่มต้น

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
1	Converting Hexadecimal	2	13	15	ตัวอย่างทดสอบแปลค่าตัวเลขและตัวอักษรเป็นเลขฐานสิบหก	เวปไซต์	http://www.codeproject.com/KB/recipes/hexencoding.aspx
2	DirectoryInfoExtensions	1	2	-	ตัวอย่างทดสอบการตรวจสอบและเรียกใช้ข้อมูลของไฟล์ไดเรกทอรี่	เวปไซต์	http://refactormycode.com/codes/600-directoryinfo-copyto
3	QueueExample1	1	2	2	ตัวอย่างทดสอบการทำงานแอพพลิเคชันแบบคิว	เวปไซต์	http://www.codeproject.com/KB/recipes/Queues_Collection.aspx
4	Assignment1MH_Base	5	44	20	ตัวอย่างทดสอบใบสั่งซื้อสินค้า	เวปไซต์	http://refactormycode.com/codes/1282-firts-time-refactoring
5	EventedListExample	9	54	7	ตัวอย่างทดสอบการรับส่งอีเวนท์	เวปไซต์	http://www.codeproject.com/KB/recipes/EventedList.aspx
6	ImmutableCollections	14	124	47	ตัวอย่างทดสอบการสร้างโครงสร้างแบบคอนเล็กชันในการเรียกใช้งาน	เวปไซต์	http://www.codeproject.com/KB/recipes/persistentdatastructures.aspx

ตารางที่ ฉ.1 รายการกลุ่มโค้ดตัวอย่างชุดเริ่มต้น (ต่อ)

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
7	FeatureEnvy	3	15	5	ตัวอย่างทดสอบการรีแฟคทอริ่ง Feature Envy	เวปไซต์	Codementor
8	inappropriateintimacy	3	12	6	ตัวอย่างทดสอบการรีแฟคทอริ่ง inappropriateintimacy	เวปไซต์	Codementor
9	LAZY_CLASS1	3	8	6	ตัวอย่างทดสอบการรีแฟคทอริ่ง LAZY_CLASS ที่ 1	เวปไซต์	Codementor
10	LAZY_CLASS2	3	6	5	ตัวอย่างทดสอบการรีแฟค ทอริ่ง LAZY_CLASS ที่ 2	เวปไซต์	Codementor
11	LONG_PARAMETER_LISTS	3	25	9	ตัวอย่างทดสอบการรีแฟคตอริ่ง LONG_PARAMETER_LISTS	เวปไซต์	Codementor
12	Party Planner 2	4	26	28	ตัวอย่างทดสอบการวางแผนงานเลี้ยง	หนังสือ	Head First C# : A Learner's Guide to Real-World Programming with Visual C# and .NET O'Reilly Media, May 2010

ตารางที่ ฉ.1 รายการกลุ่มโค้ดตัวอย่างชุดเริ่มต้น (ต่อ)

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
13	PartialClassAddIn	2	18	10	ตัวอย่างทดสอบการเรียกใช้คลาสเพิ่มเติม	หนังสือ	Cshape 3.0
14	Using Delegates	4	8	1	ตัวอย่างทดสอบการเรียกใช้งานแบบดิลิเกต	หนังสือ	Cshape 3.0
15	Hit_the_keys	3	8	12	ตัวอย่างทดสอบเกมส์สุ่มใช้กุญแจ	หนังสือ	Head first
16	List_of_Ducks	5	16	3	ตัวอย่างทดสอบเกมส์เรียกไพ่ในสำรับ	หนังสือ	Head first
17	Sloppy_Joe	3	6	11	ตัวอย่างทดสอบเกมส์สุ่มเมนูอาหาร	หนังสือ	Head first
18	TallGuy	3	9	4	ตัวอย่างทดสอบการกำหนดพฤติกรรมในการทำงานจา กส่วนต่อประสาน 2	หนังสือ	Head first

ตารางที่ ฉ.1 รายการกลุ่มโค้ดตัวอย่างชุดเริ่มต้น (ต่อ)

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
19	SimpleLinqToXml	1	5	-	ตัวอย่างทดสอบการแปลงลิงคิวเป็นเอ็กซ์แอมแอล	หนังสือ	Learning CSharp 3.0
20	Mixed_Messages	4	10	1	ตัวอย่างทดสอบการแสดงข้อความ	หนังสือ	Head first
21	PartialClassInterfaces	2	29	3	ตัวอย่างทดสอบการเรียกใช้ ส่วนต่อประสานในคลาส	หนังสือ	Cshape 3.0
22	TaxApp	2	6	8	ตัวอย่างทดสอบการคำนวณภาษีขายอย่างง่าย	หนังสือ	Cshape 3.0
23	UnhandledThreadException	2	9	2	ตัวอย่างทดสอบข้อเสียเมื่อมีการทำงานผิดพลาด	หนังสือ	Cshape 3.0
24	UnhandledWPFException	2	13	4	ตัวอย่างทดสอบการทำงานแอพพลิเคชันดับบลิวพีเอฟ	หนังสือ	Cshape 3.0

ตารางที่ ฉ.1 รายการกลุ่มโค้ดตัวอย่างชุดเริ่มต้น (ต่อ)

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
25	WebBrowser	2	17	13	ตัวอย่างทดสอบการเรียกใช้เวปเบราเซอร์	หนังสือ	Cookbook
26	Joe_and_Bob	3	11	13	ตัวอย่างทดสอบเกมส์คำนวณเงินฝาก	หนังสือ	Head first
27	Mileage_calculator	2	6	14	ตัวอย่างทดสอบการคำนวนระยะทางจากจุดเริ่มต้นและสิ้นสุด	หนังสือ	Head first
28	Time_to_start_coding	2	8	5	ตัวอย่างทดสอบการเขียนโปรแกรมเบื้องต้น	หนังสือ	Head first
29	BinaryWriter	1	2	-	ตัวอย่างทดสอบการนับไบนารี่	หนังสือ	Head first
30	LINQ to XML	1	8	5	ตัวอย่างทดสอบการแปลข้อมูลแล้วลิงคิวเป็นเอ็กซ์เอ็มแอล	หนังสือ	Learning CSharp 3.0
31	OfficeSample	2	9	3	ตัวอย่างทดสอบการเรียกใช้งานโปรแกรมไมโครซอฟต์ออฟฟิส	หนังสือ	Learning CSharp 3.0

ตารางที่ ฉ.1 รายการกลุ่มโค้ดตัวอย่างชุดเริ่มต้น (ต่อ)

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
32	LinqToNorthwind	7	135	51	ตัวอย่างทดสอบการเชื่อมต่อฐานข้อมูลนอร์ทวินแบบลิงคิว	หนังสือ	Learning CSharp 3.0
33	ObjectDumper	1	10	4	ตัวอย่างทดสอบการจัดเก็บและรวบรวมข้อมูลภายในวัตถุ	หนังสือ	Learning CSharp 3.0
34	PasteXmlAsLinq	2	24	9	ตัวอย่างทดสอบการแปลงค่าเอ็กซ์แอมแอลเป็นลิงคิว	หนังสือ	Learning CSharp 3.0
35	PropertyBag	7	112	33	ตัวอย่างทดสอบการกำหนดคุณลักษณะในแก่วัตถุในกา ร เรียกใช้งาน	หนังสือ	Learning CSharp 3.0
36	Rss	2	14	12	ตัวอย่างทดสอบการเรียกใช้งานอาร์แอลแอล	หนังสือ	Learning CSharp 3.0
37	SimpleLambdas	2	31	20	ตัวอย่างทดสอบการเรียกใช้งาน Lambdas	หนังสือ	Learning CSharp 3.0

ตารางที่ ฉ.1 รายการกลุ่มโค้ดตัวอย่างชุดเริ่มต้น (ต่อ)

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
38	WinFormsDataBinding	3	13	66	ตัวอย่างทดสอบการแปลดาต้าในวินโดส์ฟอร์ม	หนังสือ	Learning CSharp 3.0
39	XMLdoc	1	5	1	ตัวอย่างทดสอบการเรียกใช้งานเอกสารเอ็กซ์แอมแอล	หนังสือ	Cookbook
40	XQuery	1	7	-	ตัวอย่างทดสอบการเรียกใช้งาน XQuery	หนังสือ	Learning CSharp 3.0
41	Encryption and Decryption	2	6	3	ตัวอย่างทดสอบแสดงการเข้ารหัสและถอดรหัสตัวอักษร	เวปไซต์	http://refactormycode.com/codes/334-encryption-and-decryption
42	PrimeGenerator	1	9	2	ตัวอย่างทดสอบการสร้างจำนวนเฉพาะ	หนังสือ	Clean code Robert C. Martin
43	WeekValidator	1	6	1	ตัวอย่างทดสอบการตรวจสอบวันหยุดสุดสัปดาห์	เวปไซต์	http://refactormycode.com/codes/1529-validate-a-week

ตารางที่ ฉ.1 รายการกลุ่มโค้ดตัวอย่างชุดเริ่มต้น (ต่อ)

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
44	Set Collections	8	92	5	ตัวอย่างทดสอบการสร้างต้นแบบของโครงสร้างเซตเพื่อช่วยในการทำงานของแอพพลิเคชัน	เวปไซต์	http://www.codeproject.com/KB/recipes/sets.aspx
45	OperatorOverloading	2	19	6	ตัวอย่างทดสอบการทำงานและการประกาศโอเวอร์โหลด	หนังสือ	Learning CSharp 3.0
46	ProtectedSnflar	5	13	1	ตัวอย่างทดสอบการทำงานแบบโมเดลวิวคอนโทล	เวปไซต์	http://refactormycode.com/codes/1302-how-can-i-srp-princible-in-protected-class-structure
47	Generics_Csharp	4	12	4	ตัวอย่างทดสอบการประกาศ Generics	หนังสือ	Learning CSharp 3.0
48	Pool_Puzzle	2	8	2	ตัวอย่างทดสอบเกมส์พูล	หนังสือ	Head First C# : A Learner's Guide to Real-World Programming with Visual C# and .NET O'Reilly Media, May 2010

ตารางที่ ฉ.1 รายการกลุ่มโค้ดตัวอย่างชุดเริ่มต้น (ต่อ)

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
49	Playing Card	3	10	5	ตัวอย่างทดสอบเกมส์การ์ด	หนังสือ	Head First C# : A Learner's Guide to Real-World Programming with Visual C# and .NET O'Reilly Media, May 2010
50	Secret Ingredients	4	13	8	ตัวอย่างทดสอบส่วนผสมของอาหาร	หนังสือ	Head First C# : A Learner's Guide to Real-World Programming with Visual C# and .NET O'Reilly Media, May 2010
51	Fingers the Clown	4	11	5	ตัวอย่างทดสอบการกำหนดพฤติกรรมในการทำงานจาก ส่วนต่อประสาน	หนังสือ	Head First C# : A Learner's Guide to Real-World Programming with Visual C# and .NET O'Reilly Media, May 2010
52	PlanetMission	5	13	7	ตัวอย่างทดสอบการคำนว I ระยะทางระหว่างดาวเคราะห์	หนังสือ	Head First C# : A Learner's Guide to Real-World Programming with Visual C# and .NET O'Reilly Media, May 2010
53	Baseball	6	20	12	ตัวอย่างทดสอบเกมส์เบสบอล	หนังสือ	Head First C# : A Learner's Guide to Real-World Programming with Visual C# and .NET O'Reilly Media, May 2010

ตารางที่ ฉ.1 รายการกลุ่มโค้ดตัวอย่างชุดเริ่มต้น (ต่อ)

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
54	Go Fish	8	55	24	ตัวอย่างทดสอบเกมส์ตกปลา	หนังสือ	Head First C# : A Learner's Guide to Real-World Programming with Visual C# and .NET O'Reilly Media, May 2010
55	JewelThief	6	14	5	ตัวอย่างทดสอบเกมส์ขโมยเพชร	หนังสือ	Head First C# : A Learner's Guide to Real-World Programming with Visual C# and .NET O'Reilly Media, May 2010
56	Let's Build a House	7	25	20	ตัวอย่างทดสอบการวางแผนสร้างบ้าน	หนังสือ	Head First C# : A Learner's Guide to Real-World Programming with Visual C# and .NET O'Reilly Media, May 2010
57	TravellingSalesmanProblem	2	12	6	ตัวอย่างทดสอบการคำนวณระยะทางการเดินทางของเซลแมน	เวปไซต์	http://www.codeproject.com/KB/recipes/simulatedAnnealingTSP.aspx
58	ADO	1	2	-	ตัวอย่างการทดสอบแสดงผลการเรียกใช้งาน	เวปไซต์	http://www.codeproject.com/KB/recipes/simulatedAnnealingTSP.aspx
59	MutexFun	2	12	7	ตัวอย่างทดสอบการเรียกใช้หน่วยความจำร่วมกัน	หนังสือ	Cshape 3.0

ตารางที่ ฉ.1 รายการกลุ่มโค้ดตัวอย่างชุดเริ่มต้น (ต่อ)

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
60	NamedPipes	1	2	-	ตัวอย่างทดสอบวิธีกำหนดชื่อของไฟล์สำหรับการทำงาน	หนังสือ	Cshape 3.0

ตารางที่ ฉ.2 รายการกลุ่มโค้ดตัวอย่างชุดทวนสอบ

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
1	Contacts	7	129	51	ตัวอย่างทดสอบการเก็บข้อมูลบุคคล	หนังสือ	Cookbook
2	AnonymousDelegates	2	6	4	ตัวอย่างทดสอบการเรียกใช้งานแบบดิลิเกต2	หนังสือ	Learning CSharp 3.0
3	Attributes	5	14	3	ตัวอย่างทดสอบกำหนดแอตทริบิวต์	หนังสือ	Learning CSharp 3.0
4	ConditionalMethods	2	4	-	ตัวอย่างทดสอบการสร้างเมท็อดที่ใช้เปรียบเทียบ	หนังสือ	Learning CSharp 3.0

ตารางที่ ฉ.2 รายการกลุ่มโค้ดตัวอย่างชุดทวนสอบ (ต่อ)

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
5	Delegates	5	15	5	ตัวอย่างทดสอบการเรียกใช้งานแบบดิลิเกต	หนังสือ	Learning CSharp 3.0
6	EmployeeTracker.Common	3	17	3	ตัวอย่างทดสอบส่วนกลางของระบบพนักงาน	หนังสือ	Learning CSharp 3.0
7	EmployeeTracker.Fakes	3	30	7	ตัวอย่างทดสอบระบบชั่วคราวของระบบพนักงาน	หนังสือ	Learning CSharp 3.0
8	Tokens	1	3	1	ตัวอย่างทดสอบการรับโทคเคน	หนังสือ	Learning CSharp 3.0
9	Owner drawn text table control	7	75	36	ตัวอย่างทดสอบวิธีการแสดงข้อความในตารางอเนกประสงค์	หนังสือ	http://www.codeproject.com/KB/grid/BTable.aspx
10	Unsafe	3	14	3	ตัวอย่างทดสอบการเรียกใช้งานพอยเตอร์	หนังสือ	Learning CSharp 3.0
11	Reflector	2	62	2	ตัวอย่างทดสอบการทำงานแบบรีเฟลคเตอร์	หนังสือ	Learning CSharp 3.0

ตารางที่ ฉ.2 รายการกลุ่มโค้ดตัวอย่างชุดทวนสอบ (ต่อ)

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
12	Swapping_elephants	3	12	9	ตัวอย่างทดสอบการเรียกใช้งานคลาสที่อยู่ภายใน	หนังสือ	Head first
13	CommandLine	1	2	-	ตัวอย่างทดสอบการสั่งการโดยคอมมานด์	หนังสือ	Learning CSharp 3.0
14	EmployeeTracker.Model	5	69	31	ตัวอย่างทดสอบระบบจัดเก็บข้อมูลของระบบพนักงาน	หนังสือ	Learning CSharp 3.0
15	OleDbSample	1	2	-	ตัวอย่างทดสอบการเชื่อมต่อฐานข้อมูลด้วยโอเลย์	หนังสือ	Learning CSharp 3.0
16	Properties	6	20	7	ตัวอย่างทดสอบการประกาศคุณสมบัติของคลาส	หนังสือ	Learning CSharp 3.0
17	Structs	1	4	-	ตัวอย่างทดสอบการประกาศตัวแปรแบบโครงสร้าง	หนังสือ	Learning CSharp 3.0
18	Threading	8	23	11	ตัวอย่างทดสอบการเรียกใช้งานแบบทะลุผ่าน	หนังสือ	Learning CSharp 3.0

ตารางที่ ฉ.2 รายการกลุ่มโค้ดตัวอย่างชุดทวนสอบ (ต่อ)

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
19	Business Days	1	7	-	ตัวอย่างทดสอบการบันทึกไดอารี่	เวปไซต์	http://www.codeproject.com/KB/recipes/TrinaryTree.aspx
20	Familytree	4	37	15	ตัวอย่างทดสอบการสร้างข้อมูลเครือญาติ	เวปไซต์	http://www.codeproject.com/KB/recipes/TrinaryTree.aspx
21	Breakfast for Lumberjacks	3	13	18	ตัวอย่างการทดสอบเกมส์ทำอาหาร	เวปไซต์	http://www.codeproject.com/KB/recipes/TrinaryTree.aspx
22	Beehive Simulator	6	54	55	ตัวอย่างทดสอบการจำลองการทำงานของรังผึ้ง	หนังสือ	Head First C# : A Learner's Guide to Real-World Programming with Visual C# and .NET O'Reilly Media, May 2010
23	ClassLeaf	3	11	14	ตัวอย่างทดสอบการสร้างและเรียกใช้งานคลาสลูก	เวปไซต์	http://www.codeproject.com/KB/recipes/TrinaryTree.aspx
24	COMInteropPart1	1	8	1	ตัวอย่างทดสอบการเชื่อมต่อกับคอมโพเนนท์	หนังสือ	Learning CSharp 3.0

ตารางที่ ฉ.2 รายการกลุ่มโค้ดตัวอย่างชุดทวนสอบ (ต่อ)

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
25	EmployeeTracker.Employee	1	12	3	ตัวอย่างทดสอบส่วนเก็บรายละเอียดของระบบพนักงาน	หนังสือ	Learning CSharp 3.0
26	Generics2	5	17	6	ตัวอย่างทดสอบการเรียกใช้งานทั่วไป	หนังสือ	Learning CSharp 3.0
27	Pinvoke	1	12	-	ตัวอย่างทดสอบการเรียกใช้งานฟังก์ชันในดีแอลแอล	หนังสือ	Learning CSharp 3.0
28	Security	1	3	-	ตัวอย่างทดสอบการกำหนดสิทธิการอ่านไฟล์	หนังสือ	Learning CSharp 3.0
29	LinqToXmlDataBinding	2	15	46	ตัวอย่างทดสอบการแปลข้อมูลจากลิงคิวไปสู่เอ็กซ์เอ็มแอล	หนังสือ	Learning CSharp 3.0
30	SimpleLinqToObjects	1	3	1	ตัวอย่างทดสอบการแปลงลิงคิวเป็นวัตถุ	หนังสือ	Learning CSharp 3.0
31	Animations	2	6	3	ตัวอย่างทดสอบการทำคอนโทรลให้มีการภาพเคลื่อนไหว	หนังสือ	Cshape 3.0

ตารางที่ ฉ.2 รายการกลุ่มโค้ดตัวอย่างชุดทวนสอบ (ต่อ)

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
32	ConsoleTCPServer	2	10	5	ตัวอย่างทดสอบการเชื่อมต่อกับเซฟเวอร์ด้วยโปรโตคอลทีซีพี	หนังสือ	Cshape 3.0
33	Custom_Icomparer	3	14	3	ตัวอย่างทดสอบการสร้างการเปรียบเทียบด้วยส่วนต่อประสานไอคอมแพเรอร์	หนังสือ	Cshape 3.0
34	FrmFileCopier	3	21	12	ตัวอย่างทดสอบการสร้างแอพพลิเคชันสำเนาไฟล์	หนังสือ	Cshape 3.0
35	CableBill	3	10	10	ตัวอย่างทดสอบการจำลองการคิดค่าใช้จ่ายและส่วนลด	หนังสือ	Cookbook
36	Familiar_math_symbols	2	5	2	ตัวอย่างทดสอบเกมส์เดาค่าตัวเลข	หนังสือ	Head first
37	Talker_Tester	3	7	6	ตัวอย่างทดสอบการส่งข้อความโต้ตอบระหว่างกัน	หนังสือ	Head first
38	Two_Decks	5	31	4	ตัวอย่างทดสอบเกมส์เรียกใช้หรือค้นหาไพ่ในไพ่สองสำรับ	หนังสือ	Head first

ตารางที่ ฉ.2 รายการกลุ่มโค้ดตัวอย่างชุดทวนสอบ (ต่อ)

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
39	BeeControl	3	8	5	ตัวอย่างทดสอบเกมส์ควบคุมภาพเคลื่อนไหวของผึ้ง	หนังสือ	Head first
40	DataContractSerializer	2	10	5	ตัวอย่างทดสอบการสร้างและจัดการซีเรียไลซ์เซอร์	หนังสือ	Head first
41	Equality	4	19	3	ตัวอย่างทดสอบการเปรียบเทียบค่าและตำแหน่งของวัตถุ	หนังสือ	Head first
42	ExcuseManager	3	26	24	ตัวอย่างทดสอบเรียกสั่งงานไฟล์ต่างๆ	หนังสือ	Head first
43	optional_parameters	2	10	5	ตัวอย่างทดสอบการกำหนดพารามิเตอร์เสริม	หนังสือ	Head first
44	Simple_Text_Editor	2	6	9	ตัวอย่างทดสอบการสร้างแอพพลิเคชันสำหรับกระดาษข้อความ	หนังสือ	Cookbook

ตารางที่ ฉ.2 รายการกลุ่มโค้ดตัวอย่างชุดทวนสอบ (ต่อ)

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
45	Whack_a_mole	3	18	15	ตัวอย่างทดสอบเกมส์เล่นตีตัวตุ่น	หนังสือ	Cookbook
46	DeclareArraySample	1	2	-	ตัวอย่างทดสอบการประกาศอาเรย์	หนังสือ	Cookbook
47	Events	3	12	2	ตัวอย่างทดสอบงานแสดงต่างๆ	หนังสือ	Learning CSharp 3.0
48	ExplicitInterface	1	4	2	ตัวอย่างทดสอบการกำหนด ส่วนต่อประสาน เพื่อเรียกใช้จากภายนอก	หนังสือ	Learning CSharp 3.0
49	Libraries	3	6	-	ตัวอย่างทดสอบการเรียกใช้งานฟังก์ชัน	หนังสือ	Learning CSharp 3.0
50	NamedAndOptional	1	3	-	ตัวอย่างทดสอบการกำหนดชื่อในการเรียกใช้งานอย่างสั้น	หนังสือ	Learning CSharp 3.0

ตารางที่ ฉ.2 รายการกลุ่มโค้ดตัวอย่างชุดทวนสอบ (ต่อ)

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
51	Nullable	3	9	-	ตัวอย่างทดสอบการประกาศค่าว่างจากการเรียกใช้งาน	หนังสือ	Learning CSharp 3.0
52	PartialTypes	2	7	-	ตัวอย่างทดสอบการกำหนดชนิดข้อมูลบางส่วน	หนังสือ	Learning CSharp 3.0
53	SimpleVariance	3	6	2	ตัวอย่างทดสอบการประกาศตัวแปรแบบแวร์เลียน	หนังสือ	Cookbook
54	UserConversions	3	17	3	ตัวอย่างทดสอบการประกาศแปลงค่าต่างๆ	หนังสือ	Learning CSharp 3.0
55	Versioning	2	9	-	ตัวอย่างทดสอบการกำหนดเวอร์ชันเพื่อเรียกใช้งาน	หนังสือ	Cookbook
56	Yield	3	14	6	ตัวอย่างทดสอบแสดงการใช้งานของส่วนต่อประสาน	หนังสือ	Cookbook

ตารางที่ ฉ.2 รายการกลุ่มโค้ดตัวอย่างชุดทวนสอบ (ต่อ)

ลำดับ	รายการโค้ด	จำนวนคลาส	จำนวนเมท็อด	จำนวนแอตทริบิวต์	จุดประสงค์การทำงาน	แหล่งที่มา	แหล่งอ้างอิง
57	Image resizing	3	7	2	ตัวอย่างทดสอบการย่อขนาดภาพ	หนังสือ	Head First C# : A Learner's Guide to Real-World Programming with Visual C# and .NET O'Reilly Media, May 2010
58	FlashyThing	2	5	2	ตัวอย่างทดสอบเกมส์สลับหน้าจอ	หนังสือ	Head first
59	StartingPoint	1	4	3	ตัวอย่างทดสอบการรับข้อมูลจากคีย์บอร์ด	หนังสือ	Cshape 3.0
60	ImageMapConverter	1	2	1	ตัวอย่างทดสอบ ความสามารถการแปลงรูปภาพ	หนังสือ	Cshape 3.0

ประวัติผู้เขียนวิทยานิพนธ์

นายพรชัย เลิศหทัยรัตน์ เกิดเมื่อวันที่ 22 พฤศจิกายน พ.ศ. 2525 สำเร็จการศึกษาระดับปริญญาวิทยาศาสตรบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์ จาก คณะวิทยา ศาสตร์ มหาวิทยาลัย มหิดล ในปีการศึกษา 2547 และเข้าศึกษาต่อในหลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์ ภาควิช าวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2552

ขณะที่ศึกษานั้น ผู้วิจัยได้ร่วมทำบทความกับอาจารย์ที่ปรึกษา ซึ่งมีบทความที่ได้รับการคัดเลือกเพื่อนำเสนอและตีพิมพ์ในงานประชุมวิชาการทั้งระดับชาติและนานาชาติ รวมทั้งสิ้น 2 บทความ โดยมีรายละเอียดดังต่อไปนี้

1) บทความวิชาการเรื่อง An approach for Source Code Classification Using Software Metrics and Fuzzy Logic to improve Code Quality with Refactoring techniques ซึ่งได้รับการคัดเลือกเพื่อนำเสนอและตีพิมพ์ในงาน " The 2nd International Conference on Software Engineering and Computer Systems(ICSECS2011)" ระหว่างวันที่ 27 - 29 มิถุนายน 2554 ณ มหาวิทยาลัยมาเลเซียปะหัง รัฐปะหัง ประเทศมาเลเซีย

2) บทความวิชาการเรื่อง “An approach for source code classification to enhance maintainability” ซึ่งได้รับการคัดเลือกเพื่อนำเสนอและตีพิมพ์ในงาน “การประชุมวิชาการร่วมสาขาวิทยาการคอมพิวเตอร์และวิศวกรรมซอฟต์แวร์ ครั้งที่ 8 (The 8th International Joint Conference on Computer Science and Software Engineering: JCSSE 2011) ” ระหว่างวันที่ 11 - 13 พฤษภาคม 2554 ณ คณะเทคโนโลยีสารสนเทศและการสื่อสาร มหาวิทยาลัยมหิดล วิทยาเขตศาลายา นครปฐม ประเทศไทย