การเปรียบเทียบคุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริง ด้วยจำนวนโค้ดโคลนที่แตกต่างกัน

นางสาวธนาภรณ์ กังพานิชกุล

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาการพัฒนาซอฟต์แวร์ด้านธุรกิจ ภาควิชาสถิติ
คณะพาณิชยศาสตร์และการบัญชี จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2556

A COMPARISON OF SOFTWARE QUALITY FROM ASPECT REFACTORING WITH DIFFERENT AMOUNT OF CODE CLONE

Miss Thanaporn Kungpanichkul

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science Program in Business Software Development
Department of Statistics
Faculty of Commerce and Accountancy
Chulalongkorn University
Academic Year 2013

หัวข้อวิทยานิพนธ์	การเปรียบเทียบคุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริงด้วยจำนวนโค้ดโคลนที่แตกต่างกัน
โดย	นางสาวธนาภรณ์ กังพานิชกุล
สาขาวิชา	การพัฒนาซอฟต์แวร์ด้านธุรกิจ
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร.อัษฎาพร ทรัพย์สมบูรณ์

คณะพาณิชยศาสตร์และการบัญชี จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะพาณิชยศาสตร์และการบัญชี
(รองศาสตราจารย์ ดร.พสุ เดชะรินทร์)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.สมจารี ปรียานนท์)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(รองศาสตราจารย์ ดร.อัษฎาพร ทรัพย์สมบูรณ์)

.....กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.จันทร์เจ้า มงคลนาวิน)

.....กรรมการภายนอกมหาวิทยาลัย
(ผู้ช่วยศาสตราจารย์ ดร.วรลักษณ์ วงศ์โดยหวัง ศิริเจริญ)

ธนาภรณ์ กังพานิชกุล : การเปรียบเทียบคุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริงด้วยจำนวนโค้ดโคลนที่แตกต่างกัน. (A COMPARISON OF SOFTWARE QUALITY FROM ASPECT REFACTORING WITH DIFFERENT AMOUNT OF CODE CLONE) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: รศ. ดร.อัษฎาพร ทรัพย์สมบูรณ์, 204 หน้า.

การโปรแกรมเชิงแอสเป็กถูกเสนอเพื่อแก้ไขปัญหาโค้ดกระจัดกระจายและโค้ดพันกันในซอร์สโค้ดของซอฟต์แวร์ สามารถส่งผลให้คุณภาพซอฟต์แวร์ดีขึ้น งานวิจัยนี้จึงสนใจศึกษาแนวทางการใช้การโปรแกรมเชิงแอสเป็กสำหรับปรับปรุงซอฟต์แวร์เชิงวัตถุด้วยการทำแอสเป็กรีแฟคทอริง โดยมุ่งสนใจปัญหาโค้ดกระจัดกระจายที่ทำให้เกิดโค้ดซ้ำกันภายในหลายคลาส จึงเลือกจัดการเมท็อดคอลที่ซ้ำกันในหลายคลาสด้วยแอสเป็ก และพิจารณาผลจากการใช้แอสเป็กสำหรับจัดการเมท็อดคอลในจำนวนซ้ำของเมท็อดคอลที่แตกต่างกัน โดยประเมินคุณภาพของซอฟต์แวร์หลังทำแอสเป็กรีแฟคทอริง จากการรวบรวมค่ามาตรวัดของมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็ก ผลมาตรวัดเชิงวัตถุพบว่าจากมาตรวัดซีเคทั้งหมด 6 มาตรวัด มีเพียง 3 มาตรวัดที่ได้รับผลกระทบจากการทำแอสเป็กรีแฟคทอริง ได้แก่ มาตรวัดคัพพลิงบีทวีนอ็อบเจกต์คลาส มาตรวัดเรสปอนซ์ฟอร์อะคลาส และมาตรวัดแลคออฟโคฮีชันอินเมท็อด โดยมาตรวัดเรสปอนซ์ฟอร์อะคลาส แสดงให้เห็นว่าคุณภาพซอฟต์แวร์ปรับปรุงดีขึ้น และปัจจัยที่ส่งผลต่อมาตรวัดเรสปอนซ์ฟอร์อะคลาส ได้แก่ จำนวนซ้ำของเมท็อดคอลและจำนวนคลาสที่สามารถจัดการเมท็อดคอลทั้งหมด ขณะที่มาตรวัดเชิงแอสเป็กทั้งหมด 10 มาตรวัด มีทั้งหมด 7 มาตรวัดที่ได้รับผลกระทบจากการทำแอสเป็กรีแฟคทอริง แต่ปัจจัยที่ส่งผลให้ค่ามาตรวัดเปลี่ยนแปลงไม่ได้เกิดจากจำนวนซ้ำของเมท็อดคอลที่จัดการด้วยแอสเป็ก หากเป็นปัจจัยอื่นๆแตกต่างกันไปในแต่ละมาตรวัดของมาตรวัดเชิงแอสเป็ก เช่น จำนวนแอดไวซ์ที่อิมพลีเมนต์ภายในแอสเป็ก จำนวนแอดไวซ์ที่เข้ามาขัดขวางการทำงานของคลาส จำนวนคลาสที่แอสเป็กสามารถเข้าไปขัดขวางการทำงาน เป็นต้น นอกจากนี้ยังพบว่าทั้งมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็กแสดงให้เห็นว่าลักษณะของเมท็อดคอลที่จัดการด้วยแอสเป็กสามารถส่งผลต่อมาตรวัดที่เกี่ยวข้องกับคลัพพลิงและปรับปรุงคุณภาพซอฟต์แวร์ให้ดียิ่งขึ้น

ภาควิชา สถิติ ลายมือชื่อนิสิต .....

สาขาวิชา การพัฒนาซอฟต์แวร์ด้านธุรกิจ ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก .....

ปีการศึกษา 2556

5381801326 : MAJOR BUSINESS SOFTWARE DEVELOPMENT
KEYWORDS: ASPECT-ORIENTED PROGRAMMING / ASPECT REFACTORING /
SOFTWARE QUALITY / OBJECT-ORIENTED METRICS / ASPECT-ORIENTED METRICS

THANAPORN KUNGPANICHKUL: A COMPARISON OF SOFTWARE QUALITY FROM ASPECT REFACTORING WITH DIFFERENT AMOUNT OF CODE CLONE. ADVISOR: ASSOC. PROF. ASSADAPORN SAPSOMBOON, 204 pp.

Aspect-oriented programming is an approach to solve code scattering and code tangling which help improve software quality. This research was to find a guideline in using Aspect-oriented programming to improve object-oriented software using aspect refactoring process with a focus on code scattering problem with duplicated code in several classes. The aspect refactoring process used in this research covers extracting different number of duplicated method calls in various classes into aspects resulting in several versions of software. CK Object-oriented metrics and Aspect-oriented metrics (AO) were used as software metrics on software versions applying aspect refactoring. The result showed that only three out of six CK metrics, Coupling Between Object classes (CBO), Response For a Class (RFC) and Lack of Cohesion in Methods (LCOM), were affected from aspect refactoring. Only RFC metric showed that software quality was improved as the number of duplicated codes grows or the number of classes affected by aspect refactoring grows. Considering AO metric, only seven out of ten AO metrics were affected from aspect refactoring. However, it cannot be concluded that the number of duplicated method calls affect the AO metrics. The number of advices implemented in aspect, the number of advices interrupt in classes and the number of classes interrupted by aspects were factors affect the AO metrics. In addition, the metric from OO and AO metrics concerning coupling from method calls indicate an improved software quality from aspect refactoring.

Department:	Statistics	Student's Signature	.....
Field of Study:	Business Software Development	Advisor's Signature	.....
Academic Year:	2013		

กิตติกรรมประกาศ

วิทยานิพนธ์นี้จะสำเร็จลุล่วงและสมบูรณ์ไปได้ด้วยดี ผู้วิจัยต้องขอกราบขอบพระคุณรองศาสตราจารย์ ดร.อัษฎาพร ทรัพย์สมบูรณ์ อาจารย์ที่ปรึกษาวิทยานิพนธ์เป็นอย่างยิ่งที่ได้สละเวลาให้คำปรึกษาและคำแนะนำต่างๆ ตลอดจนแนวทางในการทำงานวิจัยรวมถึงการตรวจแก้วิทยานิพนธ์ฉบับนี้ให้จนเสร็จสมบูรณ์ และขอขอบพระคุณผู้ช่วยศาสตราจารย์ ดร.สมจารี ปรียานนท์ ประธานกรรมการวิทยานิพนธ์ และผู้ช่วยศาสตราจารย์ ดร.จันทร์เจ้า มงคลนาวิน กรรมการวิทยานิพนธ์ ที่กรุณาให้คำแนะนำและแก้ไขรูปแบบงานวิจัย รวมถึงการปรับปรุงเนื้อหาของวิทยานิพนธ์ฉบับนี้ให้สมบูรณ์ และขอขอบพระคุณผู้ช่วยศาสตราจารย์ ดร.วรลักษณ์ วงศ์โดยหวัง ศิริเจริญ กรรมการภายนอกมหาวิทยาลัย ที่ช่วยชี้แนะพร้อมให้คำแนะนำจนงานวิจัยลุล่วงไปได้ด้วยดี

ขอบคุณเพื่อนๆ ที่ให้คำปรึกษาและให้กำลังใจในการทำวิทยานิพนธ์ และสุดท้ายผู้วิจัยกราบขอบพระคุณคุณพ่อ คุณแม่ และพี่สาวที่เข้าใจและคอยมอบกำลังใจเสมอมา ทำให้ผู้วิจัยสามารถทำวิทยานิพนธ์ฉบับนี้ได้สำเร็จลุล่วง

สารบัญ

หน้า

หน้า

สารบัญตาราง

หน้า

หน้า

หน้า

หน้า

หน้า

หน้า

สารบัญภาพ

หน้า

หน้า

หน้า

หน้า

หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

คุณภาพซอฟต์แวร์ (Software quality) เป็นเรื่องสำคัญที่ต้องคำนึงถึงในการพัฒนาซอฟต์แวร์ ไอเอสโอ/ไออีซี 9126 (ISO/IEC 9126) เป็นมาตรฐานของคุณภาพซอฟต์แวร์ได้นำเสนอแบบจำลองคุณภาพซอฟต์แวร์ที่ประกอบด้วย 6 ลักษณะเฉพาะ ได้แก่ ความสามารถเชิงฟังก์ชัน (Functionality) ความเชื่อถือได้ (Reliability) ความสามารถการใช้งานง่าย (Usability) ประสิทธิภาพ (Efficiency) ความสามารถการบำรุงรักษา (Maintainability) และความสามารถในการเคลื่อนย้าย (Portability) ปัจจุบันซอฟต์แวร์มีขนาดใหญ่และมีความซับซ้อนมาก รวมถึงการแข่งขันทางการตลาดที่สูงทำให้เกิดการเปลี่ยนแปลงความต้องการ (Requirement) ของซอฟต์แวร์บ่อยครั้ง ดังนั้นซอฟต์แวร์ที่มีคุณภาพดีกว่าจะสามารถปรับปรุงให้ตรงกับความต้องการได้ง่าย (Kumar, Grover and Kumar, 2009) เพื่อเพิ่มคุณภาพซอฟต์แวร์จึงมีผู้คิดค้นและนำเสนอแนวทางสำหรับพัฒนาซอฟต์แวร์ด้วยการโปรแกรมเชิงวัตถุ (Object-oriented programming) โดยใช้แนวคิดของอ็อบเจกต์ แต่การพัฒนาด้วยการโปรแกรมเชิงวัตถุยังทำให้เกิดปัญหาโค้ดกระจัดกระจาย (Code scattering) คือการเกิดโค้ดซ้ำกันในหลายคลาส และปัญหาโค้ดพันกัน (Code tangling) คือการทำงานของโค้ดที่เกี่ยวข้องกับหลายคอนเซิร์นรวมกันอยู่ในคลาสเดียว การเกิดโค้ดซ้ำกันในหลายคลาสหรือโค้ดโคลน (Code clone) ภายในซอร์สโค้ดของซอฟต์แวร์เกิดจากการอิมพลีเม้นต์ของฟังก์ชันหนึ่งซึ่งเป็นฟังก์ชันสำคัญที่จำเป็นต้องมีในหลายคลาส เช่น การบันทึกล็อก (Logging) การอนุญาตและการพิสูจน์ตัวตน (Authorization and Authentication) และการตรวจหาและจัดการข้อยกเว้น (Exception detection and handling) เป็นต้น ฟังก์ชันเหล่านี้ที่ทำให้เกิดโค้ดโคลนเรียกว่า ครอสคัทติ้งคอนเซิร์น (Crosscutting concern) (Ahuja and Goel, 2008; Kapser, 2009; Laddad, 2003b) ปัญหาดังกล่าวจะส่งผลทำให้ซอฟต์แวร์ยากต่อการทำความเข้าใจ การบำรุงรักษาและการนำกลับมาใช้ใหม่

Kiczales และคณะ (1997) จึงนำเสนอการโปรแกรมเชิงแอสเป็ก (Aspect-oriented programming) สำหรับช่วยอิมพลีเม้นต์ครอสคัทติ้งคอนเซิร์นด้วยการใช้แอสเป็ก (Aspect) ซึ่งการโปรแกรมเชิงแอสเป็กสามารถช่วยแก้ไขปัญหาโค้ดกระจัดกระจายได้ ดังนั้นจึงมีการนำเสนอกระบวนการสำหรับปรับปรุงซอฟต์แวร์เชิงวัตถุเดิมโดยการนำการโปรแกรมเชิงแอสเป็กเข้ามาช่วยปรับปรุงด้วยกระบวนการแอสเป็กรีแฟคทอริง (Aspect refactoring) เป็นกระบวนการสำหรับปรับปรุงซอร์สโค้ดจากการจัดการครอสคัทติ้งคอนเซิร์นเป็นแอสเป็ก (Vidal et al., 2009) หลังจากมีการนำเสนอการโปรแกรมเชิงแอสเป็กเป็นแนวทางในการพัฒนาซอฟต์แวร์จึงทำให้มีผู้ศึกษาเปรียบเทียบคุณภาพซอฟต์แวร์ระหว่างซอฟต์แวร์ที่พัฒนาด้วยการโปรแกรมเชิงวัตถุกับซอฟต์แวร์ที่พัฒนาด้วยการโปรแกรมเชิงแอสเป็ก โดยประเมินด้วยชุดมาตรวัดที่แตกต่างกันไปดังนี้ Kulesza และคณะ (2006) ใช้ชุดมาตรวัดที่ดัดแปลงจากมาตรวัดเชิงวัตถุสามารถแบ่งมาตรวัดตามคุณลักษณะของ

ซอฟต์แวร์ดังนี้ การแยกคอนเซิร์น (Separation of concern - SoC) คัพพลิง (Coupling) โคฮีชัน (Cohesion) และขนาด (Size) ขณะที่ Kumar และคณะ (2007) ประเมินจากความสามารถในการเปลี่ยนแปลงจากผลกระทบของการเปลี่ยนแปลง (change impact) และมาตรวัดเชิงแอสเป็กของ Ceccato และ Tonella (2004) ส่วน Mguni และ Ayalew (2013) ใช้ชุดมาตรวัดสองชุดสำหรับการประเมินคุณภาพซอฟต์แวร์ ได้แก่ ชุดมาตรวัดระดับคอนเซิร์น (Concern level metrics) และชุดมาตรวัดความซับซ้อนทางโครงสร้าง (Structural Complexity Metrics) ถึงแม้หลายงานวิจัยจะประเมินคุณภาพซอฟต์แวร์จากชุดมาตรวัดที่แตกต่างกัน แต่ผลของงานวิจัยแสดงให้เห็นว่าการพัฒนาซอฟต์แวร์ด้วยการโปรแกรมเชิงแอสเป็กช่วยทำให้ความสามารถการบำรุงรักษาของซอฟต์แวร์ดีขึ้นเมื่อเทียบกับการพัฒนาซอฟต์แวร์ด้วยการโปรแกรมเชิงวัตถุ นอกจากนี้งานวิจัยในอดีตยังศึกษาข้อดีของการพัฒนาซอฟต์แวร์ด้วยการโปรแกรมเชิงแอสเป็กโดยเปรียบเทียบจากเวลาที่ใช้พัฒนา พบว่าการจัดการครอสคัทติ้งคอนเซิร์นที่ทำให้เกิดโค้ดโคลนซ้ำกันมากกว่า 36 ตำแหน่งหรือมีส่วนของโค้ดซ้ำกันในหลายคลาสทั้งหมด 36 ตำแหน่ง การอิมพลีเม้นต์ครอสรอสคัทติ้งคอนเซิร์นดังกล่าวด้วยแอสเป็กจะใช้เวลาในการพัฒนาเร็วกว่าการอิมพลีเม้นต์ด้วยการโปรแกรมเชิงวัตถุ (Hanenberg, Kleinschmager and Walter, 2009)

จากงานวิจัยข้างต้นได้พิจารณาเพียงแค่เวลาที่ใช้ในการพัฒนาแต่การตัดสินใจนำแอสเป็กมาใช้ควรพิจารณาคุณภาพซอฟต์แวร์ร่วมด้วย โดยผลการเปรียบเทียบคุณภาพซอฟต์แวร์ของการใช้การโปรแกรมเชิงแอสเป็กที่กล่าวไว้ในงานวิจัยในอดีตยังไม่ได้ศึกษาความแตกต่างระหว่างคุณภาพซอฟต์แวร์จากการนำแอสเป็กมาใช้สำหรับจัดการโค้ดโคลนที่เกิดขึ้นภายในซอร์สโค้ด ผู้วิจัยจึงสนใจศึกษาแนวทางการใช้การโปรแกรมเชิงแอสเป็กที่เหมาะสมสำหรับปรับปรุงซอฟต์แวร์เดิมที่พัฒนาด้วยการโปรแกรมเชิงวัตถุด้วยกระบวนการแอสเป็กรีแฟคทอริง โดยมุ่งสนใจการแก้ไขเพียงลักษณะของปัญหาโค้ดกระจัดกระจายที่ทำให้เกิดจำนวนซ้ำของโค้ดโคลนแตกต่างกัน ดังนั้นเพื่อควบคุมลักษณะของครอสคัทติ้งคอนเซิร์นที่จัดการให้มีความคล้ายกันมากที่สุดและมีจำนวนซ้ำของโค้ดที่หลากหลาย ผู้วิจัยจึงสนใจการจัดการครอสคัทติ้งคอนเซิร์นที่มีลักษณะเมท็อดคอล (Method Call) โดยเปรียบเทียบจากการจัดการจำนวนซ้ำของเมท็อดคอลที่แตกต่างกันเพื่อแนะนำการใช้การโปรแกรมเชิงแอสเป็กที่เหมาะสม

มาตรวัดซอฟต์แวร์ (Software metric) สามารถนำมาใช้ประเมินคุณภาพซอฟต์แวร์ โดยการโปรแกรมเชิงวัตถุได้มีการนำเสนอชุดมาตรวัดซอฟต์แวร์ไว้หลากหลาย เช่น มาตรวัดซีเค (CK metric) (Chidamber and Kemerer, 1993) และมาตรวัดเอ็มโอโอดี (MOOD metric) (Abreu, 1995) เป็นต้น และการโปรแกรมเชิงแอสเป็กมีหลายงานวิจัยที่นำเสนอชุดมาตรวัดเช่นกันโดยส่วนใหญ่ปรับปรุงจากมาตรวัดเชิงวัตถุ เช่น มาตรวัดของ Ceccato และ Tonella (2004) และ มาตรวัดของ Sant'Anna และคณะ (2003) เป็นต้น เนื่องจากงานวิจัยนี้ปรับปรุงซอฟต์แวร์เดิมที่พัฒนาด้วยการโปรแกรมเชิงวัตถุ และผู้วิจัยยังไม่พบงานวิจัยที่บอกความแตกต่างอย่างชัดเจนจากการใช้มาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็กสำหรับวัดคุณภาพซอฟต์แวร์ที่ปรับปรุงด้วยการใช้การโปรแกรมเชิงแอสเป็ก งานวิจัยนี้จึงเลือกใช้ทั้งมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็กสำหรับประเมินคุณภาพซอฟต์แวร์

ดังนั้นงานวิจัยนี้จึงปรับปรุงซอฟต์แวร์เชิงวัตถุด้วยการทำแอสเป็กรีแฟคทอริง โดยแก้ไขซอร์สโค้ดด้วยการจัดการเมท็อดคอลที่ซ้ำกันด้วยแอสเป็ก เพื่อพิจารณาผลของการทำแอสเป็กรีแฟคทอริงจากจำนวนซ้ำของเมท็อดคอลที่แตกต่างกัน โดยพิจารณาคุณภาพซอฟต์แวร์หลังทำแอสเป็กรีแฟคทอริงจากการเก็บรวบรวมค่ามาตรวัดของชุดมาตรวัดทั้งหมดสองชุด ได้แก่ มาตรวัดเชิงวัตถุ (Object-oriented metrics) และมาตรวัดเชิงแอสเป็ก (Aspect-oriented metrics) เพื่อเป็นแนวทางในการแนะนำลักษณะของโค้ดที่เหมาะสมต่อการปรับปรุงซอฟต์แวร์ให้มีคุณภาพด้วยกระบวนการแอสเป็กรีแฟคทอริง

1.2 วัตถุประสงค์ของงานวิจัย

1. เพื่อเปรียบเทียบมาตรวัดเชิงวัตถุของซอฟต์แวร์ที่ปรับปรุงด้วยการทำแอสเป็กรีแฟคทอริงจากจำนวนโค้ดโคลนที่แตกต่างกัน
2. เพื่อเปรียบเทียบมาตรวัดเชิงแอสเป็กของซอฟต์แวร์ที่ปรับปรุงด้วยการทำแอสเป็กรีแฟคทอริงจากจำนวนโค้ดโคลนที่แตกต่างกัน
3. เพื่อวิเคราะห์คุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริงด้วยจำนวนโค้ดโคลนที่แตกต่างกัน

1.3 ขอบเขตของการวิจัย

1. งานวิจัยนี้สนใจศึกษาการปรับปรุงซอฟต์แวร์ด้วยการโปรแกรมเชิงแอสเป็ก จากกระบวนการแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลที่ซ้ำภายในซอร์สโค้ดของซอฟต์แวร์เชิงวัตถุ โดยซอฟต์แวร์เชิงวัตถุพัฒนาด้วยภาษาจาวา (Java) ส่วนการโปรแกรมเชิงแอสเป็กเลือกใช้ภาษาแอสเป็กเจ (AspectJ)
2. การปรับปรุงซอฟต์แวร์ด้วยการทำแอสเป็กรีแฟคทอริงจะสนใจจัดการเมท็อดคอลที่ซ้ำกันด้วยแอสเป็ก โดยการอิมพลีเม้นต์การเรียกใช้งานเมท็อดทำให้เกิดเมท็อดคอลซ้ำกันในหลายคลาส
3. การค้นหาเมท็อดคอลและจำนวนซ้ำของเมท็อดคอลที่เกิดขึ้นภายในซอร์สโค้ดของซอฟต์แวร์จะใช้เทคนิคสำหรับกระบวนการแอสเป็กไมน์นิง (Aspect mining) เทคนิคที่เลือกใช้คือ แฟน-อินอะนาไลซิส (Fan-in analysis) เป็นเทคนิคสำหรับพิจารณาเมท็อดคอลที่ซ้ำกันภายในซอฟต์แวร์
4. มาตรวัดคุณภาพซอฟต์แวร์ที่นำมาใช้ประเมินคุณภาพซอฟต์แวร์ งานวิจัยนี้ได้เลือกใช้ชุดมาตรวัดทั้งหมด 2 ชุดมาตรวัด ได้แก่ มาตรวัดเชิงวัตถุของ Chidamber และ Kemerer (1993) ทั้งหมด 6 มาตรวัด และมาตรวัดเชิงแอสเป็กของ Ceccato และ Tonella (2004) ทั้งหมด 10 มาตรวัด

1.4 ขั้นตอนดำเนินงานวิจัย

1. ศึกษารายละเอียดเกี่ยวกับการโปรแกรมเชิงแอสเป็กและการทำแอสเป็กรีแฟคทอริง
2. ศึกษารายละเอียดเกี่ยวกับการวัดคุณภาพซอฟต์แวร์สำหรับซอฟต์แวร์ที่พัฒนาด้วยการโปรแกรมเชิงวัตถุและซอฟต์แวร์ที่พัฒนาด้วยการโปรแกรมเชิงแอสเป็ก
3. ศึกษาเครื่องมือที่ใช้ประกอบในการทำวิจัย
4. ออกแบบการทดลองเพื่อศึกษาคุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลในจำนวนซ้ำของเมท็อดคอลที่แตกต่างกันด้วยแอสเป็ก
5. เลือกซอฟต์แวร์ที่จะเป็นกรณีศึกษาสำหรับนำมาใช้ในการทดลองเพื่อตอบวัตถุประสงค์ของงานวิจัย
6. ทดลองตามที่ได้ออกแบบไว้
7. วิเคราะห์ผลการทดลอง
8. จัดทำเอกสารสรุปการวิจัย

1.5 ข้อจำกัดของการวิจัย

1. เนื่องจากงานวิจัยนี้ศึกษาการจัดการครอสคัทติ้งคอนเซิร์นเฉพาะลักษณะเมท็อดคอล โดยเกิดการซ้ำกันของเมท็อดคอลภายในหลายคลาส ผลการทดลองที่ได้จึงอาจจะไม่สามารถอ้างอิงกับการจัดการครอสคัทติ้งคอนเซิร์นในลักษณะอื่นๆ เช่น ลักษณะกลุ่มของคำสั่ง (Statement) เป็นต้น
2. เนื่องจากงานวิจัยนี้ศึกษาการพัฒนาซอฟต์แวร์ด้วยการโปรแกรมเชิงแอสเป็กโดยใช้ภาษาแอสเป็กเจ ผลการทดลองที่ได้จึงไม่สามารถอ้างอิงกับการพัฒนาด้วยภาษาโปรแกรมเชิงแอสเป็กภาษาอื่น เพราะโครงสร้างภาษาอาจมีความซับซ้อนแตกต่างกัน ซึ่งอาจทำให้ผลคุณภาพซอฟต์แวร์ที่ได้แตกต่างออกไป
3. เนื่องจากข้อจำกัดของภาษาแอสเป็กเจ ทำให้ไม่สามารถจัดการเมท็อดคอลที่ซ้ำกันภายในซอฟต์แวร์ทั้งหมดได้ เช่น เมท็อดคอลที่อยู่ภายในคำสั่งเงื่อนไข (Conditional statement) เป็นต้น จำนวนซ้ำของเมท็อดคอลที่เลือกจัดการด้วยแอสเป็กในงานวิจัยนี้จึงนับเฉพาะเมท็อดคอลที่สามารถจัดการด้วยแอสเป็กได้เท่านั้น

1.6 คำจำกัดความที่ใช้ในการวิจัย

1. คอนเซิร์น (Concern) หมายถึง ความต้องการ (Software Requirement) ที่สามารถตอบสนองต่อความต้องการของสเตคโฮลเดอร์ (Stakeholder) คอนเซิร์นสามารถประกอบด้วยหนึ่งความต้องการหรือหลายความต้องการที่สอดคล้องกัน (Hilliard, 2002; Laddad, 2003b; Sutton and Rouvellou, 2002)
2. ครอสคัทติ้งคอนเซิร์น (Crosscutting concern) หมายถึง คอนเซิร์นที่ไม่สามารถอิมพลีเม้นต์ด้วยคลาสหรือเมท็อดชัดเจนได้ การอิมพลีเม้นต์ครอสคัทติ้งคอนเซิร์นจะทำให้เกิดโค้ดกระจัดกระจายหรือโค้ดซ้ำกันอยู่ในหลายคลาส และทำให้มีโค้ดที่ตอบสนองต่อหลายคอนเซิร์นรวมอยู่ในคลาสเดียวกัน (Bhatti, 2008; Ceccato et al., 2006; Yuen, 2009)

3. โค้ดโคลน (Code clone) หมายถึง ส่วนของโค้ดที่เหมือนกันหรือคล้ายกันกับส่วนของโค้ดอื่นภายในซอร์สโค้ด ประเภทของโค้ดโคลนสามารถจำแนกได้เป็น (1) ความเหมือนกันของข้อความ (Textual Similarity) และ (2) ความเหมือนกันของฟังก์ชัน (Functional Similarity) โค้ดโคลนจะมีผลต่อคุณภาพซอฟต์แวร์ด้านความสามารถการบำรุงรักษา (Kamiya, Kusumoto and Inoue, 2002; Roy and Cordy, 2007)

4. การโปรแกรมเชิงแอสเป็ก (Aspect-oriented programming) หมายถึง การโปรแกรมที่ช่วยแก้ไขปัญหาโค้ดกระจัดกระจายและโค้ดพันกัน โดยจัดการครอสคัทติ้งคอนเซิร์นด้วยการสร้างแอสเป็ก (Laukkanen, 2008)

5. แอสเป็กไมน์นิง (Aspect mining) หมายถึง กระบวนการสำหรับค้นหาครอสคัทติ้งคอนเซิร์นที่มีในซอร์สโค้ดของซอฟต์แวร์ เพื่อให้ได้ครอสคัทติ้งคอนเซิร์นที่เหมาะสมสำหรับจัดการเป็นแอสเป็ก (Vidal et al., 2009)

6. แอสเป็กรีแฟคทอริง (Aspect refactoring) หมายถึง กระบวนการสำหรับจัดการครอสคัทติ้งคอนเซิร์น โดยการจัดการโค้ดที่เกี่ยวข้องกับครอสคัทติ้งคอนเซิร์นให้เป็นแอสเป็กด้วยการใช้โครงสร้างภาษาของการโปรแกรมเชิงแอสเป็ก (Vidal et al., 2009)

7. คุณภาพซอฟต์แวร์ (Software quality) หมายถึง คุณลักษณะทั้งหมดของซอฟต์แวร์ที่สามารถตอบสนองต่อความพึงพอใจและความต้องการของผู้ใช้ หากนำคุณลักษณะต่างๆมารวมไว้ในซอฟต์แวร์จะช่วยเพิ่มสมรรถภาพของซอฟต์แวร์ให้มีอายุการใช้งานที่นาน (Lifetime performance) (Fitzpatrick, 1996; Panovski, 2008)

8. มาตรวัดซอฟต์แวร์ (Software metric) หมายถึง กลุ่มข้อกำหนดสำหรับอธิบายการวัด (Measurement) โดยค่าตัวเลขที่ได้จะบ่งชี้ถึงคุณลักษณะของซอฟต์แวร์ โดยสามารถพิจารณาผ่านแบบจำลองเพื่อคาดคะเนความต้องการทรัพยากรของซอฟต์แวร์และคุณภาพซอฟต์แวร์ (Fenton and Pfleeger, 1998)

1.7 ประโยชน์ที่คาดว่าจะได้รับ

1. เพื่อเป็นแนวทางในการพัฒนาซอฟต์แวร์ด้วยการโปรแกรมเชิงแอสเป็กจากการปรับปรุงซอร์สโค้ดด้วยกระบวนการแอสเป็กรีแฟคทอริง โดยพิจารณาจำนวนโค้ดโคลนที่เกิดขึ้นเนื่องจากการอิมพลีเม้นต์ครอสคัทติ้งคอนเซิร์นและลักษณะโค้ดที่เหมาะสมต่อการจัดการด้วยแอสเป็ก โดยสามารถใช้ประโยชน์จากการโปรแกรมเชิงแอสเป็กและปรับปรุงคุณภาพซอฟต์แวร์ให้ดียิ่งขึ้น

2. ช่วยพัฒนาองค์ความรู้ที่เกี่ยวข้องกับการทำแอสเป็กรีแฟคทอริงสำหรับพัฒนาซอฟต์แวร์ด้วยการโปรแกรมเชิงแอสเป็ก

บทที่ 2
การทบทวนวรรณกรรม

2.1 บทนำ

การนำเสนอวรรณกรรมในอดีตที่เกี่ยวข้องมีจุดประสงค์เพื่อรายงานความเป็นมา ความหมาย ทฤษฎี และงานวิจัยในอดีตสำหรับใช้เป็นแนวทางในการศึกษาเปรียบเทียบคุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริงด้วยจำนวนโค้ดโคลนที่แตกต่างกัน ประกอบด้วยหัวข้อย่อย คือ (1) คอนเซิร์น (Concern) (2) การโปรแกรมเชิงแอสเป็ก (Aspect-Oriented Programming) (3) แอสเป็กไมน์นิง (Aspect Mining) (4) แอสเป็กรีแฟคทอริง (Aspect Refactoring) (5) คุณภาพซอฟต์แวร์ (Software Quality) และ (6) งานวิจัยที่เกี่ยวข้อง

2.2 คอนเซิร์น (Concern)

ปัจจุบันหลายงานวิจัยได้ให้นิยามคำว่า คอนเซิร์น (Concern) ไว้หลากหลายดังนี้

Hilliard (2002) กล่าวว่าคอนเซิร์นเป็นสิ่งที่สนใจที่เกี่ยวข้องกับการพัฒนาซอฟต์แวร์ โดยการดำเนินงานหรือคุณลักษณะของสิ่งที่สนใจมีความสำคัญต่อสเตคโฮลเดอร์ (Stakeholder)

Sutton และ Rouvellou (2002) กล่าวว่าคอนเซิร์นเป็นสิ่งที่สนใจในซอฟต์แวร์ สามารถกำหนดด้วยกลุ่มของความต้องการ (Requirement) ที่สอดคล้องกัน

Laddad (2003) กล่าวว่าคอนเซิร์นเป็นความต้องการ (Requirement) หรือสิ่งที่ควรพิจารณาเพื่อให้บรรลุเป้าหมายของซอฟต์แวร์

Kaur และ Johari (2009) กล่าวว่าคอนเซิร์นเป็นความต้องการเชิงฟังก์ชัน (Functional requirement) และความต้องการที่ไม่ใช่เชิงฟังก์ชัน (Non-functional requirement) ที่สามารถตอบสนองต่อความต้องการของสเตคโฮลเดอร์

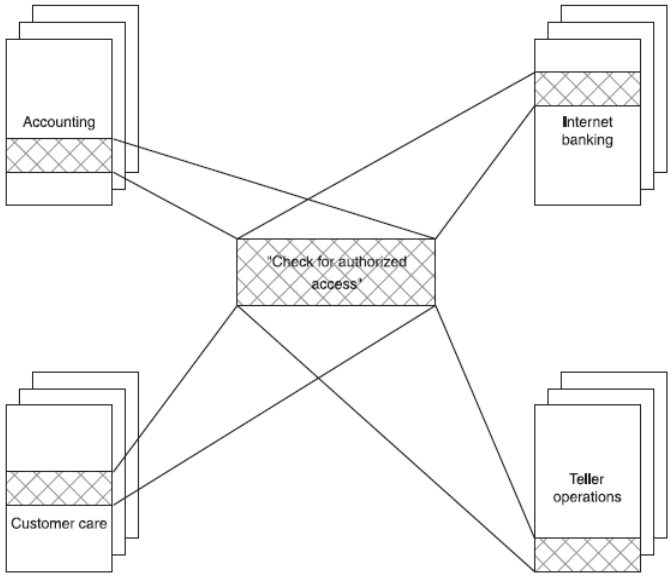
ดังนั้นงานวิจัยนี้จึงกำหนดนิยามของคำว่าคอนเซิร์น คือ ความต้องการ (Requirement) ที่สามารถตอบสนองต่อความต้องการของสเตคโฮลเดอร์ โดยคอนเซิร์นประกอบด้วยหนึ่งความต้องการหรือหลายความต้องการที่สอดคล้องกัน ความต้องการดังกล่าวสามารถแบ่งเป็นความต้องการเชิงฟังก์ชัน (Functional requirement) และความต้องการที่ไม่ใช่เชิงฟังก์ชัน (Non-functional requirement) (Hilliard, 2002; Kaur and Johari, 2009; Laddad, 2003b; Sutton and Rouvellou, 2002)

การพัฒนาซอฟต์แวร์ด้วยการโปรแกรมเชิงวัตถุ ทุกอ็อบเจ็กต์ (Object) หรือ แพคเกจ (Package) ของอ็อบเจ็กต์สามารถรองรับการทำงานเพื่อตอบสนองความต้องการเชิงฟังก์ชัน และเนื่องด้วยการพัฒนาซอฟต์แวร์ต้องตอบสนองต่อหลายความต้องการเชิงฟังก์ชัน ทำให้การอิมพลีเม้นต์ (Implement) คอนเซิร์นประเภทหนึ่งไม่สามารถอิมพลีเม้นต์ด้วยคลาสหรือเมท็อดเดียวได้ ผลจากการอิมพลีเม้นต์คอนเซิร์นทำให้เกิดปัญหาโค้ดกระจัดกระจายหรือเกิดโค้ดซ้ำกันอยู่ในหลายคลาส นอกจากนี้ทำให้คลาสหรือเมท็อดที่รับผิดชอบการทำงานของคอนเซิร์นหลักมีคอนเซิร์นประเภทอื่น

รวมอยู่ด้วย เรียกคอนเซิร์นประเภทนี้ว่า ครอสคัทติ้งคอนเซิร์น (Crosscutting concern) (Marin, Deursen and Moonen, 2007) การอิมพลีเม้นต์ครอสคัทติ้งคอนเซิร์นจึงทำให้เกิดโค้ดกระจัดกระจาย (Code Scattering) และ โค้ดพันกัน (Code Tangling) (Laddad, 2003b) ดังนี้

(1) โค้ดกระจัดกระจาย (Code Scattering)

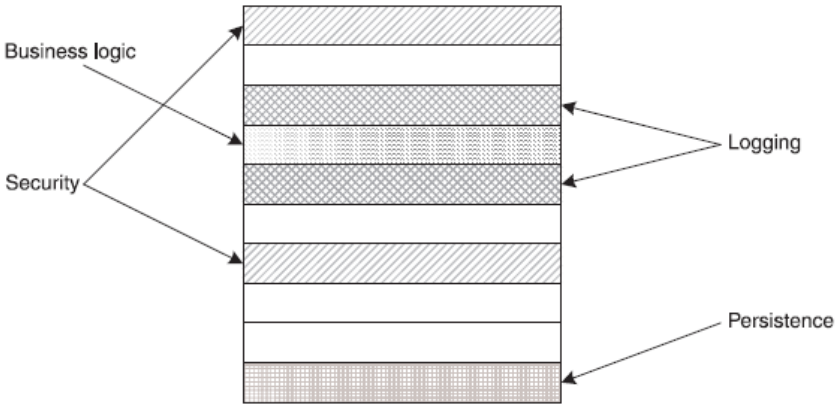
โค้ดกระจัดกระจายเกิดเมื่อส่วนของโค้ดที่เกี่ยวข้องกับครอสคัทติ้งคอนเซิร์นมีตำแหน่งทำงานซ้ำกันอยู่ในหลายคลาส จากรูปที่ 2.1 แสดงตัวอย่างซอฟต์แวร์ธนาคารและการอิมพลีเม้นต์ครอสคัทติ้งคอนเซิร์น ได้แก่ การตรวจสอบสิทธิ์การเข้าใช้งานซอฟต์แวร์ (Authorization) เพื่อตรวจสอบสิทธิ์ผู้ใช้งานสำหรับการเข้าใช้บริการจากคลาสต่างๆ การอิมพลีเม้นต์การตรวจสอบสิทธิ์การเข้าใช้งานทำให้เกิดโค้ดที่เกี่ยวข้องกระจัดกระจายอยู่ในหลายคลาส โดยเป็นคลาสที่เกี่ยวข้องกับคอนเซิร์นหลักดังนี้ การบัญชี (Accounting) การทำธุรกรรมทางการเงินผ่านอินเทอร์เน็ต (Internet banking) การบริการลูกค้าสัมพันธ์ (Customer care) และ การดำเนินการรับฝาก - ถอนเงิน (Teller operation)



รูปที่ 2.1 ลักษณะโค้ดกระจัดกระจายของครอสคัทติ้งคอนเซิร์น ของ Laddad (2003)

(2) โค้ดพันกัน (code tangling)

โค้ดพันกันเกิดเมื่อคลาสหนึ่งมีการทำงานของโค้ดที่เกี่ยวข้องกับหลายคอนเซิร์นรวมอยู่ภายในคลาสเดียวกัน จากรูปที่ 2.2 แสดงตัวอย่างของคลาสที่อิมพลีเม้นต์หลายคอนเซิร์นรวมอยู่ภายในคลาสเดียว คอนเซิร์นที่อิมพลีเม้นต์ ได้แก่ ตรรกะที่เกี่ยวข้องกับธุรกิจ (Business logic) การบันทึกล็อก (Logging) ความปลอดภัย (Security) และเพอร์ซิสเทนซ์ (Persistence)



รูปที่ 2.2 ลักษณะโค้ดพันกันของครอสคัทติ้งคอนเซิร์น ของ Laddad (2003)

ตัวอย่างของครอสคัทติ้งคอนเซิร์น (Ahuja and Goel, 2008) ได้แก่

(1) การบันทึกล็อก (Logging) เป็นการบันทึกรายละเอียดของการทำงานภายในซอฟต์แวร์ การบันทึกล็อกจึงมีเป้าหมายเพื่อสร้างความเข้าใจในพฤติกรรมของซอฟต์แวร์ ทดสอบซอฟต์แวร์ (Software testing) และการดีบัก (Debugging) การบันทึกล็อกต้องแทรกคำสั่ง (Statement) เอาไว้หลายตำแหน่งในเมท็อดของคลาสทำให้เกิดโค้ดซ้ำกันในหลายตำแหน่ง การบันทึกล็อกจึงเป็นครอสคัทติ้งคอนเซิร์น (Ahuja and Goel, 2008) ตัวอย่างการอิมพลีเม้นต์การบันทึกล็อก ดังรูป 2.3 ภายในคลาส ShoppingCart ทั้งเมท็อด addItem และ removeItem จะมีคำสั่งของการบันทึกล็อกแทรกอยู่ภายในเมท็อด โดยการบันทึกล็อกจะไม่เกี่ยวข้องกับการทำงานฟังก์ชันหลักของเมท็อด ดังนั้นส่งผลให้การบันทึกล็อกกระจัดกระจายหรือซ้ำกันอยู่ในหลายเมท็อดของคลาสและมีโค้ดที่ไม่ได้เกี่ยวข้องกับการทำงานฟังก์ชันหลักรวมอยู่ (Laddad, 2003b)

```
import java.util.*;
import java.util.logging.*;
public class ShoppingCart {
    static Logger logger = Logger.getLogger("trace");
    private List items = new Vector();
    public void addItem(Item item) {
        logger.logp(Level.INFO,"ShoppingCart", "addItem", "Entering");
        items.add(item);
    }
    public void removeItem(Item item) {
        logger.logp(Level.INFO,"ShoppingCart", "removeItem", "Entering");
        items.remove(item);
    }
}
```

รูปที่ 2.3 ตัวอย่างการบันทึกล็อกของคลาส ShoppingCart ของ Laddad (2003)

(2) การติดตาม (Tracing) เป็นการติดตามระหว่างที่ซอฟต์แวร์กระทำการเพื่อให้ทราบโลจิคอลโฟลว์ (Logical flow) ของซอฟต์แวร์ โดยการติดตามส่งผลทำให้ทราบถึงปัญหาที่เกี่ยวข้องกับฟังก์ชันและปัญหาสมรรถภาพของซอฟต์แวร์ การติดตามจึงต้องพิมพ์คำสั่งทุกข้อความที่มีในทุกส่วนโปรแกรม (Component) ของซอฟต์แวร์ (Ahuja and Goel, 2008)

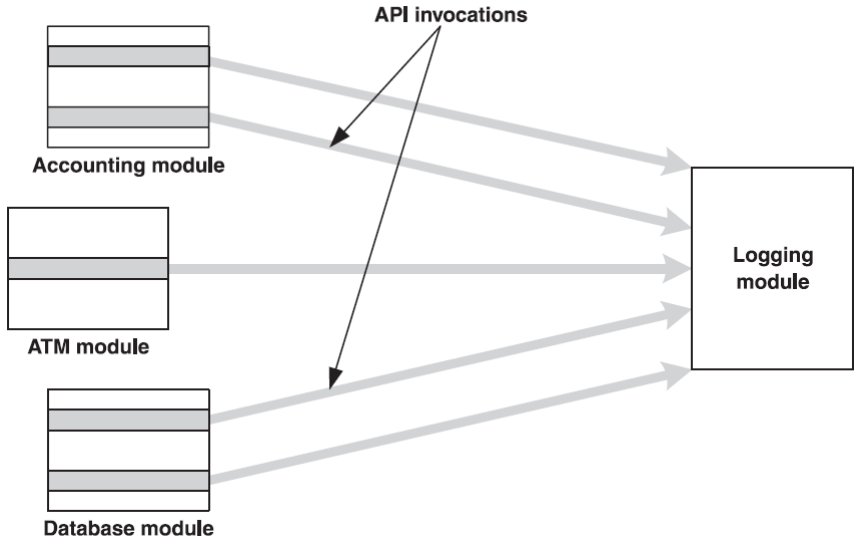
(3) การอนุญาตและการพิสูจน์ตัวตน (Authorization and Authentication) โดยการพิสูจน์ตัวตนเป็นกระบวนการที่ดำเนินการเพื่อตรวจสอบและพิสูจน์ว่าผู้ใช้สามารถเข้าถึงและใช้งานระบบได้ ดังนั้นการพิสูจน์ตัวตนจะดำเนินการก่อน ส่วนการอนุญาตเป็นกระบวนการที่ทำหลังจากผู้ใช้งานสามารถเข้าใช้งานระบบได้ โดยการอนุญาตจะเป็นกระบวนการที่ควบคุมบทบาทของผู้ใช้งานว่าฟังก์ชันไหนของระบบที่ผู้ใช้งานสามารถเข้าถึงและใช้งานได้ (Ahuja and Goel, 2008)

(4) การตรวจหาและการจัดการข้อยกเว้น (Exception Detection and Handling) โดยข้อยกเว้น (Exception) เป็นพฤติกรรมของซอฟต์แวร์ที่บ่งบอกว่าโอเปอเรชั่นไม่สามารถดำเนินการได้สำเร็จลุล่วง สามารถกำหนดด้วยการจัดการข้อยกเว้นว่าจะกู้คืนการดำเนินการโอเปอเรชั่นใหม่หรือเลือกที่จะไม่สนใจข้อยกเว้นดังกล่าว ดังนั้นโค้ดที่ใช้สำหรับการตรวจหาและการจัดการข้อยกเว้นจึงรวมอยู่กับโค้ดฟังก์ชันการทำงานหลัก (Ahuja and Goel, 2008)

(5) แคชชิง (Caching) เป็นการเก็บข้อมูลที่มีการเข้าถึงบ่อยครั้งในหน่วยความจำ เพื่อลดจำนวนครั้งของการเรียกใช้งานข้อมูลจากดาต้าเบสโดยตรงและทำให้สมรรถภาพของซอฟต์แวร์ปรับปรุงดีขึ้น (Ahuja and Goel, 2008)

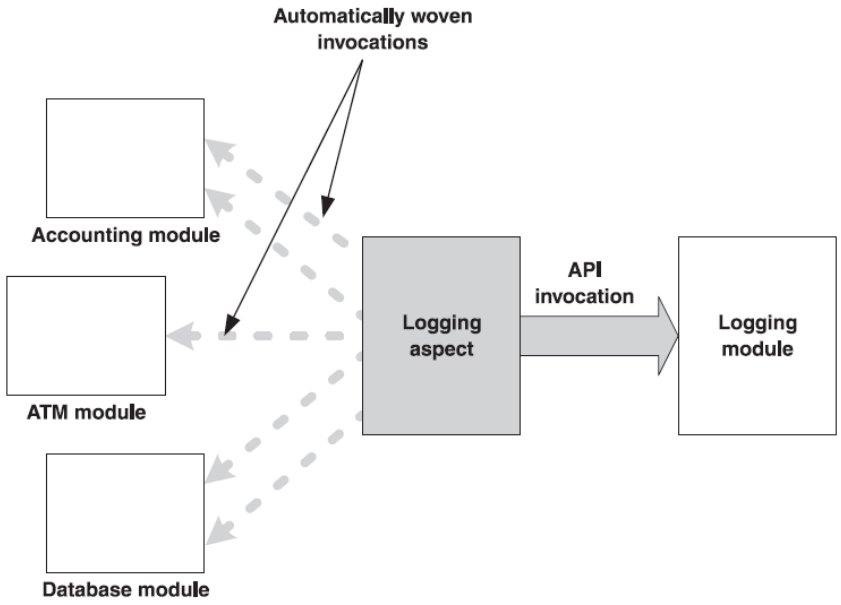
2.3 การโปรแกรมเชิงแอสเป็ก (Aspect-Oriented Programming)

การอิมพลีเม้นต์บางฟังก์ชันด้วยการโปรแกรมเชิงวัตถุส่งผลให้เกิดปัญหาโค้ดกระจัดกระจาย (Code scattering) ตัวอย่างดังรูปที่ 2.4 การอิมพลีเม้นต์ฟังก์ชันการบันทึกล็อกภายในคลาสต่างๆ ด้วยการกำหนดเมท็อดคอลสำหรับเรียกใช้งานฟังก์ชันการบันทึกล็อก จึงทำให้เกิดเมท็อดคอลซ้ำๆกันที่เรียกใช้งานฟังก์ชันการบันทึกล็อกอยู่ภายในเมท็อดของคลาสต่างๆ (Laddad, 2003b)



รูปที่ 2.4 การอิมพลีเม้นต์การบันทึกล็อกด้วยเมท็อดคอล ของ Laddad (2003)

Kiczales และคณะ (1997) นำเสนอการโปรแกรมเชิงแอสเป็กสำหรับช่วยแก้ไขปัญหาโค้ดกระจัดกระจาย โดยจัดการครอสคัทติ้งคอนเซิร์นด้วยการสร้างยูนิตใหม่ที่แยกออกมาจากคลาสเรียกว่าแอสเป็ก (Aspect) (Laukkanen, 2008) จากปัญหาดังรูปที่ 2.4 ข้างต้น สามารถนำการโปรแกรมเชิงแอสเป็กเข้ามาช่วยอิมพลีเม้นต์การบันทึกล็อกด้วยการสร้างแอสเป็ก และกำหนดตำแหน่งการทำงานของซอฟต์แวร์ที่ต้องการให้ดำเนินการบันทึกล็อก เช่น เมื่อทุกเมท็อดทำงาน (Execute method) จะกำหนดให้ตำแหน่งดังกล่าวเกิดการบันทึกล็อก เป็นต้น โดยภายในแอสเป็กจะเป็นผู้รับผิดชอบในการเรียกใช้งานฟังก์ชันการบันทึกล็อกแทน ทำให้ช่วยลดปัญหาเมท็อดคอลที่ซ้ำกันภายในคลาสต่างๆ และทำให้การขึ้นต่อกันระหว่างคลาสฟังก์ชันหลักและคลาสบันทึกล็อกลดลง ดังรูปที่ 2.5

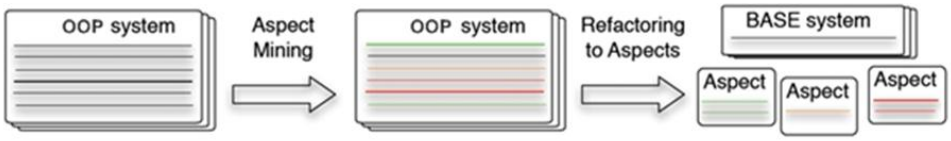


รูปที่ 2.5 การอิมพลีเม้นต์การบันทึกล็อกด้วยแอสเป็ก ของ Laddad (2003)

การนำการโปรแกรมเชิงแอสเป็กมาประยุกต์ใช้กับซอฟต์แวร์เดิมที่พัฒนาด้วยการโปรแกรมเชิงวัตถุด้วยหลักการรีเอ็นจิเนียริ่ง (Reengineering) ประกอบด้วย 2 ขั้นตอนตามลำดับ (Ceccato et al., 2006) ดังรูปที่ 2.6

(1) แอสเป็กไมน์นิง (Aspect mining) เป็นกระบวนการสำหรับค้นหาครอสคัทติ้งคอนเซิร์นที่มีในซอฟต์แวร์ เพื่อให้ได้ครอสคัทติ้งคอนเซิร์นที่เหมาะสมสำหรับจัดการเป็นแอสเป็ก (Vidal et al., 2009)

(2) แอสเป็กรีแฟคทอริง (Aspect refactoring) เป็นกระบวนการสำหรับจัดการครอสคัทติ้งคอนเซิร์น โดยจัดการโค้ดที่เกี่ยวข้องกับครอสคัทติ้งคอนเซิร์นด้วยการใช้แอสเป็ก โครงสร้างภาษาของการโปรแกรมเชิงแอสเป็ก ได้แก่ แอสเป็ก (Aspect) จอยน์พอยท์ (Join point) พอยท์คัท (Pointcut) แอดไวซ์ (Advice) และอินโทรดักชัน (Introduction) (Vidal et al., 2009)



รูปที่ 2.6 การปรับปรุงซอฟต์แวร์ด้วยการโปรแกรมเชิงแอสเป็ก
ดัดแปลงจากงานวิจัยของ Kellens, Mens, และ Tonella (2007)

แอสเป็กเจ (AspectJ) เป็นภาษาโปรแกรมเชิงแอสเป็กภาษาแรกและเป็นที่นิยม โดยเป็นภาษาที่พัฒนาขยายมาจากภาษาจาวาจึงสามารถนำมาใช้พัฒนาร่วมกับซอฟต์แวร์ที่พัฒนาด้วยภาษาจาวาได้ (Kiczales et al., 2001) อีคลิปส์-เอเจดีที (Eclipse-AJDT) เป็นเครื่องมือที่รองรับสำหรับการโปรแกรมด้วยภาษาแอสเป็กเจ สามารถระบุกฎสำหรับการวีฟวิง (Weaving rule) เป็นการระบุตำแหน่งการทำงานของซอฟต์แวร์ที่ต้องการให้โค้ดที่เกี่ยวข้องกับครอสคัทติ้งคอนเซิร์นทำงาน สามารถกำหนดด้วยการใช้โครงสร้างต่างๆ ดังนี้

(1) แอสเป็ก (Aspect) เป็นยูนิตจำเพาะ (Modularization unit) สำหรับอิมพลีเม้นต์ครอสคัทติ้งคอนเซิร์น โดยภายในแอสเป็กมีโค้ดระบุกฎวีฟวิงสำหรับทั้งครอสคัทติ้งแบบไดนามิก (Dynamic) และสแตติก (Static) ครอสคัทติ้งแบบไดนามิกเป็นการระบุพฤติกรรมใหม่ที่เพิ่มเข้ามาหรือแทนที่การดำเนินการของซอฟต์แวร์ ส่วนครอสคัทติ้งแบบสแตติกไม่กระทบพฤติกรรมของซอฟต์แวร์ แต่เป็นการปรับเปลี่ยนโครงสร้างของซอฟต์แวร์ เช่น การเพิ่มแอตทริบิวต์หรือเมท็อดให้กับคลาส เป็นต้น ภายในแอสเป็กจะมีพอยท์คัท แอดไวซ์ อินโทรดักชัน และการประกาศตัวแปร แอสเป็กจึงมีลักษณะคล้ายกับคลาสของการโปรแกรมเชิงวัตถุ (Laddad, 2003b)

(2) จอยน์พอยท์ (Join point) คือตำแหน่งใดๆที่อยู่ในการดำเนินการของซอฟต์แวร์ เป็นตำแหน่งที่ระบุเพื่อให้แอสเป็กเข้าไปขัดขวางการทำงานของซอฟต์แวร์ และทำงานตามโค้ดที่กำหนดภายในแอสเป็กแทน ตัวอย่างจอยน์พอยท์ เช่น การเรียกใช้งานเมท็อด การดำเนินการเมท็อด และการกำหนดค่าแอตทริบิวต์ เป็นต้น (Laddad, 2003b)

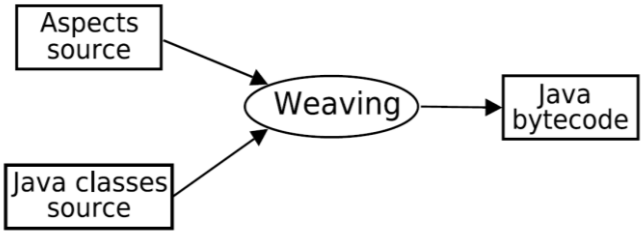
(3) พอยท์คัท (Pointcut) เป็นโครงสร้างสำหรับนำจอยน์พอยท์หรือกลุ่มของจอยน์พอยท์มาใช้โดยกำหนดและสร้างไว้ภายในแอสเป็ก อีกทั้งภายในพอยท์คัทสามารถเก็บค่าในตำแหน่งจอยน์พอยท์ที่ระบุเพื่อนำค่าดังกล่าวมาใช้ในแอดไวซ์ พอยท์คัทสามารถจำแนกได้ดังนี้ พอยท์คัทสำหรับเมท็อด พอยท์คัทสำหรับแอตทริบิวต์หรือฟิลด์ พอยท์คัทสำหรับการจัดการข้อยกเว้น พอยท์คัทสำหรับคอนโทรลโฟลว์ (Cavallaro and Miraz, 2010; Laddad, 2003b)

(4) แอดไวซ์ (Advice) แอดไวซ์จะมีลักษณะคล้ายเมท็อดของการโปรแกรมเชิงวัตถุ เป็นส่วนสำหรับระบุโค้ดที่ต้องการให้ทำงานเมื่อซอฟต์แวร์ดำเนินการมาถึงจอยน์พอยท์ที่กำหนดไว้ด้วยพอยท์คัท โดยโค้ดดังกล่าวอาจเป็นส่วนที่เพิ่มหรือแทนที่การดำเนินการของซอฟต์แวร์ สามารถกำหนดด้วยประเภทของแอดไวซ์ ดังนี้ แอดไวซ์ประเภทบีฟอร์ (Before) สำหรับกำหนดให้โค้ดภายในแอดไวซ์ทำงานก่อนพอยท์คัทที่กำหนด แอดไวซ์ประเภทอาฟเตอร์ (After) สำหรับกำหนดให้โค้ด

ภายในแอดไวซ์ทำงานหลังพอยท์คัทที่กำหนด และแอดไวซ์ประเภทอะราวด์ (Around) สำหรับกำหนดให้โค้ดภายในแอดไวซ์ทำงานแทนที่พอยท์คัทที่กำหนด (Laddad, 2003b)

(5) อินโทรดักชัน (Introduction) หรือ การประกาศอินเตอร์-ไทป์ (Inter-type declarations) เป็นคำสั่งสำหรับครอสคัทติ้งแบบสแตติก (Static crosscutting) โดยคำสั่งจะปรับเปลี่ยนคลาส อินเตอร์เฟส และแอสเป็กภายในซอร์สโค้ด การปรับเปลี่ยนจะไม่กระทบพฤติกรรมของซอฟต์แวร์ เช่น การแก้ไขลำดับชั้นคลาส การเพิ่มแอตทริบิวต์หรือฟิลด์ใหม่ให้กับคลาส การเพิ่มเมท็อดใหม่ให้กับคลาส เป็นต้น (Cavallaro and Miraz, 2010; Laddad, 2003b)

แอสเป็กเจสามารถจัดการครอสคัทติ้งคอนเซิร์นด้วยการสร้างแอสเป็ก เริ่มต้นโดยการนำโค้ดที่เกี่ยวข้องกับครอสคัทติ้งคอนเซิร์นออกจากตำแหน่งเดิมภายในคลาส และระบุจอยน์พอยท์ด้วยการสร้างพอยท์คัทเพื่อกำหนดไปยังตำแหน่งจอยน์พอยท์ และนำโค้ดที่เกี่ยวข้องกับครอสคัทติ้งคอนเซิร์นมาไว้ในแอดไวซ์ของแอสเป็ก ทั้งคลาสและแอสเป็กที่สร้างจะถูกคอมไพร์ลเป็นจาวาไบต์โค้ด (Java bytecode) ด้วยแอสเป็กเจคอมไพเลอร์ (AspectJ compiler) โดยภายในไบต์โค้ด โค้ดที่อยู่ภายในแอดไวซ์ของแอสเป็กจะถูกนำเข้าไปแทรกตามตำแหน่งที่ระบุไว้ด้วยพอยท์คัท กระบวนการดังกล่าวจะเรียกว่าวีฟวิง (Weaving) โดยจะมีแอสเป็กวีฟเวอร์ (Aspect Weaver) เป็นตัวรวมการทำงานระหว่างคลาสและแอสเป็ก ดังรูปที่ 2.7 (Cavallaro and Miraz, 2010; Filman et al., 2005)



รูปที่ 2.7 กระบวนการวีฟวิง ของ Cavallaro และ Miraz (2010)

นอกจากนี้ยังมีภาษาโปรแกรมเชิงแอสเป็กสำหรับนำมาพัฒนาร่วมกับภาษาโปรแกรมอื่นๆ ได้แก่ แอสเป็กซี (AspectC) แอสเป็กซี++ (AspectC++) แอสเป็กเอส (AspectS) แจสโค (JasCo) และไฮเปอร์เจ (HyperJ) เป็นต้น (Singh and Gill, 2012) นอกจากภาษาโปรแกรมเชิงแอสเป็กยังมีการนำเสนอกรอบแนวคิด (Framework) สำหรับพัฒนาด้วยการโปรแกรมเชิงแอสเป็กมากมาย ได้แก่ เจเอซี (JAC) เจบอสเอโอพี (JBoss AOP) สปริง (Spring) และแอสเป็กเวิคซ์ (AspectWerkz) เป็นต้น (Pawlak, Seinturier and Retaillé, 2005)

2.4 แอสเป็กไมน์นิง (Aspect Mining)

แอสเป็กไมน์นิง (Aspect Mining) เป็นกระบวนการสำหรับค้นหาครอสคัทติ้งคอนเซิร์นภายในซอร์สโค้ดของซอฟต์แวร์ เพื่อให้ได้ครอสคัทติ้งคอนเซิร์นที่สามารถจัดการด้วยแอสเป็ก (Vidal et al., 2009) และเนื่องจากการค้นหาครอสคัทติ้งคอนเซิร์นด้วยตนเองค่อนข้างยากเพราะปัจจุบันซอฟต์แวร์มีขนาดใหญ่และมีความซับซ้อน ประกอบกับต้องมีความรู้เกี่ยวกับฟังก์ชันการทำงานของซอฟต์แวร์ จึงทำให้การค้นหาภายในซอร์สโค้ดด้วยตนเองอาจเกิดแนวโน้มความผิดพลาด (error-

prone) ดังนั้นจึงมีหลายงานวิจัยนำเสนอเทคนิคหรือเครื่องมือสำหรับช่วยค้นหาครอสคัทติ้งคอนเซิร์นที่มีในซอฟต์แวร์ (Bhatti, 2008)

ช่วงต้นของแอสเป็กไมน์นิงส่วนใหญ่เป็นแนวคิดของการพัฒนาเครื่องมือสำหรับช่วยผู้พัฒนาค้นหา (Browsing) และนำทาง (Navigating) ภายในซอร์สโค้ดเพื่อวิเคราะห์หาครอสคัทติ้งคอนเซิร์น โดยบางเครื่องมือเป็นการวิเคราะห์คำศัพท์ (Lexical analysis) และบางเครื่องมือใช้แนวคิดการค้นหาอิงรูปแบบ (Type-based search) ตัวอย่างของเครื่องมือ ได้แก่ แอสเป็กเบราว์เซอร์ (Aspect Browser) (Griswold, Kato and Yuan, 1999) แอสเป็กไมน์นิงทูล (Aspect Mining Tool – AMT) (Hannemann and Kiczales, 2001) และคอนเซิร์นกราฟ (Concern graphs) (Robillard and Murphy, 2002) เป็นต้น

ต่อมาจึงมีหลายงานวิจัยพยายามพัฒนาแนวคิดแอสเป็กไมน์นิงให้มีความอัตโนมัติมากขึ้นสำหรับการวิเคราะห์ซอร์สโค้ดและค้นหาครอสคัทติ้งคอนเซิร์น โดยนำเสนอเทคนิคแอสเป็กไมน์นิงต่างๆ (Kellens, Mens and Tonella, 2007) ดังนี้

Breu และ Krinke (2004) นำเสนอเทคนิคเอ็กซีคิวท์ชันแพทเทิร์น (Execution patterns) สำหรับค้นหาครอสคัทติ้งคอนเซิร์นโดยวิเคราะห์จากลำดับของเหตุการณ์ (Event trace) ที่สามารถแสดงถึงพฤติกรรมของซอฟต์แวร์ และความสัมพันธ์ระหว่างการดำเนินการ (Execution relation) ที่เกิดจากการเรียกใช้งานเมท็อด จากการพิจารณาความสัมพันธ์ระหว่างการดำเนินการจะสามารถค้นหารูปแบบของการเรียกใช้งานเมท็อดที่เกิดซ้ำกันได้

Tonella และ Ceccato (2004) นำเสนอเทคนิคไดนามิกอะนาไลซิส (Dynamic analysis) พิจารณาครอสคัทติ้งคอนเซิร์นโดยพิจารณาจากลำดับการดำเนินงาน (Execution trace) ที่สร้างขึ้นมาจากการรันซอฟต์แวร์ ทำให้ทราบลำดับของเมท็อดที่ทำงาน ต่อมาจึงวิเคราะห์ลำดับการทำงานของเมท็อดด้วยการวิเคราะห์แบบฟอร์มอลคอนเซ็บท์ (Formal concept analysis) ผลลัพธ์ที่ได้ทำให้สามารถนำมาใช้พิจารณาเมท็อดที่เป็นครอสคัทติ้งคอนเซิร์น

Tourwe และ Mens (2004) นำเสนอเทคนิคไอเด็นติไฟเออร์อะนาไลซิส (Identifier analysis) โดยนำการวิเคราะห์แบบฟอร์มอลคอนเซ็บท์ (Formal concept analysis) มาพิจารณากฎเกณฑ์การตั้งชื่อของคลาสและเมท็อด กำหนดให้คลาสและเมท็อดเป็นอ็อบเจ็กต์ (Object) และคำที่ตัดมาจากชื่อคลาสและเมท็อดเป็นแอตทริบิวต์ (Attribute) เช่น อ็อบเจกต์ คือคลาส QuotedCodeConstant แอตทริบิวต์คือ ‘Quoted’ ‘Code’ และ ‘Constant’ กลุ่มของคลาสหรือเมท็อดที่ใช้แอตทริบิวต์ร่วมกันมากสุดจะพิจารณาเป็นครอสคัทติ้งคอนเซิร์น

Shepherd และ Pollock (2005) และ He และ Bai (2005) นำเสนอการค้นหาครอสคัทติ้งคอนเซิร์นด้วยเทคนิคคลัสเตอร์ริง (Clustering) เริ่มต้นจากการกำหนดให้แต่ละเมท็อดอยู่ในคลัสเตอร์ที่แตกต่างกัน จากนั้นคลัสเตอร์ที่มีความเหมือนกันโดยพิจารณาจากค่าระยะห่าง (distance) หากค่าระยะห่างระหว่างคลัสเตอร์น้อย คลัสเตอร์จะถูกรวมกันเป็นคลัสเตอร์ใหม่ Shepherd และ Pollock (2005) ใช้คำนวณค่าระยะห่างจากการตัดคำของชื่อเมท็อดและพิจารณาคำที่ชื่อเมท็อดใช้ร่วมกัน ส่วน He และ Bai (2005) พิจารณาค่าระยะห่างจากความสัมพันธ์สเตติกไดเร็คอินโวเคชั่น (Static Direct Invocation Relationship - SDIR) โดยค่าระยะห่างสามารถแสดงความสัมพันธ์ระหว่าง

เมท็อด หากมีค่าระยะห่างน้อยแสดงว่าเมท็อดถูกเรียกใช้งานด้วยกันบ่อยครั้ง แต่หากค่าระยะห่างสูงแสดงว่าเมท็อดไม่ค่อยถูกเรียกใช้งานร่วมกัน

Shepherd และคณะ (2004) และ Bruntink และคณะ (2004) นำเสนอเทคนิคสำหรับค้นหาครอสคัทติ้งคอนเซิร์น โดยใช้การค้นหาโคลน (Clone) Shepherd และคณะ (2004) จะใช้การค้นหาโคลนแบบพีดีจี (PDG) พิจารณาจากกราฟการขึ้นต่อกันของโปรแกรม (Program dependency graph – PDG) ภายในกราฟจะแทนแต่ละคำสั่ง (Statement) ด้วยโหนด และแทนความสัมพันธ์ของการขึ้นต่อกันระหว่างคำสั่งด้วยเส้นเชื่อมระหว่างโหนด Bruntink และคณะ (2004) จะใช้การค้นหาโคลนแบบเอเอสที (AST) โดยแสดงซอร์สโค้ดในรูปแบบแอ็บสแตรคซินแทกซ์ทรี (Abstract syntax tree - AST) จากนั้นค้นหาครอสคัทติ้งคอนเซิร์นภายในแอ็บสแตรคซินแทกซ์ทรีจากซับทรีที่มีความเหมือนกัน

Marin, Deursen และ Moonen (2007) นำเสนอเทคนิคแฟน-อินอะนาไลซิส (Fan-in analysis) เป็นเทคนิคสำหรับค้นหาครอสคัทติ้งคอนเซิร์นโดยพิจารณาจากปัญหาโค้ดกระจัดกระจาย โดยประเมินจากจำนวนการเรียกใช้งานเมท็อดซึ่งอิมพลีเม้นต์ฟังก์ชันที่เกี่ยวข้องกับครอสคัทติ้งคอนเซิร์น ดังนั้นค่าแฟน-อินเมตริก (fan-in metric) จึงเป็นค่าที่แสดงจำนวนการถูกเรียกใช้งานของเมท็อด โดยการเรียกใช้งานมาจากคลาสอื่นๆภายในซอร์สโค้ดของซอฟต์แวร์ ค่าแฟน-อินเมตริกจึงใช้เป็นตัวชี้วัดสำหรับปัญหาโค้ดกระจัดกระจาย หากค่าแฟน-อินเมตริกมีค่าสูงสามารถพิจารณาว่าเกิดปัญหาโค้ดกระจัดกระจายและโค้ดดังกล่าวเป็นครอสคัทติ้งคอนเซิร์น กล่าวคือมีฟังก์ชันเหมือนกันกระจายซ้ำกันอยู่ในหลายคลาส ดังนั้นค่าแฟน-อินที่สูงจึงสามารถจัดการด้วยแอสเป็ก ตัวอย่างคอนเซิร์นประเภทนี้ ได้แก่ การบันทึกล็อก (Logging) การติดตาม (Tracing) การตรวจสอบเงื่อนไขก่อนหรือหลัง (Pre- and post- condition check) และการจัดการข้อยกเว้น (Exception handling) เป็นต้น เทคนิคแฟน-อินอะนาไลซิสประกอบด้วย 3 ขั้นตอน (Marin et al., 2007) ดังนี้

1. การคำนวณค่าแฟน-อินของเมท็อดทั้งหมด

การคำนวณค่าแฟน-อินของแต่ละเมท็อดพิจารณาจากจำนวนการเรียกใช้งานเมท็อดดังกล่าว โดยค่าแฟน-อินจะพิจารณาเมท็อดที่เป็นพอลลีมอร์ฟิค (Polymorphic) ด้วย ดังนั้นการพิจารณาค่าแฟน-อิน หากมีการเรียกใช้งานเมท็อด M จากหลายคลาสที่แตกต่างกัน จะพิจารณาค่าแฟน-อินจากตำแหน่งทั้งหมดที่เรียกใช้งานเมท็อด M และกรณีที่เมท็อด M เป็นแอ็บสแตรคเมท็อด (Abstract Method) ที่ถูกเรียกใช้งานจากคลาส A และคลาส C สามารถพิจารณาค่าแฟน-อินของเมท็อด M มีค่าเท่ากับ 2

2. การคัดกรองผลลัพธ์เมท็อด

เมื่อคำนวณค่าแฟน-อินของเมท็อดทั้งหมดเรียบร้อยแล้ว ขั้นตอนนี้เป็นการคัดกรองผลลัพธ์เมท็อดบางส่วนออกไปเพื่อให้ได้เมท็อดที่เหมาะสมสำหรับเป็นครอสคัทติ้งคอนเซิร์น โดยพิจารณาจาก (1) เมท็อดที่มีค่าแฟน-อินสูงกว่าค่าเธรโชลด์ (Threshold) โดย Marin, Deursen และ Moonen (2007) แนะนำว่าส่วนใหญ่จะกำหนดค่าเธรโชลด์เท่ากับ 10 (2) เมท็อดที่เกี่ยวข้องกับการเข้าถึงแอตทริบิวต์ของคลาส ได้แก่ เมท็อดประเภทเกตเตอร์ (Getter) และเซตเตอร์ (Setter) จะไม่พิจารณาเป็นครอสคัทติ้งคอนเซิร์น (3) เมท็อดที่เกี่ยวข้องกับยูทิลิตี้ (Utility) เช่น เมท็อด toString() จะไม่พิจารณาเป็นครอสคัทติ้งคอนเซิร์นเช่นกัน

3. การวิเคราะห์ผลลัพธ์เมท็อด

การพิจารณาผลลัพธ์เมท็อดทั้งหมดเพื่อเลือกครอสคัทติ้งคอนเซิร์นที่เหมาะสม โดยพิจารณาเมท็อดที่มีค่าแฟน-อินสูง และพิจารณาตำแหน่งการเรียกใช้งานเมท็อดดังกล่าวจากคลาสต่างๆเพื่อให้สามารถจัดการด้วยพอยท์คัทของภาษาแอสเป็กเจได้ เช่น ตำแหน่งการเรียกใช้งานของเมท็อดมักเกิดขึ้นในตอนต้นหรือตอนท้ายของเมท็อด และตำแหน่งการเรียกใช้งานเมท็อดเกิดขึ้นภายในหลายๆ เมท็อดที่มีชื่อเมท็อดคล้ายกัน เป็นต้น

นอกจากนี้ Marin, Deursen และ Moonen (2007) ยังได้พัฒนาเครื่องมือชื่อเอฟไอเอ็นที (FINT) สำหรับรองรับการใช้เทคนิคแฟน-อินอะนาไลซิสเพื่อค้นหาครอสคัทติ้งคอนเซิร์นด้วยเครื่องมือดังกล่าวสามารถนำมาใช้งานร่วมกับโปรแกรมอีคลิปส์ (Eclipse) โดยเครื่องมือเอฟไอเอ็นทีพิจารณาจากซอร์สโค้ด (Source code) ของโปรเจค (Project) แพคเกจ (Package) และคลาส (Class) โดยพิจารณาและคำนวณค่าแฟน-อินของแต่ละเมท็อดอัตโนมัติ พร้อมทั้งแสดงผลลัพธ์รายชื่อของเมท็อดที่ถูกเรียกใช้งานจากคลาสต่างๆ

2.5 แอสเป็กรีแฟคทอริง (Aspect Refactoring)

รีแฟคทอริง (Refactoring) เป็นกระบวนการหรือกลุ่มเทคนิคสำหรับจัดเรียงโค้ดใหม่โดยยังคงพฤติกรรมเดิมของซอฟต์แวร์เอาไว้ (Laddad, 2003a)

แอสเป็กรีแฟคทอริง (Aspect refactoring) หรือรีแฟคทอริงเชิงแอสเป็ก (Aspect-oriented refactoring) สามารถแบ่งออกเป็น 3 ประเภทใหญ่ๆ (Almasri and Albayouk, 2006) ได้แก่

(1) Refactoring to aspects (OO -> AO) เป็นการปรับปรุงโค้ดที่พัฒนาด้วยการโปรแกรมเชิงวัตถุด้วยการใช้โครงสร้างของแอสเป็กเข้ามาปรับปรุง โดยการนำโค้ดที่เกี่ยวข้องกับครอสคัทติ้งคอนเซิร์นมาไว้ในแอสเป็ก

(2) AO -> AO Refactorings เป็นการปรับปรุงโครงสร้างภายในของแอสเป็ก

(3) Aspect-aware Refactorings เป็นการปรับปรุงแอสเป็กให้ยังคงพฤติกรรมเดิมของซอฟต์แวร์และทำงานได้อย่างถูกต้อง เมื่อมีการรีแฟคทอริงส่วนของโค้ดที่พัฒนาด้วยการโปรแกรมเชิงวัตถุ

สำหรับงานวิจัยนี้เลือกกล่าวถึงแอสเป็กรีแฟคทอริงประเภท Refactoring to aspects (OO -> AO) เพียงอย่างเดียว แอสเป็กรีแฟคทอริงในงานวิจัยนี้จึงเป็นกระบวนการสำหรับช่วยจัดเรียงโค้ดโดยจัดการครอสคัทติ้งคอนเซิร์นด้วยการใช้แอสเป็ก ส่งผลทำให้ซอฟต์แวร์ง่ายต่อการทำความเข้าใจและบำรุงรักษาซอฟต์แวร์ รวมถึงแก้ไขปัญหาโค้ดกระจัดกระจายและโค้ดพันกันได้ (Laddad, 2003b; Vidal et al., 2009) การทำแอสเป็กรีแฟคทอริงมีงานวิจัยในอดีตนำเสนอเทคนิคต่างๆเอาไว้ดังนี้

Laddad (2003) นำเสนอเทคนิคแอสเป็กรีแฟคทอริงที่สามารถนำมาใช้จัดการครอสคัทติ้งคอนเซิร์น โดยมีเทคนิคหลายรูปแบบ ได้แก่

(1) เทคนิคเอ็กซ์แทรคเมท็อดคอล (Extract method call)

(2) เทคนิคเอ็กซ์แทรคเอ็กซ์เซ็พชั่นแฮนด์ลิ่ง (Extract exception handling)

(3) เทคนิคเอ็กซ์แทรคคอนเคอร์เรนซีคอนโทรล (Extract concurrency control)

(4) เทคนิคเอ็กซ์แทรคเวิร์คเกอร์อ็อบเจกต์ครีเอชั่น (Extract worker object creation)

(5) เทคนิครีเพลสอาร์กิวเมนต์ทริคเกิลบายวอร์มโฮล (Replace argument trickle by wormhole)

(6) เทคนิคเอ็กซ์แทรคอินเตอร์เฟสอิมพลีเมนเทชั่น (Extract interface implementation)

(7) เทคนิครีเพลสโอเวอร์ไรด์วิทแอดไวซ์ (Replace override with advice)

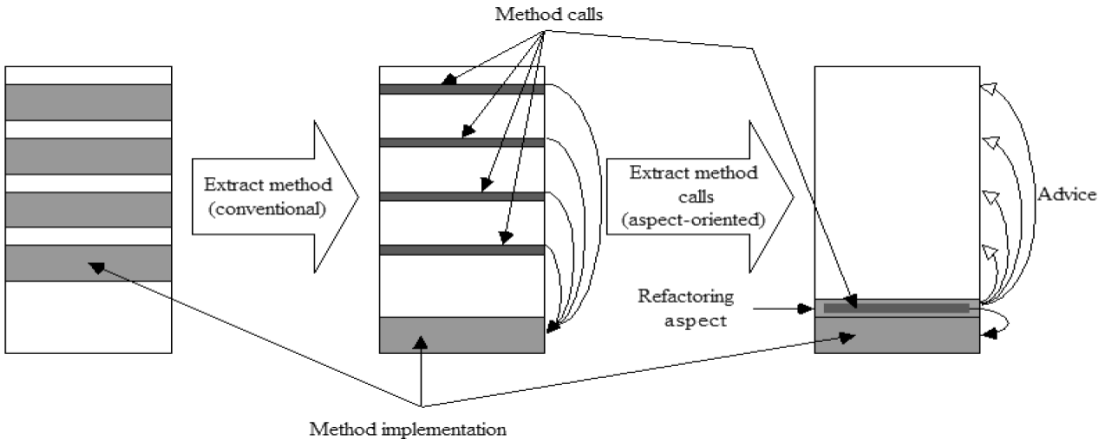
(8) เทคนิคเอ็กซ์แทรคเลซีอินนิเชียลไลเซชั่น (Extract Lazy initialization)

(9) เทคนิคเอ็กซ์แทรคคอนแทรคเอนฟอร์สเม้นท์ (Extract contract enforcement)

ในที่นี้จะขอกล่าวถึงเทคนิคแอสเป็กรีแฟคทอริง 5 เทคนิคที่สำคัญดังนี้

(1) เทคนิคเอ็กซ์แทรคเมท็อดคอล (Extract method call)

เทคนิคนี้สามารถนำมาใช้เพิ่มเติมจากการรีแฟคทอริงเดิมของการโปรแกรมเชิงวัตถุ ได้แก่ เอ็กซ์แทรคเมท็อด (Extract Method) โดยเอ็กซ์แทรคเมท็อดเป็นการดึงส่วนของโค้ดที่ซ้ำกันหลายที่ออกมาไว้ในเมท็อดและแทนที่ส่วนของโค้ดเดิมด้วยเมท็อดคอล การใช้เทคนิคเอ็กซ์แทรคเมท็อดคอลของการทำแอสเป็กรีแฟคทอริงเป็นส่วนเพิ่มเติมดังรูปที่ 2.8 โดยในรูปทางขวาสุดหลังจากการทำแอสเป็กรีแฟคทอริง เมท็อดคอลที่ซ้ำกันถูกนำออกมาไว้ในแอดไวซ์ของแอสเป็ก และกำหนดให้พอยท์คัทระบุตำแหน่งไปยังตำแหน่งเดิมที่เมท็อดคอลเคยทำงานแทน



รูปที่ 2.8 การใช้เทคนิคเอ็กซ์แทรคเมท็อดคอล ของ Laddad (2003)

ตัวอย่างคลาส Account ที่มีเมท็อดคอลซ้ำกันภายในคลาสสามารถแสดงได้ดังรูปที่ 2.9 สามารถนำเทคนิคเอ็กซ์แทรคเมท็อดคอลมาปรับปรุงโค้ดโดยมีขั้นตอนดังนี้

```
package banking;
import java.security.AccessController;
public class Account {
  private int _accountNumber;
  private float _balance;
  public Account(int accountNumber) {
      _accountNumber = accountNumber;
  }
  public int getAccountNumber() {
     AccessController.checkPermission(new BankingPermission("accountOperation"));
     return _accountNumber;
  }
  public void credit(float amount) {
     AccessController.checkPermission(new BankingPermission("accountOperation"));
      _balance = _balance + amount;
  }
  public void debit(float amount) throws InsufficientBalanceException {
     AccessController.checkPermission(new BankingPermission("accountOperation"));
      if (_balance < amount) {
          throw new InsufficientBalanceException("Insufficient total balance");
      } else {
          _balance = _balance - amount;
      }
  }
  public float getBalance() {
     AccessController.checkPermission(new BankingPermission("accountOperation"));
     return _balance;
  }
  public String toString() {
     return "Account: " + _accountNumber;
  }
}
```

รูปที่ 2.9 ตัวอย่างเมท็อดคอลซ้ำภายในคลาส ของ Laddad (2003)

1) เพิ่มแอสเป็กและกำหนดพอยท์คัทที่สามารถระบุถึงจอยน์พอยท์ทั้งหมด โดยจอยน์พอยท์เป็นตำแหน่งการทำงานของซอฟต์แวร์ที่ต้องการให้เมท็อดคอลทำงาน ต่อมาสร้างแอดไวซ์สำหรับพอยท์คัทดังกล่าว โดยระบุว่าต้องการให้โค้ดภายในแอดไวซ์ดำเนินการก่อน หลัง หรือแทนที่การทำงานของซอฟต์แวร์ ขั้นตอนนี้ยังไม่ระบุโค้ดการทำงานภายในแอดไวซ์ สามารถกำหนดได้ดังรูปที่ 2.10

```
private static aspect PermissionCheckAspect {
        private pointcut permissionCheckedExecution() :
                (execution(public int Account.getAccountNumber())
                || execution(public void Account.credit(float))
                || execution(public void Account.debit(float) throws InsufficientBalanceException)
                || execution(public float Account.getBalance()))
                   && within(Account) ;
       before() : permissionCheckedExecution() {}
}
```

รูปที่ 2.10 ตัวอย่างการสร้างแอสเป็ก พอยท์คัทและแอดไวซ์ ของ Laddad (2003)

2) เพิ่มฟังก์ชันของครอสคัทติ้งคอนเซิร์นภายในแอดไวซ์ ในกรณีนี้คือเพิ่มเมท็อดคอลที่ต้องการจัดการภายในแอดไวซ์ ดังรูปที่ 2.11 จากนั้นให้นำเมท็อดคอลที่ซ้ำกันทั้งหมดออกจากภายในคลาส

```
before() : permissionCheckedExecution() {
        AccessController.checkPermission(new BankingPermission(“accountOperation”));
}
```

รูปที่ 2.11 การกำหนดเมท็อดคอลที่ต้องการให้ทำงานภายในแอดไวซ์ ของ Laddad (2003)

3) แก้ไขพอยท์คัทให้ครอบคลุมมากขึ้นด้วยการใช้ไวลด์การ์ด (Wildcard) ดังรูปที่ 2.12 เพื่อให้ง่ายต่อการเพิ่มหรือแก้ไขฟังก์ชันของซอฟต์แวร์ในภายหลัง

```
private pointcut permissionCheckedExecution() :
    (execution(public * Account.*(..))
    && !execution(String Account.toString()))
    && within(Account);
```

รูปที่ 2.12 ตัวอย่างการแก้ไขพอยท์คัทด้วยไวลด์การ์ด ของ Laddad (2003)

หลังจากจัดการเมท็อดคอลภายในคลาส Account ด้วยเทคนิคเอ็กซ์แทรคเมท็อดคอลของแอสเป็กรีแฟคทอริง จะได้คลาส Account ที่จัดเรียงโค้ดใหม่เรียบร้อยดังรูปที่ 2.13

```
package banking;
import java.security.AccessController;
public class Account {
      private int _accountNumber;
      private float _balance;
      public Account(int accountNumber) {
           _accountNumber = accountNumber;
      }
      public int getAccountNumber() {
          return _accountNumber;
      }
      public void credit(float amount) {
            _balance = _balance + amount;
      }
      public void debit(float amount) throws InsufficientBalanceException {
           if (_balance < amount) {
                throw new InsufficientBalanceException("Insufficient total balance");
           } else {
                 _balance = _balance - amount;
           }
       }
       public float getBalance() {       return _balance;       }
       public String toString() {
            return "Account: " + _accountNumber;
       }
       private static aspect PermissionCheckAspect {
            private pointcut permissionCheckedExecution() :
                  (execution(public * Account.*(..))
                  && !execution(String Account.toString()))
                  && within(Account);
            before() : permissionCheckedExecution() {
                  AccessController.checkPermission(new BankingPermission(“accountOperation”));
            }
       }
}
```

รูปที่ 2.13 ตัวอย่างคลาส Account หลังทำแอสเป็กรีแฟคทอริง ของ Laddad (2003)

(2) เทคนิคเอ็กซ์แทรคเอ็กซ์เซ็พชั่นแฮนด์ลิ่ง (Extract exception handling)

การจัดการข้อยกเว้น (Exception handling) เป็นครอสคัทติ้งคอนเซิร์นที่มีอยู่ในหลายคลาส โดยมีโครงสร้างคือบล็อกของ try/catch ที่ไม่สามารถจัดการได้ด้วยการรีแฟคทอริงแบบเดิมของการโปรแกรมเชิงวัตถุ ตัวอย่างการจัดการข้อยกเว้นที่ซ้ำกันในคลาสสามารถแสดงได้ดังรูปที่ 2.14

```
public class LibraryDelegate {
      private LibrarySession _session;
      public LibraryDelegate() throws LibraryException { init(); }
      private void init() throws LibraryException {
           try {
                LibrarySessionHome home = (LibrarySessionHome)ServiceLocator.getInstance().
                getRemoteHome("Library", LibrarySessionHome.class);
                _session = home.create();
           } catch (ServiceLocatorException ex) {
                throw new LibraryException(ex);
           } catch (CreateException ex) {
                throw new LibraryException(ex);
           } catch (RemoteException ex) {
                throw new LibraryException(ex);
           }
      }
      public void addBook(BookTO book) throws LibraryException {
           try {
                  _session.addBook(book);
           } catch (RemoteException ex) {
                throw new LibraryException(ex);
          }
      }
      public void removeBook(BookTO book) throws LibraryException {
           try {
                _session. removeBook(book);
           } catch (RemoteException ex) {
                throw new LibraryException(ex);
          }
      }
}
```

รูปที่ 2.14 ตัวอย่างการจัดการข้อยกเว้นที่ซ้ำกันภายในคลาส LibraryDelegate ของ Laddad (2003)

จากตัวอย่างบล็อก try/catch ที่ซ้ำกันภายในคลาส LibraryDelegate สามารถจัดการด้วยการดึงบล็อกดังกล่าวออกมาเขียนไว้ในแอสเป็ก ตัวอย่างแอสเป็กที่อิมพลีเม้นต์การจัดการข้อยกเว้นดังรูปที่ 2.15

```
aspect LibaryExceptionHandling {
      declare soft : RemoteException
         : call(* *.*(..) throws RemoteException) && within(LibraryDelegate);
      declare soft : ServiceLocatorException
         : call(* *.*(..) throws ServiceLocatorException) && within(LibraryDelegate);
      declare soft : CreateException
         : call(* *.*(..) throws CreateException) && within(LibraryDelegate);
      after() throwing(SoftException ex) throws LibraryException
          : execution(* LibraryDelegate.*(..) throws LibraryException)
          && within(LibraryDelegate) {
             throw new LibraryException(ex.getWrappedThrowable());
      }
}
```

รูปที่ 2.15 ตัวอย่างการใช้เทคนิคเอ็กซ์แทรคเอ็กซ์เซ็พชั่นแฮนด์ลิ่ง ของ Laddad (2003)

(3) เทคนิคเอ็กซ์แทรคคอนเคอร์เรนซีคอนโทรล (Extract concurrency control)

การอิมพลีเม้นต์การควบคุมภาวะพร้อมกัน (Concurrency control) สามารถทำให้เกิดโค้ดซ้ำกันในหลายเมท็อด ตัวอย่างของการควบคุมภาวะพร้อมกัน ได้แก่ แพทเทิร์นสำหรับล็อกการอ่าน-เขียน (Read-write lock pattern) เป็นต้น สำหรับเมท็อดการอ่านข้อมูล ก่อนการอ่านข้อมูลทุกครั้งต้องเรียกใช้งานการล็อกการอ่านและเมื่ออ่านข้อมูลสำเร็จจึงยกเลิกการล็อกการอ่าน ส่วนเมท็อดการแก้ไขข้อมูล ก่อนการแก้ไขข้อมูลทุกครั้งต้องเรียกใช้งานการล็อกการเขียนและเมื่อแก้ไขข้อมูลสำเร็จจึงยกเลิกการล็อกการเขียนเช่นเดียวกัน (Laddad, 2003a)

(4) เทคนิคเอ็กซ์แทรคอินเตอร์เฟสอิมพลีเมนเทชั่น (Extract interface implementation)

กรณีที่หลายคลาสอิมพลีเม้นต์อินเตอร์เฟสสามารถทำให้เกิดโค้ดซ้ำภายในคลาส ในกรณีที่ภายในคลาสอินเตอร์เฟสไม่ได้สร้างการอิมพลีเม้นต์โดยปริยาย (Default implement) เอาไว้ ทำให้คลาสเหล่านั้นต้องกำหนดการอิมพลีเม้นต์ของเมท็อดที่รับมาจากอินเตอร์เฟสเอง สามารถจัดการได้ด้วยการใช้อินโทรดักชันหรือการประกาศอินเตอร์-ไทป์ (Laddad, 2003a)

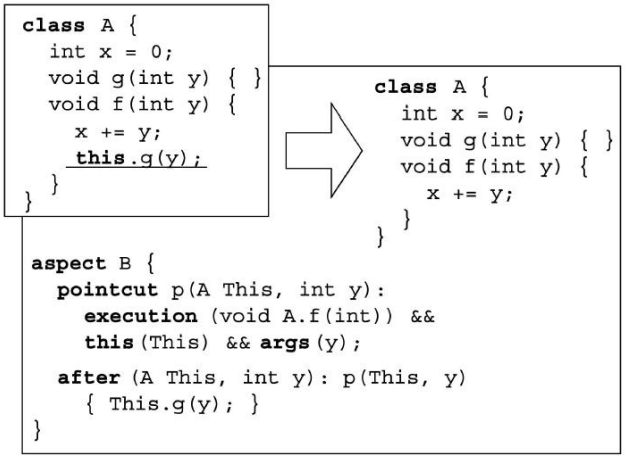
(5) เทคนิคเอ็กซ์แทรคคอนแทรคเอนฟอร์สเม้นท์ (Extract contract enforcement)

เป็นเทคนิคสำหรับจัดการโค้ดที่เกี่ยวข้องกับการตรวจสอบเงื่อนไขทั้งเงื่อนไขแบบก่อนและหลัง (pre-condition และ post-condition) โดยเป็นเงื่อนไขสำหรับตรวจสอบค่าของพารามิเตอร์ที่รับเข้ามาและค่าที่ส่งคืนกลับไปของเมท็อด การจัดการโค้ดดังกล่าวสามารถจัดการโดยนำมาไว้ในแอสเป็ก (Almasri and Albayouk, 2006; Laddad, 2003a)

นอกจากนี้ Binkley และคณะ (2006) ได้นำเสนอเทคนิคสำหรับการทำแอสเป็กรีแฟคทอริงไว้เช่นกัน โดยนำเสนอเทคนิคสำหรับจัดการครอสคัทติ้งคอนเซิร์นที่มีลักษณะเมท็อดคอล มีทั้งหมด 6 เทคนิค (Binkley et al., 2006) ดังนี้

(1) เทคนิคเอ็กซ์แทรคบิกินนิง/เอนออฟเมท็อด (Extract Beginning/End of Method)

ตำแหน่งเมท็อดคอลที่ต้องการจัดการอยู่ในตอนต้นหรือตอนท้ายของเมท็อด จากรูปที่ 2.16 ภายในคลาส A และเมท็อด f() จะมีเมท็อดคอล this.g(y) อยู่ในตอนท้ายของเมท็อด สามารถจัดการเมท็อดคอลด้วยการสร้างแอสเป็กและแอดไวซ์ประเภทอาฟเตอร์ (After) จากนั้นกำหนดพอยท์คัท p โดยระบุตำแหน่งจอยน์พอยท์ไปยังการดำเนินการของเมท็อด f() ดังนั้นเมื่อซอฟต์แวร์ดำเนินการมาถึงเมท็อด f() และดำเนินการเมท็อดเสร็จสิ้น โค้ดภายในแอดไวซ์ของแอสเป็กคือ เมท็อดคอล this.g(y) จึงทำงาน พารามิเตอร์ y ที่ต้องใช้ภายในแอดไวซ์จะได้จากการระบุ arg() ภายในพอยท์คัท ดังนั้นในกรณีเมท็อดคอลอยู่ในตอนต้นของเมท็อดสามารถจัดการโดยการเปลี่ยนเป็นแอดไวซ์ประเภทบีฟอร์ (Before)

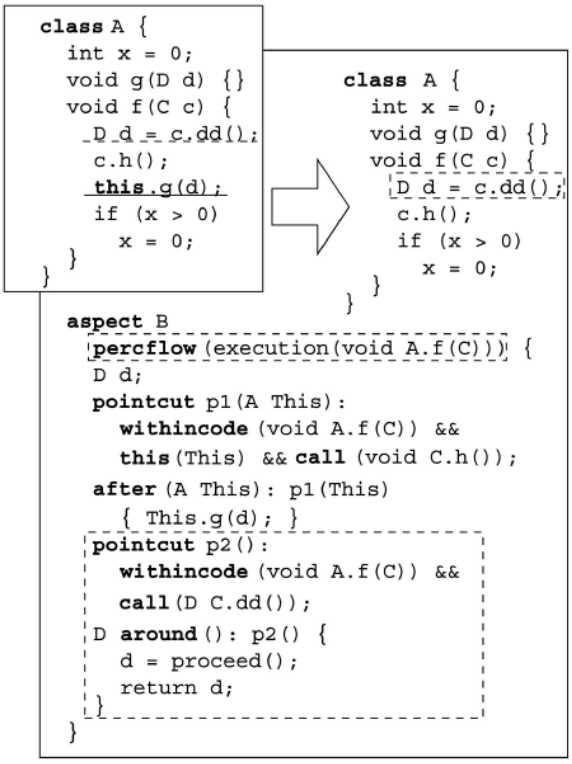


รูปที่ 2.16 เทคนิคเอ็กซ์แทรคบิกินนิง/เอนออฟเมท็อดของ Binkley และคณะ (2006)

(2) เทคนิคเอ็กซ์แทรคบีฟอร์/อาฟเตอร์คอล (Extract before/after Call)

ตำแหน่งเมท็อดคอลที่ต้องการจัดการอยู่ก่อนหรือหลังเมท็อดคอลอื่นๆ จากรูปที่ 2.17 ภายในคลาส A และเมท็อด f() มีเมท็อดคอล this.g(d) โดยตำแหน่งอยู่หลังเมท็อดคอล c.h() สามารถจัดการเมท็อดคอลดังกล่าวด้วยการสร้างแอสเป็กและแอดไวซ์ประเภทอาฟเตอร์ จากนั้นกำหนดพอยท์คัท p1 โดยระบุตำแหน่งจอยน์พอยท์ไปยังการเรียกใช้งานเมท็อด c.h() ดังนั้นเมื่อซอฟต์แวร์ดำเนินการมาถึง c.h() หลังจากเรียกใช้งาน c.h() เสร็จสิ้น โค้ดภายในแอดไวซ์ของแอสเป็ก

คือ เมท็อดคอล this.g(d) จึงทำงาน พอยท์คัท p2 และ percflow เป็นการกำหนดเพื่อเก็บค่าของตัวแปร d ที่ต้องใช้เป็นพารามิเตอร์สำหรับเมท็อดคอล This.g(d) ที่อยู่ในแอดไวซ์ของพอยท์คัท p1

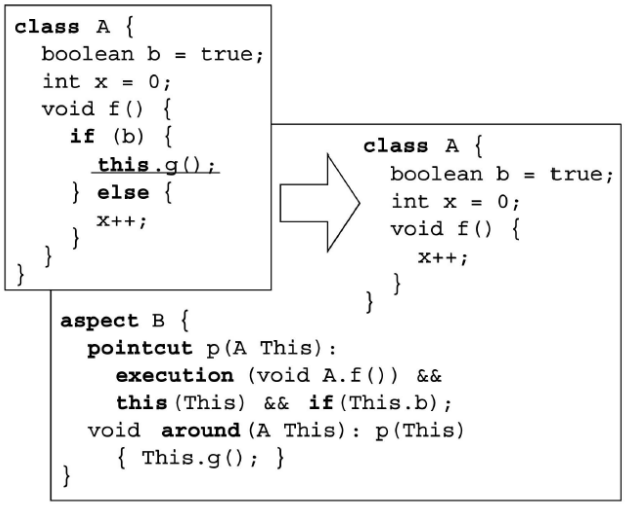


รูปที่ 2.17 เทคนิคเอ็กซ์แทรคบีฟอร์/อาฟเตอร์คอล ของ Binkley และคณะ (2006)

(3) เทคนิคเอ็กซ์แทรคคอนดิชันนอล (Extract Conditional)

ตำแหน่งเมท็อดคอลที่ต้องการจัดการอยู่ภายใต้คำสั่งเงื่อนไข (Conditional statement) จากรูปที่ 2.18 ภายในคลาส A และเมท็อด f() จะมีเมท็อดคอล this.g() โดยตำแหน่งอยู่ภายใต้คำสั่งเงื่อนไข if(b) เป็นจริง สามารถจัดการเมท็อดคอลด้วยการสร้างแอสเป็กและแอดไวซ์ประเภทอะราวด์ (Around) เพื่อให้ทำงานแทนที่เมท็อด f() และกำหนดพอยท์คัท p โดยระบุตำแหน่งจอยน์พอยท์ไปยังการดำเนินการของเมท็อด f() พร้อมทั้งกำหนดเงื่อนไข if(This.b) ดังนั้นเมื่อซอฟต์แวร์ดำเนินการมาถึงเมท็อด f() และเงื่อนไข if(This.b) เป็นจริง โค้ดภายในแอดไวซ์ของแอสเป็กคือ เมท็อดคอล this.g() จึงทำงานแทนที่โค้ดภายในเมท็อด f()

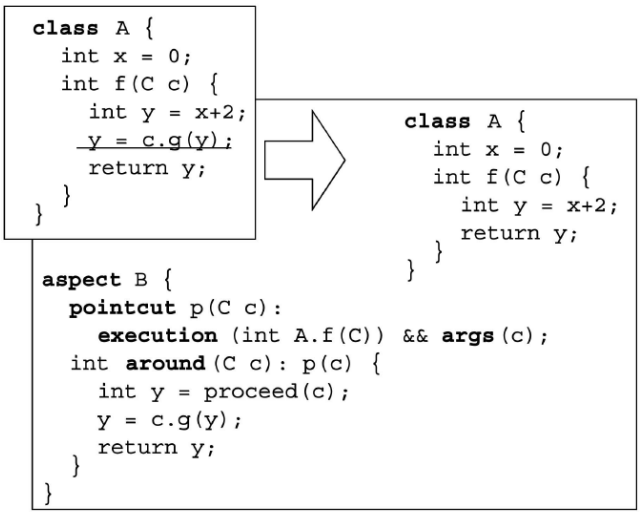
กรณีที่เปลี่ยนแปลงตำแหน่งของคำสั่ง x++ ให้อยู่ด้านนอกเงื่อนไข if(b) แสดงว่าไม่ว่าผลของเงื่อนไขเป็นจริงหรือเท็จต้องทำงานคำสั่ง x++ เสมอ แก้ไขได้ด้วยการเพิ่มคำสั่ง proceed() ไว้ในตอนท้ายของแอดไวซ์ เพื่อให้ดำเนินการคำสั่งที่มีในเมท็อด f()



รูปที่ 2.18 เทคนิคเอ็กซ์แทรคคอนดิชันนอล ของ Binkley และคณะ (2006)

(4) เทคนิคพรีรีเทิร์น (Pre Return)

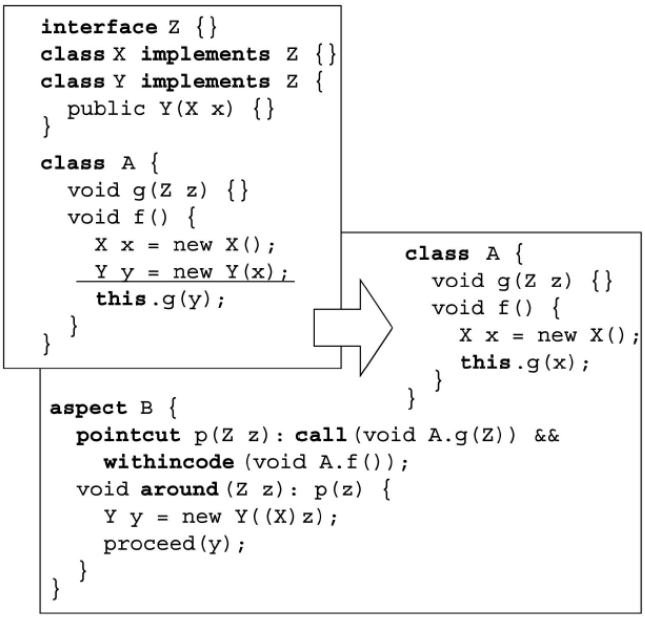
ตำแหน่งเมท็อดคอลที่ต้องการจัดการอยู่ก่อนคำสั่งรีเทิร์น (Return statement) จากรูปที่ 2.19 ภายในคลาส A และเมท็อด f() มีเมท็อดคอล y = c.g(y) โดยตำแหน่งอยู่ก่อนคำสั่งรีเทิร์น สามารถจัดการเมท็อดคอลด้วยการสร้างแอสเป็กและแอดไวซ์ประเภทอะราวด์ จากนั้นกำหนดพอยท์คัท p โดยระบุตำแหน่งจอยน์พอยท์ไปยังการดำเนินการเมท็อด f() ดังนั้นเมื่อซอฟต์แวร์ดำเนินการมาถึงเมท็อด f() โค้ดภายในแอดไวซ์ของแอสเป็กจึงทำงาน โดยมีคำสั่ง proceed(c) เพื่อให้ดำเนินการโค้ดภายในเมท็อด f() ก่อนและเมื่อดำเนินการเสร็จสิ้นจึงเก็บค่าด้วยตัวแปร y ของแอสเป็ก และนำตัวแปร y มาใช้ในเมท็อดคอล y = c.g(y) และส่งค่า y คืน ดังนั้นคำสั่งในแอดไวซ์จึงเปลี่ยนแปลงค่า y ที่รับมาจากเมท็อด f()



รูปที่ 2.19 เทคนิคพรีรีเทิร์น ของ Binkley และคณะ (2006)

(5) เทคนิคเอ็กซ์แทรคแรพเพอร์ (Extract Wrapper)

โค้ดที่ต้องการจัดการอยู่ในแพทเทิร์นแรพเพอร์ (Wrapper pattern) จากรูปที่ 2.20 ภายในคลาส A และเมท็อด f() จะมีโค้ด Y y = new Y(x) โดยอ็อบเจกต์ x จะถูก y ห่อหุ้มไว้ก่อนส่งไปเป็นพารามิเตอร์ในเมท็อดคอล this.g(y) สามารถจัดการด้วยการสร้างแอสเป็กและแอดไวซ์ประเภทอะราวด์ จากนั้นกำหนดพอยท์คัท p โดยระบุตำแหน่งจอยน์พอยท์ไปยังการดำเนินการเมท็อด g() ภายในแอดไวซ์กำหนดด้วยโค้ดที่ต้องการจัดการคือ Y y = new Y(x) และคำสั่ง proceed()



รูปที่ 2.20 เทคนิคเอ็กซ์แทรคแรพเพอร์ ของ Binkley และคณะ (2006)

(6) เทคนิคเอ็กซ์แทรคเอ็กซ์เซ็พชั่นแฮนด์ลิ่ง (Extract Exception Handling)

โค้ดที่ต้องการจัดการคือบล็อกของการจัดการข้อยกเว้น (Exception handling block) เช่นเดียวกับเทคนิคเอ็กซ์แทรคเอ็กซ์เซ็พชั่นแฮนด์ลิ่ง ของ Laddad (2003)

2.6 คุณภาพซอฟต์แวร์ (Software Quality)

การกำหนดนิยามของคำว่าคุณภาพ (Quality) เป็นเรื่องยากเนื่องจากการตัดสินคุณภาพของผลิตภัณฑ์หนึ่งๆขึ้นอยู่กับความคิดและความคาดหวังของแต่ละบุคคลที่ต้องการจากผลิตภัณฑ์ ดังนั้นคำว่าคุณภาพจึงมีความหมายและมุมมองที่แตกต่างกันไป โดยหลายงานวิจัยให้คำนิยามของคำว่าคุณภาพดังนี้

- Garvin (1984) แสดงนิยามของคุณภาพในมุมมองต่างๆ ดังนี้
  มุมมองของผู้ใช้งาน คุณภาพคือการที่ผลิตภัณฑ์ตรงกับวัตถุประสงค์ที่ผู้ใช้งานต้องการ
  มุมมองของผู้ผลิต คุณภาพคือการที่ผลิตภัณฑ์สามารถผลิตออกมาได้ตรงและสอดคล้องกับข้อกำหนดคุณลักษณะที่ต้องการ

มุมมองของผลิตภัณฑ์ นักเศรษฐศาสตร์จะพิจารณาคุณภาพของผลิตภัณฑ์โดยดูจากผลกระทบต่อค่าใช้จ่าย เช่น คุณภาพจะสูง หากค่าใช้จ่ายสูง เป็นต้น มุมมองของมูลค่าผลิตภัณฑ์ คุณภาพขึ้นอยู่กับจำนวนเงินที่ผู้ใช้เต็มใจที่จะจ่าย

- Kan (2002) กล่าวว่าคุณภาพเป็นแนวคิดหลายมุมมองที่ประกอบด้วยสิ่งที่สนใจและคุณลักษณะคุณภาพของสิ่งที่สนใจ ดังนั้นคนจึงให้นิยามคุณภาพแตกต่างกันตามสิ่งที่สนใจของแต่ละคน
- Stamelos และ Sfetsos (2007) กล่าวว่าคุณภาพประกอบด้วยกลุ่มลักษณะเฉพาะ (Feature) สำหรับตอบสนองต่อความต้องการของผู้ใช้และสร้างความพึงพอใจแก่ผู้ใช้ นอกจากนี้หากผลิตภัณฑ์มีคุณภาพดีจะไม่เกิดความบกพร่องในผลิตภัณฑ์
- ไอเอสโอ 8402 (ISO 8402) กล่าวว่าคุณภาพคือกลุ่มลักษณะเฉพาะของผลิตภัณฑ์หรือบริการที่ส่งผลต่อความพึงพอใจและความต้องการของผู้ใช้ (Panovski, 2008)

นอกจากนี้หลายงานวิจัยได้ให้คำนิยามของคำว่า คุณภาพซอฟต์แวร์ (Software quality) เอาไว้ดังนี้

- Ince (1994) กล่าวว่าคุณภาพซอฟต์แวร์สัมพันธ์กับกลุ่มปัจจัยคุณภาพ (Quality factor) เป็นสิ่งที่นักพัฒนาคิดว่ามีความสำคัญต่อซอฟต์แวร์ที่ผลิตแต่ผู้ใช้งานไม่ได้ตระหนักถึง ดังนั้นจึงควรนำกลุ่มปัจจัยคุณภาพมาพิจารณาร่วมด้วยในข้อกำหนดความต้องการ (Requirement Specification)
- Fitzpatrick (1996) กล่าวว่าคุณภาพซอฟต์แวร์คือกลุ่มลักษณะเฉพาะที่ต้องการในซอฟต์แวร์เพื่อช่วยเพิ่มสมรรถภาพของซอฟต์แวร์ให้มีอายุการใช้งานที่นาน (Lifetime performance)
- ไอเอสโอ/ไออีซี 9126 (ISO/IEC 9126) กล่าวว่าคุณภาพซอฟต์แวร์คือความสามารถของซอฟต์แวร์ที่สามารถตอบสนองความพึงพอใจของผู้ใช้ในการใช้งานตามที่กำหนด (ISO/IEC9126-1, 2001)
- Immonen (2009) กล่าวว่าคุณภาพซอฟต์แวร์คือความสามารถที่จะตอบสนองต่อความคาดหวังของผู้ใช้ โดยต้องใช้ทรัพยากรและมีระดับความซับซ้อนของระบบที่เหมาะสม

คุณภาพซอฟต์แวร์สามารถประเมินได้โดยประกอบด้วย 2 ปัจจัย (Elish and Alshayeb, 2012) ได้แก่

(1) คุณลักษณะภายนอกของคุณภาพ (External quality attribute) เป็นคุณลักษณะที่สามารถวัดได้โดยอ้อม โดยคุณลักษณะภายนอกส่วนใหญ่ผู้ใช้งานซอฟต์แวร์เป็นผู้พิจารณา เช่น ความสามารถการบำรุงรักษา และความน่าเชื่อถือ เป็นต้น

(2) คุณลักษณะภายในของคุณภาพ (Internal quality attribute) เป็นคุณลักษณะที่สามารถวัดได้โดยตรง โดยคุณลักษณะภายในส่วนใหญ่นักพัฒนาซอฟต์แวร์เป็นผู้พิจารณา ดังนั้นคุณลักษณะภายในจึงสามารถพิจารณาจากซอร์สโค้ดของผลิตภัณฑ์ซอฟต์แวร์ได้โดยตรง เช่น จำนวนบรรทัดของโค้ด เป็นต้น

2.6.1 แบบจำลองคุณภาพซอฟต์แวร์ (Software quality model)

เนื่องจากคุณภาพซอฟต์แวร์สามารถประเมินจากคุณลักษณะภายนอกหลายๆลักษณะประกอบกัน ดังนั้นเพื่อสร้างความเข้าใจและสามารถวัดคุณภาพของผลิตภัณฑ์ซอฟต์แวร์ จึงมีการสร้างแบบจำลองเพื่อแสดงความสัมพันธ์ระหว่างคุณลักษณะ (Characteristic) ดังนั้นแบบจำลองคุณภาพซอฟต์แวร์ (Software quality model) จึงแสดงกลุ่มคุณลักษณะและความสัมพันธ์ระหว่างคุณลักษณะ เพื่อใช้เป็นเกณฑ์สำหรับการกำหนดความต้องการทางคุณภาพและเพื่อประเมินคุณภาพซอฟต์แวร์ (ISO/IEC9126-1, 2001) แบบจำลองคุณภาพซอฟต์แวร์สามารถจำแนกได้เป็น 2 กลุ่ม ได้แก่ (1) แบบจำลองเชิงลำดับชั้น (Hierarchical quality model) และ (2) แบบจำลองไม่เป็นเชิงลำดับชั้น (Non-Hierarcjical quality model) โดยมีการนำเสนอแบบจำลองเชิงลำดับชั้นเป็นจำนวนมาก ได้แก่ แบบจำลองของ McCall (McCall, Richards and Walters, 1977) แบบจำลองของ Boehm (Boehm, Brown and Lipow, 1978) แบบจำลองคุณภาพ FURPS ที่นำเสนอโดย Grady และ HP (Grady and Caswell, 1987) และแบบจำลองคุณภาพไอเอสโอ/ไออีซี 9126 (ISO/IEC 9126) (ISO/IEC9126-1, 2001) เป็นต้น (Kumar, 2012)

แบบจำลองคุณภาพไอเอสโอ/ไออีซี 9126 เป็นแบบจำลองที่พัฒนาเพื่อเป็นมาตรฐานของแบบจำลองคุณภาพซอฟต์แวร์ เนื่องจากแบบจำลองที่นำเสนอในอดีตมีค่อนข้างมาก ไอเอสโอ/ไออีซี เจทีซี (ISO/IEC Joint Technical Committee : JTC) จึงพัฒนาแบบจำลองที่นำความคิดเห็นของคนส่วนใหญ่มาใช้เพื่อพัฒนาให้เกิดเป็นแบบจำลองที่สามารถใช้เป็นมาตรฐาน และพัฒนาขยายมาจากแบบจำลองเดิมที่นำเสนอไว้ โครงสร้างของแบบจำลองคุณภาพไอเอสโอ/ไออีซี 9126 ประกอบด้วยคุณลักษณะ (Characteristic) และคุณลักษณะย่อย (Sub-characteristic) โดยคุณลักษณะย่อยสามารถพิจารณาจากมาตรวัด (Metrics) หรือตัวชี้วัด (Indicator) และเนื่องจากแบบจำลองคุณภาพไอเอสโอ/ไออีซี 9126 เป็นแบบจำลองที่นำเสนอมาสำหรับพิจารณาซอฟต์แวร์เชิงวัตถุ ดังนั้น Kumar และคณะ (2009) จึงนำเสนอแบบจำลองคุณภาพเอโอเอสคิวยูเอเอ็มโอ (AOSQUAMO) เป็นแบบจำลองที่ปรับปรุงมาเพื่อพิจารณาซอฟต์แวร์เชิงแอสเป็ก โดยปรับปรุงมาจากแบบจำลองคุณภาพไอเอสโอ/ไออีซี 9126 โดยเพิ่มคุณลักษณะย่อยทั้งหมด 4 ลักษณะ ได้แก่ ความสามารถการนำกลับมาใช้ใหม่ (Reusability) ความซับซ้อน (Complexity) ความสามารถการลดจำนวนโค้ด (Code-reducibility) และสภาพเป็นส่วนจำเพาะ (Modularity) แบบจำลองคุณภาพเอโอเอสคิวยูเอเอ็มโอสามารถแสดงได้ดังตารางที่ 2.1

ตารางที่ 2.1 แบบจำลองคุณภาพเอโอเอสคิวยูเอเอ็มโอ (AOSQUAMO)
ของ Kumar และคณะ (2009)

ประเภทคุณภาพ	คุณลักษณะ	คุณลักษณะย่อย
Quality Software Product	Functionality	Suitability
		Accuracy
		Interoperability
		Security
		Compliance
		Reusability
	Reliability	Maturity
		Fault Tolerance
		Recoverability
	Usability	Understandability
		Learnability
		Operability
		Complexity
	Efficiency	Time Behavior
		Resource Utilization
		Code-reducibility
	Maintainability	Analyzability
		Changeability
		Stability
		Testability
		Modularity
	Portability	Adaptability
		Replaceability
		Installability
		Co-Existence

แบบจำลองคุณภาพเอโอเอสคิวยูเอเอ็มโอมีคุณลักษณะทั้งหมด 6 ลักษณะ ดังนี้

(1) ฟังก์ชันการทำงาน (Functionality) คือ ความสามารถของผลิตภัณฑ์ซอฟต์แวร์ที่รองรับฟังก์ชัน โดยสามารถตอบสนองต่อความต้องการเมื่อใช้งานซอฟต์แวร์ภายใต้สภาพแวดล้อมหรือเงื่อนไขที่ระบุ

(2) ความเชื่อถือได้ (Reliability) คือ ความสามารถของผลิตภัณฑ์ซอฟต์แวร์ที่สามารถคงระดับสมรรถภาพของซอฟต์แวร์ โดยซอฟต์แวร์สามารถทำงานสำเร็จบรรลุผลได้ภายใต้เงื่อนไขและช่วงเวลาที่ระบุ

(3) ความสามารถการใช้งานง่าย (Usability) คือ ความสามารถของผลิตภัณฑ์ซอฟต์แวร์ในการทำความเข้าใจ การเรียนรู้และใช้งานซอฟต์แวร์ รวมถึงสามารถดึงดูดความสนใจของผู้ใช้งานได้เมื่อใช้งานภายใต้เงื่อนไขที่ระบุ หรือความพยายาม (Effort) ที่ต้องการเพื่อใช้งานผลิตภัณฑ์ซอฟต์แวร์

(4) ประสิทธิภาพ (Efficiency) คือ ความสามารถของผลิตภัณฑ์ซอฟต์แวร์ที่มีสมรรถภาพผู้ใช้งานสามารถใช้งานซอฟต์แวร์ได้สำเร็จบรรลุผลด้วยจำนวนทรัพยากรที่เหมาะสมภายใต้เงื่อนไขที่กำหนด

(5) ความสามารถการบำรุงรักษา (Maintainability) คือ ความสามารถของผลิตภัณฑ์ซอฟต์แวร์ที่สามารถแก้ไขได้ โดยการแก้ไขซอฟต์แวร์จะต้องมีความถูกต้อง การปรับปรุงที่ทำให้ซอฟต์แวร์ดีขึ้น และสามารถปรับตัวรองรับหากเกิดการเปลี่ยนแปลงของสภาพแวดล้อม ความต้องการ และข้อกำหนดเชิงฟังก์ชัน

(6) ความสามารถการถ่ายโอน (Portability) คือ ความสามารถของผลิตภัณฑ์ซอฟต์แวร์ที่สามารถเปลี่ยนย้ายซอฟต์แวร์จากสภาพแวดล้อมหนึ่งไปยังสภาพแวดล้อมอื่นๆได้

แบบจำลองคุณภาพเอโอเอสคิวยูเอเอ็มโอมีคุณลักษณะย่อยทั้งหมด 25 ลักษณะซึ่งกำหนดเข้ากับคุณลักษณะดังตารางที่ 2.1 แสดงข้างต้น โดยขอยกนิยามของคุณลักษณะย่อยมาเพียงบางส่วนที่สนใจศึกษาในงานวิจัยนี้ ได้แก่

(1) ความสามารถการนำกลับมาใช้ใหม่ (Reusability) คือ ความสามารถของผลิตภัณฑ์ซอฟต์แวร์ที่สามารถนำฟังก์ชันและโค้ดที่มีอยู่กลับมาใช้ใหม่

(2) ความสามารถในการทำความเข้าใจ (Understandability) คือ ความสามารถของผลิตภัณฑ์ซอฟต์แวร์ที่ผู้ใช้สามารถทำความเข้าใจถึงวิธีการใช้งานได้อย่างเหมาะสมสำหรับแต่ละภารกิจและเงื่อนไขการใช้

(3) ความซับซ้อน (Complexity) คือ ความซับซ้อนของซอฟต์แวร์ สามารถจำแนกเป็นความซับซ้อนของโค้ด (Code complexity) เป็นความซับซ้อนที่เกิดจากแอตทริบิวต์ เมท็อด และแอดไวซ์ เป็นต้น และความซับซ้อนของการปฏิสัมพันธ์ (Interaction complexity) เป็นความซับซ้อนที่เกิดจากการปฏิสัมพันธ์ระหว่างคลาส-คลาส แอสเป็ก-แอสเป็ก และคลาส-แอสเป็ก

(4) ความสามารถการทดสอบ (Testability) คือ ความสามารถของผลิตภัณฑ์ซอฟต์แวร์ที่สามารถนำมาตรวจสอบความสมเหตุสมผล

2.6.2 มาตรวัดซอฟต์แวร์ (Software metric)

จากแบบจำลองคุณภาพซอฟต์แวร์ที่แสดงถึงคุณลักษณะภายนอกและความสัมพันธ์ระหว่างคุณลักษณะในรูปแบบของแบบจำลองเชิงลำดับชั้น Fenton และ Pfleeger (1998) กล่าวไว้ว่ามาตรวัดซอฟต์แวร์สามารถใช้วัดคุณลักษณะภายในของคุณภาพเพื่อประเมินคุณลักษณะภายนอกได้ (Elish and Alshayeb, 2012) ดังนั้นเพื่อประเมินคุณภาพของผลิตภัณฑ์ซอฟต์แวร์จึงนำมาตรวัดซอฟต์แวร์มาใช้สำหรับวัดคุณภาพ โดยมาตรวัดซอฟต์แวร์มีหลายงานวิจัยให้คำนิยามเอาไว้ดังนี้

Mills (1988) กล่าวว่ามาตรวัดซอฟต์แวร์สามารถนำมาใช้วัดผลิตภัณฑ์ซอฟต์แวร์และกระบวนการของซอฟต์แวร์ เพื่อช่วยประมาณค่าใช้จ่าย ตารางงาน และคุณภาพของซอฟต์แวร์ ข้อมูลที่ได้จากมาตรวัดซอฟต์แวร์สามารถนำไปจัดการและควบคุมการพัฒนาซอฟต์แวร์ให้ได้ซอฟต์แวร์ที่ปรับปรุงดีขึ้น

Ordonez และ Hadded (2008) กล่าวว่ามาตรวัดซอฟต์แวร์เป็นตัวชี้วัดเชิงปริมาณที่สามารถบ่งบอกถึงคุณลักษณะของส่วนโปรแกรม ระบบ และกระบวนการได้

มาตรวัดซอฟต์แวร์สามารถจำแนกออกเป็นมาตรวัดสำหรับผลิตภัณฑ์ (Product metrics) และมาตรวัดสำหรับกระบวนการ (Process metrics) โดยมาตรวัดสำหรับผลิตภัณฑ์จะสามารถวัดผลิตภัณฑ์ซอฟต์แวร์ที่ได้จากในขั้นตอนต่างๆของการพัฒนาซอฟต์แวร์ เช่น ความต้องการ (Requirement) หรือซอฟต์แวร์ที่นำไปติดตั้ง เป็นต้น ส่วนมาตรวัดสำหรับกระบวนการสามารถวัดกระบวนการที่เกิดขึ้นระหว่างการพัฒนาซอฟต์แวร์ เช่น เวลาที่ใช้พัฒนาโดยรวม ประเภทของระเบียบวิธีที่ใช้ และระดับประสบการณ์ของผู้พัฒนา เป็นต้น โดยประเภทมาตรวัดที่สนใจในงานวิจัยนี้คือมาตรวัดสำหรับผลิตภัณฑ์เพื่อนำไปวัดคุณภาพของผลิตภัณฑ์ซอฟต์แวร์ เนื่องจากการพัฒนาด้วยการโปรแกรมเชิงแอสเป็กเป็นการพัฒนาที่ขยายมาจากการโปรแกรมเชิงวัตถุ มาตรวัดซอฟต์แวร์ที่สนใจจึงได้แก่ มาตรวัดเชิงวัตถุ (Object-oriented metrics – OO metrics) และมาตรวัดเชิงแอสเป็ก (Aspect-oriented metrics – AO metrics) มีรายละเอียดดังนี้

- **มาตรวัดเชิงวัตถุ (Object-oriented metrics – OO metrics)**

การวัดคุณภาพซอฟต์แวร์สำหรับซอฟต์แวร์ที่พัฒนาด้วยการโปรแกรมเชิงวัตถุ Chidamber และ Kemerer (1993) ได้นำเสนอชุดมาตรวัดซีเค (CK metrics) สำหรับวัดคุณภาพของการออกแบบเชิงวัตถุ โดยมาตรวัดซอฟต์แวร์ที่นำเสนอมีดังนี้ (Aggarwal et al., 2006; Rosenberg and Hyatt, 1996)

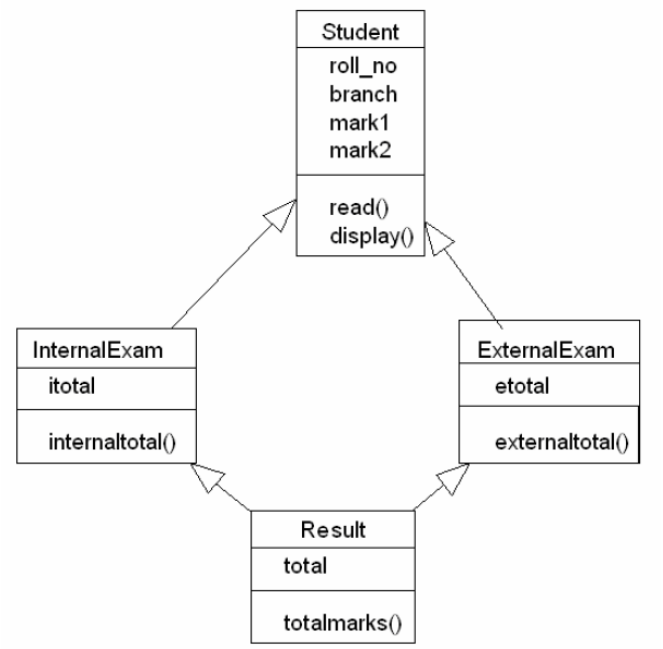
(1) เวทเทดเมท็อดเปอคลาส (Weighted Method per Class - WMC)

มาตรวัด WMC คือจำนวนเมท็อดที่อิมพลีเม้นต์ภายในคลาส หรือผลรวมความซับซ้อนของทุกเมท็อดภายในคลาส โดยความซับซ้อนสามารถคำนวณจากค่าความซับซ้อนไซโคลเมตริก (Cyclometric complexity) การวัดด้วยค่าความซับซ้อนไซโคลเมตริกในทางปฏิบัติทำได้ยาก เนื่องจากไม่สามารถเข้าถึงทุกเมท็อดได้ทั้งหมดภายในลำดับชั้นคลาส ดังนั้นในทางปฏิบัติจึงกำหนดค่าความซับซ้อนสำหรับเมท็อดเท่ากับ 1.0 มาตรวัด WMC จึงเท่ากับจำนวนเมท็อดที่มีภายในคลาส

มาตรวัด WMC แสดงถึงความซับซ้อนของคลาส ทำให้สามารถประเมินจำนวนเวลาและความพยายามที่ต้องใช้สำหรับการพัฒนาและบำรุงรักษาคลาส และหากมีจำนวนเมท็อดภายในคลาสมากจะส่งผลต่อคลาสลูกที่สืบทอดเมท็อดจากคลาสแม่

(2) เดพออฟอินเฮอริแทนทรี (Depth of Inheritance Tree - DIT)

มาตรวัด DIT วัดระดับความลึกของคลาสภายในลำดับชั้นการรับทอด (Inheritance hierarchy) โดยพิจารณาจำนวนระดับที่มากสุดจากคลาสโหนด (Class node) ไปยังรูทโหนด (Root node) ของทรี (Tree) จากรูปที่ 2.21 แสดงตัวอย่างการพิจารณามาตรวัด DIT โดยคลาส Result มีค่ามาตรวัด DIT เท่ากับ 2 ส่วนคลาส InternalExam และคลาส ExternalExam มีค่ามาตรวัด DIT เท่ากับ 1



รูปที่ 2.21 ตัวอย่างลำดับชั้นการรับทอด ของ Aggarwal และคณะ (2006)

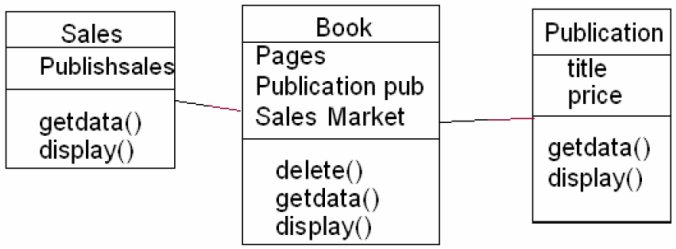
หากค่ามาตรวัด DIT มีค่าสูงสามารถแสดงถึงระดับความลึกและความซับซ้อนที่มาก เนื่องจากจำนวนคลาสที่เกี่ยวข้องและจำนวนเมท็อดที่รับทอดมีจำนวนมาก โดยจำนวนเมท็อดที่รับทอดมามีจำนวนเพิ่มมากขึ้นตามระดับความลึกภายในลำดับชั้น แต่การมีระดับความลึกภายในลำดับชั้นที่มากสามารถแสดงถึงความสามารถการนำกลับมาใช้ใหม่ของเมท็อดที่ดีเช่นกัน

(3) นัมเบอร์ออฟชิลเดรน (Number of Children - NOC)

มาตรวัด NOC วัดจำนวนคลาสลูก (Subclass) ที่รับทอดเมท็อดมาจากคลาสภายในลำดับชั้น จากรูปที่ 2.21 ข้างต้น แสดงตัวอย่างการพิจารณามาตรวัด NOC โดยคลาส Student มีค่ามาตรวัด NOC เท่ากับ 2 กรณีที่มีจำนวนคลาสลูกมากอาจส่งผลต่อการใช้ความพยายามและเวลาสำหรับการทดสอบเมท็อดของคลาสมาก หรือแสดงว่าการกำหนดคลาสไม่เหมาะสม โดยคลาสลูกบางคลาสอาจไม่ถูกเรียกใช้งาน การมีจำนวนคลาสลูกมากสามารถส่งผลดีต่อความสามารถการนำกลับมาใช้ใหม่

(4) คัพพลิงบีทวีนอ็อบเจกต์คลาส (Coupling Between Object classes - CBO)

มาตรวัด CBO วัดจำนวนคลาสที่สัมพันธ์กับคลาสที่พิจารณา สามารถพิจารณาความสัมพันธ์ระหว่างคลาสจากการเรียกใช้งานเมท็อดของคลาสอื่นหรือมีการประกาศตัวแปรอินสแตนซ์ (Instance Variable) ดังรูปที่ 2.22 แสดงตัวอย่างการพิจารณามาตรวัด CBO โดยคลาส Book มีค่ามาตรวัด CBO เท่ากับ 2 เนื่องจากภายในคลาส Book ประกาศตัวแปรอินสแตนซ์ของคลาส Sales และคลาส Publication



รูปที่ 2.22 ตัวอย่างคลาสไดอะแกรม ของ Aggarwal และคณะ (2006)

หากค่ามาตรวัด CBO มีค่าสูงแสดงว่ามีคัพพลิง (Coupling) สูงเช่นกัน ส่งผลต่อความสามารถการนำกลับมาใช้ใหม่ในซอฟต์แวร์อื่นทำได้ยาก และการแก้ไขซอฟต์แวร์สามารถส่งผลกระทบต่อคลาสอื่นได้มาก ความสามารถการบำรุงรักษาจึงทำได้ยากเช่นกัน อีกทั้งยังส่งผลให้เกิดความซับซ้อนและยากต่อการทำความเข้าใจเนื่องจากมีคลาสที่มาเกี่ยวข้องจำนวนมาก

(5) เรสปอนซ์ฟอร์อะคลาส (Response For a Class - RFC)

มาตรวัด RFC วัดความซับซ้อนของคลาสจากจำนวนการสื่อสารระหว่างคลาสอื่นๆ โดยพิจารณาจากจำนวนเมท็อดทั้งหมดที่ทำงานเพื่อตอบสนองต่อสาร (Message) ที่ได้รับมาจากอ็อบเจกต์ของคลาสหรือจากบางเมท็อดภายในคลาส รวมถึงเมท็อดทั้งหมดที่สามารถเข้าถึงได้ภายในลำดับชั้นคลาส มาตรวัด RFC = |RS| สามารถแสดงด้วยสมการดังนี้

$$RS = \{M_i\} \cup all\ j\{R_{ij}\}$$

โดย M_i คือเซตของเมท็อดทั้งหมดภายในคลาส

R_{ij} คือเซตของเมท็อดที่ถูกเรียกใช้งานจาก M_i

จากรูปที่ 2.22 ข้างต้น แสดงตัวอย่างการพิจารณามาตรวัด RFC โดยภายในคลาส Book เมท็อด getdata() และ display() ได้เรียกใช้งานเมท็อดของคลาส Sales และ Publication ด้วย ดังนั้นค่ามาตรวัด RFC จึงมีค่าเท่ากับ 7 หากมีเมท็อดจำนวนมากที่ทำงานตอบสนองต่อคลาสสามารถแสดงถึงความซับซ้อนของคลาสที่มากเช่นกัน รวมถึงการทำความเข้าใจที่ยากและต้องใช้เวลาในการทดสอบนาน

(6) แลคออฟโคฮีชันอินเมท็อด (Lack of Cohesion in Methods - LCOM)

มาตรวัด LCOM วัดระดับความเกี่ยวข้องกันระหว่างเมท็อดภายในคลาส โดยพิจารณาจากแอตทริบิวต์หรือตัวแปรอินสแตนซ์ที่นำไปใช้ในเมท็อดของคลาส หากเมท็อดมีความเกี่ยวข้องกันมากมักใช้ชุดตัวแปรเดียวกัน ค่ามาตรวัด LCOM สามารถพิจารณาได้ดังสมการ ดังนี้

$$LCOM = |P| - |Q|, \text{ if } |P| > |Q|$$

โดย P คือ $\{(I_i, I_j) \mid I_i \cap I_j = 0\}$ จำนวนเซตอินเตอร์เซคมีค่าเป็น 0

Q คือ $\{(I_i, I_j) \mid I_i \cap I_j \neq 0\}$ จำนวนเซตอินเตอร์เซคมีค่าไม่เป็น 0

I คือ เซตของตัวแปรอินสแตนซ์ที่ใช้ในเมท็อดหนึ่งๆ

กำหนดตัวอย่างให้มีเมท็อดทั้งหมด 4 เมท็อด โดยแต่ละเมท็อดมีเซตของตัวแปรอินสแตนซ์ที่ใช้ในเมท็อดดังนี้

I_1 = {book_id, pub_id, book_name, author_name, price}

I_2 = {book_id}

I_3 = {book_name}

I_4 = {author_name}

ค่ามาตรวัด LCOM ของคลาสมีค่าเท่ากับ 0 เนื่องจาก P และ Q มีค่าเป็น 3 (LCOM = |3|-|3|) โดย P เป็นเซตอินเตอร์เซคที่มีค่าเป็น 0 ได้แก่ $I_2 \cap I_3$, $I_2 \cap I_4$ และ $I_3 \cap I_4$ ส่วน P เป็นเซตอินเตอร์เซคที่มีค่าไม่เป็น 0 ได้แก่ $I_1 \cap I_2$, $I_1 \cap I_3$ และ $I_1 \cap I_4$

หากค่ามาตรวัด LCOM มีค่าสูงแสดงว่ามีโคฮีชัน (Cohesion) ต่ำ คลาสที่มีค่ามาตรวัด LCOM สูงสามารถปรับปรุงได้ด้วยการแบ่งคลาสให้เล็กลงเพื่อเพิ่มโคฮีชัน แต่หากมีค่ามาตรวัด LCOM ต่ำจะแสดงถึงการห่อหุ้มที่ดี (Encapsulation)

- **มาตรวัดเชิงแอสเป็ก (Aspect-oriented metrics – AO metrics)**

การโปรแกรมเชิงแอสเป็กนำมาใช้เพื่อแก้ไขปัญหาโค้ดกระจัดกระจายและโค้ดพันกันที่เกิดจากการพัฒนาด้วยการโปรแกรมเชิงวัตถุ ดังนั้นการพัฒนาด้วยการโปรแกรมเชิงแอสเป็กจึงยังคงการอิมพลีเม้นต์ฟังก์ชันหลักด้วยการโปรแกรมเชิงวัตถุ และนำโครงสร้างของแอสเป็กเข้ามาปรับปรุงซอร์สโค้ด มาตรวัดเชิงแอสเป็กส่วนใหญ่จึงมีการปรับปรุงแนวคิดมาจากมาตรวัดเชิงวัตถุ โดยมีหลายงานวิจัยนำเสนอมาตรวัดเอาไว้ ได้แก่

Zhao (2002) นำเสนอมาตรวัดสำหรับวัดความซับซ้อนของซอฟต์แวร์เชิงแอสเป็กโดยพิจารณาจากความสัมพันธ์ของการขึ้นต่อกัน (Program dependence relation) สามารถพิจารณาในหลายระดับทั้งระดับโมดูล แอสเป็ก และระบบ

Sant'Anna และคณะ (2003) เสนอกรอบแนวคิดสำหรับประเมินคุณภาพ โดยมาตรวัดที่นำเสนอสามารถจำแนกเป็นมาตรวัดเกี่ยวกับการแยกคอนเซิร์น (Seperation of concern) คัพพลิง (Coupling) โคฮีชัน (Cohesion) และขนาด (Size)

ส่วน Ceccato และ Tonella (2004) นำเสนอชุดมาตรวัดที่ปรับปรุงจากมาตรวัดเชิงวัตถุของ Chidamber และ Kemerer (1993) โดยเพิ่มเติมเกี่ยวกับนิยามแอสเป็กเพื่อให้สามารถนำมาใช้ประเมินซอฟต์แวร์เชิงแอสเป็ก โดยกำหนดให้มอดูล (Module) เป็นคำแทนคลาสและแอสเป็ก ส่วน

โอเปอร์เรชัน (Operation) เป็นคำแทนเมท็อด แอดไวซ์ และอินโทรดักชัน มาตรวัดเชิงแอสเป็กมีทั้งหมด 10 มาตรวัด (Ceccato and Tonella, 2004) ดังนี้

(1) เวทเทดโอเปอร์เรชั่นอินมอดูล (Weighted Operations in Module - WOM) คือ จำนวนโอเปอเรชั่นภายในมอดูล ได้แก่ จำนวนเมท็อด จำนวนแอดไวซ์ และจำนวนอินโทรดักชันที่มีภายในคลาสหรือแอสเป็ก โดยมาตรวัด WOM สามารถนำมาพิจารณาความซับซ้อนของคลาสและแอสเป็ก

(2) เดพออฟอินเฮอริแทนทรี (Depth of Inheritance Tree - DIT) คือระดับความลึกจากมอดูลถึงคลาสหรือแอสเป็กที่เป็นรูทในลำดับชั้น ดังนั้นจึงสามารถนับจากจำนวนชั้นการรับทอดจากมอดูลไปยังรูท

(3) นัมเบอร์ออฟชิลเดรน (Number Of Children - NOC) คือจำนวนคลาสลูกหรือจำนวนซับแอสเป็กของมอดูลที่พิจารณา

(4) ครอสคัทติ้งดีกรีออฟแอนแอสเป็ก (Crosscutting Degree of an Aspect - CDA) คือ จำนวนมอดูลทั้งหมดที่ได้รับผลกระทบจากพอยท์คัทและอินโทรดักชันที่กำหนดภายในแอสเป็ก หรือจำนวนมอดูลทั้งหมดที่ถูกขัดขวางการทำงานจากแอสเป็กที่พิจารณา

(5) คัพพลิงออนแอดไวซ์เอ็กซิคิวชั่น (Coupling on Advice Execution - CAE) คือจำนวนแอสเป็กทั้งหมดที่ภายในมีแอดไวซ์ที่ทำงานเมื่อการดำเนินการของโอเปอเรชั่นเกิดขึ้นภายในมอดูลที่พิจารณา หรือจำนวนแอสเป็กทั้งหมดที่สามารถเข้ามาขัดขวางการทำงานของมอดูลที่พิจารณา

(6) คัพพลิงออนเมท็อดคอล (Coupling on Method Call - CMC) คือจำนวนมอดูลทั้งหมดที่มีโอเปอเรชั่นที่สามารถเรียกใช้งานได้จากมอดูลที่พิจารณา

(7) คัพพลิงออนฟิลด์แอคเซส (Coupling on Field Access - CFA) คือจำนวนมอดูลทั้งหมดที่มีฟิลด์ (Field) ที่สามารถเข้าถึงได้จากมอดูลที่พิจารณา

(8) คัพพลิงบีทวีนมอดูล (Coupling between Modules - CBM) คือจำนวนความสัมพันธ์ระหว่างมอดูล โดยพิจารณาความสัมพันธ์จากการเรียกใช้งานโอเปอเรชั่นและการเข้าถึงฟิลด์ของมอดูลอื่น

(9) เรสปอนซ์ฟอร์อะมอดูล (Response For a Module - RFM) คือจำนวนเมท็อดและแอดไวซ์ที่ทำงานเพื่อตอบสนองต่อสาร (Message) ที่ได้รับจากมอดูลที่กำหนด

(10) แลคออฟโคฮีชันอินโอเปอร์เรชั่น (Lack of Cohesion in Operations - LCO) คือ ความสัมพันธ์ระหว่างโอเปอเรชั่นภายในมอดูล พิจารณาจากจำนวนคู่ของโอเปอเรชั่นที่เข้าถึงฟิลด์ของคลาสและแอสเป็กแตกต่างกันหักลบกับจำนวนคู่ของโอเปอเรชั่นที่เข้าถึงฟิลด์เดียวกัน

เนื่องจากมาตรวัดเชิงแอสเป็กของ Ceccato และ Tonella (2004) เป็นชุดมาตรวัดที่ปรับปรุงจากมาตรวัดเชิงวัตถุของ Chidamber และ Kemerer (1993) สามารถเปรียบเทียบนิยามระหว่างมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็กที่มีนิยามคล้ายกันได้ดังตารางที่ 2.2

ตารางที่ 2.2 ข้อแตกต่างของนิยามระหว่างมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็ก

ลำดับ	มาตรวัดเชิงวัตถุ/ มาตรวัดเชิงแอสเป็ก	นิยามมาตรวัด	
		ความแตกต่าง	ความเหมือน
1	Weighted Method per Class (WMC)	-	จำนวนเมท็อดที่อิมพลีเม้นต์ภายในคลาส
	Weighted Operations in Module (WOM)	จำนวนแอดไวซ์ที่อิมพลีเม้นต์ภายในแอสเป็ก	
2	Depth of Inheritance Tree (DIT)	-	ระดับความลึกของคลาสภายในลำดับชั้นการรับทอด
	Depth of Inheritance Tree (DIT)	ระดับความลึกของแอสเป็กภายในลำดับชั้นการรับทอด	
3	Number of Children (NOC)	-	จำนวนคลาสลูก
	Number Of Children (NOC)	จำนวนซับแอสเป็ก	
4	Coupling Between Object classes (CBO)	ความสัมพันธ์ระหว่างคลาสที่เกิดจากการประกาศตัวแปรอินสแตนซ์ของคลาสอื่น	ความสัมพันธ์ระหว่างคลาสที่เกิดจากการเรียกใช้งานเมท็อดของคลาสอื่น
	Coupling on Method Call (CMC)	-	
5	Coupling Between Object classes (CBO)	ความสัมพันธ์ระหว่างคลาสที่เกิดจากการประกาศตัวแปรอินสแตนซ์ของคลาสอื่น	ความสัมพันธ์ระหว่างคลาสที่เกิดจากการเรียกใช้งานเมท็อดของคลาสอื่น
	Coupling between Modules (CBM)	ความสัมพันธ์ระหว่างคลาสหรือแอสเป็กที่เกิดจากการเข้าถึงฟิลด์ของมอดูลอื่น	
6	Response For a Class (RFC)	-	จำนวนเมท็อดที่ทำงานตอบสนองต่อคลาส
	Response For a Module (RFM)	จำนวนแอดไวซ์ที่เข้ามาขัดขวางการทำงานคลาส	
7	Lack of Cohesion in Methods (LCOM)	-	ความเกี่ยวข้องกันระหว่างเมท็อดภายในคลาสจากการเข้าถึงฟิลด์ของคลาส
	Lack of Cohesion in Operations (LCO)	ความเกี่ยวข้องกันระหว่างแอดไวซ์ภายในแอสเป็กจากการเข้าถึงฟิลด์ของแอสเป็ก	

จากมาตรวัดเชิงวัตถุของ Chidamber และ Kemerer (1993) ทั้งหมด 6 มาตรวัด สามารถใช้วัดคุณลักษณะภายในของคุณภาพเพื่อประเมินคุณลักษณะภายนอกของซอฟต์แวร์ได้ สำหรับงานวิจัยนี้วิเคราะห์ความสัมพันธ์ระหว่างมาตรวัดเชิงวัตถุกับคุณภาพซอฟต์แวร์ โดยอ้างอิงจากงานวิจัยของ Kulkarni, Kalshetty และ Arde (2010) ที่พิจารณาคุณภาพซอฟต์แวร์ ได้แก่ ความสามารถในการทำความเข้าใจ (Understandability) ความสามารถการบำรุงรักษา (Maintainability) ความสามารถการนำกลับมาใช้ใหม่ (Reusability) และความสามารถการทดสอบ (Testability) สามารถแสดงความสัมพันธ์ได้ดังตารางที่ 2.3

ตารางที่ 2.3 ความสัมพันธ์ระหว่างมาตรวัดเชิงวัตถุและคุณภาพซอฟต์แวร์ ของ Kulkarni, Kalshetty และ Arde (2010)

คุณภาพซอฟต์แวร์	**มาตรวัดเชิงวัตถุของ Chidamber และ Kemerer (1993)**					
	Weighted Method per Class (WMC)	Depth of Inheritance Tree (DIT)	Number of Children (NOC)	Coupling Between Object classes (CBO)	Response For a Class (RFC)	Lack of Cohesion in Methods (LCOM)
Understandability	-	✓	-	✓	✓	-
Maintainability	✓	-	-	✓	-	-
Reusability	✓	✓	✓	✓	-	-
Testability	-	-	✓	✓	✓	-

จากมาตรวัดเชิงแอสเป็กของ Ceccato และ Tonella (2004) ทั้งหมด 10 มาตรวัด สามารถใช้วัดคุณลักษณะภายในของคุณภาพเพื่อประเมินคุณลักษณะภายนอกของซอฟต์แวร์ได้เช่นกัน สำหรับงานวิจัยนี้วิเคราะห์ความสัมพันธ์ระหว่างมาตรวัดเชิงแอสเป็กกับคุณภาพซอฟต์แวร์ โดยอ้างอิงจากงานวิจัยของ Sirbi และ Kulkarni (2013) ที่แสดงความสัมพันธ์เพียงบางมาตรวัด ได้แก่ มาตรวัด WOM CBM RFM และ LCO ดังตารางที่ 2.4

ตารางที่ 2.4 ความสัมพันธ์ระหว่างมาตรวัดเชิงแอสเป็กและคุณภาพซอฟต์แวร์ ของ Sirbi และ Kulkarni (2013)

คุณภาพซอฟต์แวร์	**มาตรวัดเชิงแอสเป็ก ของ Ceccato และ Tonella (2004)**			
	Weighted Operations in Module (WOM)	Coupling between Modules (CBM)	Response For a Module (RFM)	Lack of Cohesion in Operations (LCO)
Efficiency	-	-	-	✓
Complexity	✓	-	-	-
Understandability	-	-	✓	-
Maintainability	✓	✓	-	-
Reusability	-	✓	-	✓
Testability	-	-	✓	-

2.7 งานวิจัยที่เกี่ยวข้อง

สำหรับการเปรียบเทียบคุณภาพของซอฟต์แวร์เชิงแอสเป็กมีงานวิจัยในอดีตที่เกี่ยวข้องดังนี้

Kulesza และคณะ (2006) ศึกษาการเปรียบเทียบคุณภาพซอฟต์แวร์ด้านความสามารถการบำรุงรักษาของซอฟต์แวร์ที่พัฒนาด้วยการโปรแกรมเชิงวัตถุและซอฟต์แวร์ที่พัฒนาด้วยการโปรแกรมเชิงแอสเป็ก ภาษาที่ใช้พัฒนาคือภาษาจาวาและภาษาแอสเป็กเจตามลำดับ หน่วยตัวอย่างที่ใช้ศึกษาคือ เฮลท์วอทเชอร์ (HealthWatcher) โดยการพัฒนาด้วยการโปรแกรมเชิงแอสเป็กมีการจัดการครอสคัทติ้งคอนเซิร์น ได้แก่ ดิสทริบิวชั่น (Distribution) เพอร์ซิสเทนซ์ (Persistence) และคอนเคอร์เรนซี (Concurrency) การทดลองจึงเปรียบเทียบความสามารถการบำรุงรักษาของทั้งสองซอฟต์แวร์ โดยปรับปรุงซอฟต์แวร์ตามเหตุการณ์ (scenario) ที่กำหนด และประเมินคุณภาพจากมาตรวัดซอฟต์แวร์ที่สามารถแบ่งมาตรวัดตามคุณลักษณะของซอฟต์แวร์ดังนี้ การแยกคอนเซิร์น (Separation of concern) คัพพลิง (Coupling) โคฮีชัน (Cohesion) และ ขนาด (Size) จากการทดลองได้แสดงผลมาตรวัดทั้งก่อนและหลังการบำรุงรักษาของซอฟต์แวร์ตามเหตุการณ์ที่กำหนด พบว่าคุณภาพโดยรวมของซอฟต์แวร์ที่พัฒนาด้วยการโปรแกรมเชิงแอสเป็กมีคุณภาพดีกว่าซอฟต์แวร์ที่พัฒนาด้วยการโปรแกรมเชิงวัตถุ เนื่องจากการพัฒนาด้วยการโปรแกรมเชิงแอสเป็กแสดงให้เห็นว่าจำนวนบรรทัดของโค้ดลดลง การแยกคอนเซิร์นของซอฟต์แวร์ปรับปรุงดีขึ้น คัพพลิงระหว่างโมดูลลดลง และความซับซ้อนภายในส่วนของโปรแกรมลดลง การพัฒนาซอฟต์แวร์ด้วยการโปรแกรมเชิงแอสเป็กจึงสามารถให้คุณภาพด้านความสามารถการบำรุงรักษาที่ดีกว่าการพัฒนาด้วยการโปรแกรมเชิงวัตถุ

Mguni และ Ayalew (2013) ศึกษาการเปรียบเทียบคุณภาพซอฟต์แวร์ด้านความสามารถการบำรุงรักษา โดยเปรียบเทียบระหว่างซอฟต์แวร์ที่พัฒนาด้วยการโปรแกรมเชิงวัตถุและการโปรแกรมเชิงแอสเป็กเช่นกัน แต่งานวิจัยของ Mguni และ Ayalew (2013) มุ่งเน้นศึกษาการพัฒนาของระบบสำเร็จรูป (COTS-based system) โดยหน่วยตัวอย่างที่ใช้ศึกษาคือ โอเพ่นบราโวพีโอเอส (OpenBravoPOS) เป็นตัวอย่างระบบสำเร็จรูปเกี่ยวข้องกับธุรกิจขายปลีก และการประเมินคุณภาพใช้ชุดมาตรวัดสองชุด ได้แก่ ชุดมาตรวัดระดับคอนเซิร์น (Concern level metrics) และชุดมาตรวัดความซับซ้อนทางโครงสร้าง (Structural Complexity Metrics) โดยชุดมาตรวัดความซับซ้อนทางโครงสร้างเป็นชุดมาตรวัดที่นำมาใช้เช่นเดียวกันกับในงานวิจัย การปรับปรุงโอเพ่นบราโวพีโอเอสด้วยการโปรแกรมเชิงแอสเป็กเลือกจัดการครอสคัทติ้งคอนเซิร์น ได้แก่ การจัดการช่วงเวลาสื่อสาร (Session management) การบันทึกล็อก (Logging) และการจัดการข้อยกเว้น (Exceptional handling) จากนั้นเก็บรวบรวมค่ามาตรวัดทั้งหมดของโอเพ่นบราโวพีโอเอสทั้งเวอร์ชั่นเชิงวัตถุและเชิงแอสเป็ก พบว่าชุดมาตรวัดความซับซ้อนทางโครงสร้าง ค่าแต่ละมาตรวัดของโอเพ่นบราโวพีโอเอสที่ปรับปรุงด้วยแอสเป็กแสดงค่ามาตรวัดลดลง ยกเว้นมาตรวัด CDA และ CAE เป็นมาตรวัดที่พิจารณาคัพพลิงของแอสเป็กโดยตรง ค่ามาตรวัด CDA และ CAE ของโอเพ่นบราโวพีโอเอสเวอร์ชั่นเชิงวัตถุจึงมีค่าเท่ากับ 0 ส่วนชุดมาตรวัดระดับคอนเซิร์นแสดงผลค่าแต่ละมาตรวัดของโอเพ่นบราโวพีโอเอสที่จัดการด้วยแอสเป็กให้ผลค่ามาตรวัดที่ดีขึ้นเช่นกัน นอกจากนี้ยังได้กำหนดภารกิจสำหรับการบำรุงรักษา (Maintenance task) เพื่อประเมินความสามารถการบำรุงรักษาของโอเพ่นบราโวพีโอเอส

ทั้งสองเวอร์ชั่น โดยประเมินจากความสามารถการเปลี่ยนแปลง (Changeability) จากการปรับปรุงตามภารกิจที่กำหนดทั้งหมด 2 ภารกิจ และพิจารณาจากการเปลี่ยนแปลงของโค้ด เช่น จำนวนของโมดูล จำนวนโอเปอเรชั่น และจำนวนบรรทัดของโค้ดที่เพิ่ม/นำออกหรือถูกปรับปรุง ผลแสดงให้เห็นว่าโอเพ่นบราโวพีโอเอสที่พัฒนาด้วยการโปรแกรมเชิงวัตถุได้รับผลกระทบจากการเปลี่ยนแปลงตามภารกิจที่กำหนดมากกว่า กล่าวคือมีจำนวนโมดูล จำนวนโอเปอเรชั่น และจำนวนบรรทัดของโค้ดที่ถูกแก้ไขมากกว่า ดังนั้นทั้งจากการประเมินชุดมาตรวัดสองชุดและจากการปรับปรุงตามภารกิจการบำรุงรักษาแสดงผลสอดคล้องตรงกันว่าโอเพ่นบราโวพีโอเอสที่ปรับปรุงด้วยการโปรแกรมเชิงแอสเป็กมีคุณภาพความสามารถการบำรุงรักษาที่ดีกว่าการพัฒนาด้วยการโปรแกรมเชิงวัตถุ

Kumar และคณะ (2007) วัดความสามารถการเปลี่ยนแปลง (Changeability) ของซอฟต์แวร์เชิงแอสเป็ก โดยความสามารถการเปลี่ยนแปลงเป็นคุณลักษณะย่อยของความสามารถการบำรุงรักษา การทดลองประเมินความสามารถการเปลี่ยนแปลงจากผลกระทบของการเปลี่ยนแปลง (change impact) และมาตรวัดซอฟต์แวร์ของ Ceccato และ Tonella (2004) หน่วยตัวอย่างที่นำมาใช้ในการทดลองเป็นโปรเจคเชิงแอสเป็กที่มีจำนวนโมดูลทั้งหมด 149 โมดูล ปรับปรุงด้วยภาษาแอสเป็กเจ ผลการทดลองพบว่าค่าเฉลี่ยผลกระทบของการเปลี่ยนแปลงมีค่าต่ำกว่า 1 และมีค่าน้อยกว่าเมื่อเทียบกับค่าเฉลี่ยผลกระทบของการเปลี่ยนแปลงของซอฟต์แวร์เชิงวัตถุ ซอฟต์แวร์เชิงแอสเป็กจึงง่ายต่อการเปลี่ยนแปลงแก้ไขมากกว่าซอฟต์แวร์เชิงวัตถุ นอกจากนี้ยังกล่าวว่าชุดมาตรวัดเชิงแอสเป็กสามารถนำมาใช้เป็นตัวชี้วัดของความสามารถการเปลี่ยนแปลงและความสามารถการบำรุงรักษาได้ด้วย

Sirbi และ Kulkarni (2013) ศึกษาการใช้ดีไซน์แพตเทิร์น (Design pattern) ได้แก่ อ็อบเซิร์ฟเวอร์แพตเทิร์น (Observer pattern) โดยเปรียบเทียบระหว่างการพัฒนาด้วยการโปรแกรมเชิงวัตถุและการโปรแกรมเชิงแอสเป็กเพื่อประเมินคุณภาพการออกแบบ (Design quality) โดยพิจารณามาตรวัดบางส่วนจากชุดมาตรวัดของ Ceccato และ Tonella (2004) ที่ปรับปรุงจากมาตรวัดซีเค ได้แก่ มาตรวัด WOM CBM RFM และ LCOM ผลจากการทดลองพบว่าการพัฒนาอ็อบเซิร์ฟเวอร์แพตเทิร์นด้วยแอสเป็กส่งผลให้คุณภาพด้านความสามารถการบำรุงรักษา (Maintainability) ความสามารถการนำกลับมาใช้ใหม่ (Reusability) ความสามารถในการทำความเข้าใจ (Understandability) และความสามารถการทดสอบ (Testability) ปรับปรุงดีขึ้น เนื่องจากค่ามาตรวัด WOM RFM และ LCOM ให้ผลค่ามาตรวัดที่ดีกว่าเมื่อเทียบกับการพัฒนาด้วยการโปรแกรมเชิงวัตถุ

Al-Jamimi และคณะ (2011) ศึกษาผลกระทบของการใช้แอสเป็กรีแฟคทอริงต่อคุณภาพซอฟต์แวร์ด้านความสามารถการบำรุงรักษา โดยประเมินคุณภาพจากมาตรวัดซอฟต์แวร์ที่ปรับปรุงจากมาตรวัดซีเค ได้แก่ มาตรวัด DIT, CBC, LCOO, WOC, NOA และ LOC แอสเป็กรีแฟคทอริงที่นำมาศึกษามีหลายเทคนิคเพื่อนำผลคุณภาพของแต่ละเทคนิคมาประเมินและจำแนกกลุ่มแอสเป็กรีแฟคทอริงที่ให้ผลดีต่อคุณภาพซอฟต์แวร์ โดยแอสเป็กรีแฟคทอริงที่พิจารณาจะปรับปรุงแก้ไขโค้ดแอสเป็กเพียงอย่างเดียวไม่เกี่ยวข้องกับคลาส สามารถแบ่งวิธีแอสเป็กรีแฟคทอริงออกเป็นสองกลุ่มใหญ่ๆ ได้แก่ กลุ่มแอสเป็กรีแฟคทอริงที่ใช้กับภายในแอสเป็ก และกลุ่มแอสเป็กรีแฟคทอริงที่ใช้ระหว่างแอสเป็ก หน่วยตัวอย่างเป็นซอฟต์แวร์เชิงแอสเป็กทั้งหมด 6 หน่วยตัวอย่าง ได้แก่ พรีเวย์

เลอร์ (Prevayler) เอเจฮอตดรอว์ (AJHotDraw) แอสเป็กเททริส (AspectTetris) เอเจอีเอฟดับเบิ้ลยู (AJEFW) เทเลคอม (Telecom) และสเปซวอร์ (SpaceWar) ผลจากค่ามาตรวัดทำให้สามารถจำแนกกลุ่มแอสเป็กรีแฟคทอริงที่ให้ผลดีต่อความสามารถการบำรุงรักษา พบว่ากลุ่มแอสเป็กรีแฟคทอริงที่ใช้ระหว่างแอสเป็กทั้งหมดสามารถช่วยปรับปรุงคุณภาพความสามารถการบำรุงรักษาให้ดีขึ้น แต่กลุ่มแอสเป็กรีแฟคทอริงที่ใช้กับภายในแอสเป็กส่วนใหญ่ส่งผลให้การบำรุงรักษาซอฟต์แวร์ทำได้ยากขึ้นหรือไม่ส่งผลต่อคุณภาพซอฟต์แวร์ และเนื่องจากงานวิจัยนี้พิจารณากลุ่มแอสเป็กรีแฟคทอริงที่ปรับปรุงแค่เพียงโค้ดของแอสเป็ก จึงแตกต่างกับงานวิจัยนี้ที่เลือกแอสเป็กรีแฟคทอริงที่ปรับปรุงโค้ดของคลาส

นอกจากการเปรียบเทียบคุณภาพซอฟต์แวร์ที่สำคัญต่อการพัฒนาซอฟต์แวร์แล้ว Hanenberg, Kleinschmager และ Walter (2009) ได้นำเสนอข้อพิสูจน์ของการพัฒนาซอฟต์แวร์ด้วยการโปรแกรมเชิงแอสเป็ก โดยเปรียบเทียบเวลาที่ใช้พัฒนาระหว่างการพัฒนาแบบการโปรแกรมเชิงวัตถุและการโปรแกรมเชิงแอสเป็ก เพื่อช่วยในการตัดสินใจสำหรับการนำการโปรแกรมเชิงแอสเป็กมาใช้ในการพัฒนาซอฟต์แวร์ เป้าหมายงานวิจัยจึงสนใจว่าการโปรแกรมเชิงแอสเป็กน่าจะให้ผลเวลาที่ใช้ในการพัฒนาแง่บวก โดยระบุเพิ่มเติมว่าหากนำการโปรแกรมเชิงแอสเป็กมาช่วยจัดการโค้ดโคลนที่มีจำนวนซ้ำมาก การใช้แอสเป็กน่าจะช่วยประหยัดเวลาในการพัฒนาได้มากกว่าการโปรแกรมเชิงวัตถุ แต่หากนำการโปรแกรมเชิงแอสเป็กมาช่วยจัดการโค้ดโคลนที่มีจำนวนซ้ำน้อย การโปรแกรมเชิงแอสเป็กน่าจะทำให้ความเร็วในการพัฒนาลดลง

การทดลองใช้หน่วยทดลองทั้งหมด 20 คนสำหรับเข้าร่วมทดลองพัฒนาในรูปแบบการโปรแกรมเชิงแอสเป็กและการโปรแกรมเชิงวัตถุ ก่อนการทดลองจะสอนหน่วยทดลองเกี่ยวกับภาษาแอสเป็กเจเป็นเวลา 1.5 ชั่วโมง รายละเอียดของการทดลองมีดังนี้

- การทดลองให้หน่วยทดลองพัฒนาครอสคัทติ้งคอนเซิร์นในซอร์สโค้ดของซอฟต์แวร์ตั้งต้นที่กำหนดไว้ให้ เป็นซอฟต์แวร์เกมส์ขนาดเล็กประกอบด้วยจำนวนแพคเกจ 3 แพคเกจ จำนวนคลาส 9 คลาส จำนวนเมท็อด 110 เมท็อด จำนวนคอนสตรัคเตอร์ 8 คอนสตรัคเตอร์ และจำนวนตัวแปรอินสแตนซ์ 37 ตัวแปร ซอฟต์แวร์พัฒนาด้วยภาษาจาวา (Java)
- การทดลองกำหนดให้หน่วยทดลองพัฒนาทั้งหมด 9 ครอสคัทติ้งคอนเซิร์น แต่ละครอสคัทติ้งคอนเซิร์นมีจำนวนโค้ดโคลนระบุไว้ โดยการพัฒนาทั้ง 9 ครอสคัทติ้งคอนเซิร์นด้วยการโปรแกรมเชิงวัตถุใช้ภาษาจาวา (Java) และการพัฒนาด้วยการโปรแกรมเชิงแอสเป็กใช้ภาษาแอสเป็กเจ (AspectJ)
- การทดลองแบ่งหน่วยทดลองออกเป็นสองกลุ่ม กลุ่มละ 10 คน โดยกลุ่มแรกให้พัฒนาทั้ง 9 ครอสคัทติ้งคอนเซิร์นด้วยการโปรแกรมเชิงวัตถุก่อนเสร็จแล้วจึงพัฒนาด้วยการโปรแกรมเชิงแอสเป็ก กลุ่มที่สองให้พัฒนาทั้ง 9 ครอสคัทติ้งคอนเซิร์นด้วยการโปรแกรมเชิงแอสเป็กก่อนเสร็จแล้วจึงพัฒนาด้วยการโปรแกรมเชิงวัตถุ และเก็บข้อมูลเวลาที่ใช้พัฒนาจากหน่วยทดลองทั้ง 20 คน

ผลการทดลองพบว่าการพัฒนาครอสคัทติ้งคอนเซิร์นเกี่ยวกับการบันทึกล็อก (Logging) และการตรวจสอบตัวชี้ว่าง (Nullpointer check) ที่มีจำนวนโค้ดโคลนเท่ากับ 110 และ 36 โค้ดโคลนตามลำดับ การพัฒนาด้วยการโปรแกรมเชิงแอสเป็กให้ผลเวลาที่ใช้พัฒนาดีกว่าการโปรแกรมเชิงวัตถุ แต่การพัฒนาครอสคัทติ้งคอนเซิร์นเกี่ยวกับการซินโครไนซ์ (Synchronization) ที่มีจำนวนโค้ดโคลน

เท่ากับ 52 โค้ดโคลนไม่เห็นความแตกต่างของเวลาที่ใช้พัฒนาระหว่างการพัฒนาด้วยการโปรแกรมเชิงวัตถุและการโปรแกรมเชิงแอสเป็ก ส่วนการพัฒนาอีก 6 ครอสคัทติ้งคอนเซิร์นที่เหลือที่มีจำนวนโค้ดโคลนต่ำกว่า 36 โค้ดโคลนพบว่าเวลาที่ใช้ในการพัฒนาของการโปรแกรมเชิงวัตถุดีกว่าการโปรแกรมเชิงแอสเป็ก ดังนั้นจากผลดังกล่าวการจัดการครอสคัทติ้งคอนเซิร์นที่มีจำนวนโค้ดโคลนมากกว่า 36 โค้ดโคลน ด้วยการโปรแกรมเชิงแอสเป็กสามารถพัฒนาได้เร็วกว่าเมื่อเทียบกับการโปรแกรมเชิงวัตถุ

จากงานวิจัยข้างต้นพิจารณาแค่เพียงเวลาที่ใช้พัฒนาแต่เนื่องจากการตัดสินใจนำการโปรแกรมเชิงแอสเป็กมาใช้ควรพิจารณาคุณภาพซอฟต์แวร์ร่วมด้วย ผู้วิจัยจึงสนใจศึกษาคุณภาพซอฟต์แวร์ของการจัดการโค้ดที่มีจำนวนซ้ำของโค้ดโคลนแตกต่างกัน นอกจากนี้เพื่อควบคุมผลคุณภาพซอฟต์แวร์ที่ได้ให้เกิดจากจำนวนซ้ำของโค้ดโคลนอย่างแท้จริง ผู้วิจัยจึงเลือกโค้ดโคลนที่มีลักษณะโค้ดเหมือนกันหรือใกล้เคียงกันมากที่สุด โดยงานวิจัยนี้เลือกศึกษาการจัดการเมท็อดคอลที่ซ้ำกันในหลายคลาสด้วยแอสเป็ก อีกทั้งการศึกษาคุณภาพซอฟต์แวร์ของการจัดการเมท็อดคอลที่ซ้ำกันด้วยแอสเป็กยังไม่พบการศึกษาของงานวิจัยในอดีต ผู้วิจัยจึงสนใจเปรียบเทียบคุณภาพซอฟต์แวร์ของการจัดการเมท็อดคอลที่ซ้ำกันด้วยแอสเป็ก โดยเปรียบเทียบระหว่างการจัดการจำนวนซ้ำของเมท็อดคอลที่แตกต่างกัน

บทที่ 3
ระเบียบวิธีวิจัย

3.1 บทนำ

ในบทนี้นำเสนอถึงแนวทางในการทำวิจัยเพื่อตอบวัตถุประสงค์ของการวิจัย การนำเสนอประกอบด้วย (1) ตัวแปรสำคัญที่ศึกษา (2) แนวทางการศึกษาและการทดสอบสมมติฐาน (3) ประชากรและหน่วยตัวอย่าง (4) เครื่องมือการดำเนินการ (5) แนวทางการดำเนินงานวิจัย (6) ประเด็นของความถูกต้อง (Validity) และความน่าเชื่อถือ (Reliability) ของข้อมูลที่เก็บ และ (7) กรอบการวิเคราะห์ข้อมูล

3.2 ตัวแปรสำคัญที่ศึกษา

งานวิจัยนี้ต้องการตอบวัตถุประสงค์ทั้งหมด 3 ข้อ ได้แก่ (1) เพื่อเปรียบเทียบมาตรวัดเชิงวัตถุของซอฟต์แวร์ที่ปรับปรุงด้วยการทำแอสเป็กรีแฟคทอริงจากจำนวนโค้ดโคลนที่แตกต่างกัน (2) เพื่อเปรียบเทียบมาตรวัดเชิงแอสเป็กของซอฟต์แวร์ที่ปรับปรุงด้วยการทำแอสเป็กรีแฟคทอริงจากจำนวนโค้ดโคลนที่แตกต่างกัน และ (3) เพื่อวิเคราะห์คุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริงด้วยจำนวนโค้ดโคลนที่แตกต่างกัน ตัวแปรสำคัญที่ศึกษามีดังนี้

1. ตัวแปรต้นหรือตัวแปรอิสระ (Independent Variables) มีหนึ่งตัวแปรดังนี้

1.1. จำนวนโค้ดโคลน (Amount of Code Clone) คือ จำนวนซ้ำของเมท็อดคอลที่ซ้ำกันในหลายคลาส เกิดจากการอิมพลีเม้นต์ครอสคัทติ้งคอนเซิร์น

เนื่องจากการอิมพลีเม้นต์ครอสคัทติ้งคอนเซิร์นส่งผลให้เกิดเมท็อดคอลซ้ำกันในคลาสต่างๆ ดังนั้นโค้ดโคลนของงานวิจัยนี้จึงกล่าวถึงเมท็อดคอลที่ซ้ำกันและกระจายอยู่ภายในหลายคลาส ตัวอย่างดังรูปที่ 3.1 เมท็อดคอลที่ซ้ำกันเกิดขึ้นเนื่องจากการเรียกใช้งานเมท็อด logp() ที่เกี่ยวข้องกับครอสคัทติ้งคอนเซิร์นการบันทึกล็อก (Logging) เป็นเมท็อดสำหรับแสดงลำดับและรายละเอียดของเมท็อดที่ดำเนินการ จากตัวอย่างจำนวนโค้ดโคลนมีค่าเท่ากับ 2 พิจารณาตามจำนวนซ้ำของเมท็อดคอลที่เรียกใช้งานเมท็อด logp()

Marin, Deursen และ Moonen (2007) นำเสนอเทคนิคสำหรับค้นหาครอสคัทติ้งคอนเซิร์นที่พิจารณาจากจำนวนการเรียกใช้งานของเมท็อดหรือจำนวนซ้ำของเมท็อดคอล โดยแนะนำว่าจำนวนซ้ำของเมท็อดคอลที่มากกว่า 10 ตำแหน่งเหมาะสมเป็นครอสคัทติ้งคอนเซิร์นที่ควรจัดการด้วยแอสเป็ก นอกจากนี้ Hanenberg, Kleinschmager และ Walter (2009) นำเสนอว่าจำนวนซ้ำของโค้ดมากกว่า 36 ตำแหน่ง การพัฒนาครอสคัทติ้งคอนเซิร์นด้วยแอสเป็กสามารถช่วยลดเวลาที่ใช้พัฒนาได้ดีกว่าเมื่อเทียบกับการพัฒนาด้วยการโปรแกรมเชิงวัตถุ ดังนั้นเพื่อศึกษาผลของจำนวนโค้ดโคลนต่อการทำแอสเป็กรีแฟคทอริง ผู้วิจัยจึงกำหนดให้จำนวนโค้ดโคลนที่ศึกษาในงานวิจัยนี้เป็นจำนวนซ้ำของเมท็อดคอลที่มีค่าหลากหลายเพื่อศึกษาความแตกต่างระหว่างการจัดการเมท็อดคอลในจำนวนซ้ำของเมท็อดคอลที่น้อยหรือมากแตกต่างกัน

```
import java.util.logging.*;
public class Item {
    private String _id;
    private float _price;
    static Logger _logger = Logger.getLogger("trace");
    public Item(String id, float price) {
        _id = id;
        _price = price;
    }
    public String getID() {
        _logger.logp(Level.INFO, "Item", "getID", "Entering");
        return _id;
    }
    public float getPrice() {
        _logger.logp(Level.INFO, "Item", "getPrice", "Entering");
        return _price;
    }
}
```

รูปที่ 3.1 แสดงตัวอย่างการซ้ำของเมท็อดคอล ของ Laddad (2003)

2. ตัวแปรตาม (Dependent Variable) ทั้งหมดมีดังนี้

2.1. คุณภาพซอฟต์แวร์เชิงวัตถุ คือ คุณลักษณะทั้งหมดของซอฟต์แวร์ที่สามารถตอบสนองต่อความพึงพอใจและความต้องการของผู้ใช้ โดยสามารถประเมินจากมาตรวัดเชิงวัตถุของ Chidamber และ Kemerer (1993) มาตรวัดเชิงวัตถุมีทั้งหมด 6 มาตรวัด ได้แก่

(1) เวทเทดเมท็อดเปอคลาส (Weighted Method per Class - WMC)

(2) เดพออฟอินเฮอริแทนทรี (Depth of Inheritance Tree - DIT)

(3) นัมเบอร์ออฟชิลเดรน (Number of Children - NOC)

(4) คัพพลิงบีทวีนอ็อบเจกต์คลาส (Coupling Between Object classes - CBO)

(5) เรสปอนซ์ฟอร์อะคลาส (Response For a Class - RFC)

(6) แลคออฟโคฮีชันอินเมท็อด (Lack of Cohesion in Methods - LCOM)

2.2. คุณภาพซอฟต์แวร์เชิงแอสเป็ก คือ คุณลักษณะทั้งหมดของซอฟต์แวร์ที่สามารถตอบสนองต่อความพึงพอใจและความต้องการของผู้ใช้ โดยสามารถประเมินจากมาตรวัดเชิงแอสเป็กของ Ceccato และ Tonella (2004) มาตรวัดเชิงแอสเป็กมีทั้งหมด 10 มาตรวัด ได้แก่

(1) เวทเทดโอเปอร์เรชั่นอินมอดูล (Weighted Operations in Module - WOM)

(2) เดพออฟอินเฮอริแทนทรี (Depth of Inheritance Tree - DIT)

(3) นัมเบอร์ออฟชิลเดรน (Number Of Children - NOC)

(4) คัพพลิงออนฟิลด์แอคเซส (Coupling on Field Access - CFA)

(5) คัพพลิงออนเมท็อดคอล (Coupling on Method Call - CMC)

(6) คัพพลิงบีทวีนมอดูล (Coupling between Modules - CBM)

(7) ครอสคัทติ้งดีกรีออฟแอนแอสเป็ก (Crosscutting Degree of an Aspect - CDA)

(8) คัพพลิงออนแอดไวซ์เอ็กซิคิวชั่น (Coupling on Advice Execution - CAE)

(9) เรสปอนซ์ฟอร์อะมอดูล (Response For a Module - RFM)

(10) แลคออฟโคฮีชันอินโอเปอร์เรชั่น (Lack of Cohesion in Operations - LCO)

3. ตัวแปรควบคุม (Control Variables) เพื่อให้ผลการทดลองเกิดจากการปรับเปลี่ยนค่าของตัวแปรต้นอย่างแท้จริง ผู้วิจัยจึงพยายามควบคุมปัจจัยอื่นที่อาจส่งผลกระทบต่อการทดลอง โดยควบคุมให้คงที่หรือเหมือนกันมากที่สุด ดังนี้

3.1. ลักษณะของครอสคัทติ้งคอนเซิร์น เพื่อควบคุมผลค่ามาตรวัดทั้งหมดที่ได้ให้เกิดความแตกต่างจากจำนวนซ้ำของเมท็อดคอลอย่างแท้จริง การเลือกจัดการครอสคัทติ้งคอนเซิร์นในแต่ละจำนวนซ้ำควรมีลักษณะเหมือนกันหรือใกล้เคียงกันมากที่สุด เพื่อควบคุมปัจจัยความแตกต่างของครอสคัทติ้งคอนเซิร์นที่อาจส่งผลกระทบต่อค่ามาตรวัด เนื่องจากการทำแอสเป็กรีแฟคทอริงเป็นการจัดการโค้ดที่พัฒนาด้วยการโปรแกรมเชิงวัตถุ เบื้องต้นผู้วิจัยจึงได้ศึกษาองค์ประกอบของโปรแกรมที่เป็นสาเหตุทำให้ค่ามาตรวัดเชิงวัตถุเกิดการเปลี่ยนแปลงเพื่อกำหนดลักษณะของครอสคัทติ้งคอนเซิร์น ดังตารางที่ 3.1

ตารางที่ 3.1 มาตรวัดเชิงวัตถุของ Chidamber และ Kemerer (1993) และองค์ประกอบของโปรแกรมที่ใช้สำหรับพิจารณาค่ามาตรวัด

มาตรวัดเชิงวัตถุของ Chidamber และ Kemerer (1993)	**องค์ประกอบของโปรแกรมที่ใช้สำหรับพิจารณาค่ามาตรวัด**
1. เวทเทดเมท็อดเปอคลาส (Weighted Method per Class - WMC)	จำนวนเมท็อดของคลาส
2. เดพออฟอินเฮอริแทนทรี (Depth of Inheritance Tree - DIT)	จำนวนลำดับชั้นการรับทอด
3. นัมเบอร์ออฟชิลเดรน (Number of Children - NOC)	จำนวนคลาสลูก
4. คัพพลิงบีทวีนอ็อบเจกต์คลาส (Coupling Between Object classes - CBO)	จำนวนความสัมพันธ์ระหว่างคลาส (ดูจากตัวแปรอินสแตนซ์ และเมท็อดคอลที่มีในคลาส)
5. เรสปอนซ์ฟอร์อะคลาส (Response For a Class - RFC)	จำนวนเมท็อดในคลาสรวมกับจำนวนเมท็อดที่ดำเนินการเมื่อถูกเรียกใช้งานจากคลาสดังกล่าว
6. แลคออฟโคฮีชันอินเมท็อด (Lack of Cohesion in Methods - LCOM)	จำนวนของการใช้ตัวแปรอินสแตนซ์และแอตทริบิวต์ร่วมกันระหว่างเมท็อดในคลาส

เนื่องจากการเรียกใช้งานเมท็อดเดียวกัน โดยเรียกใช้งานมาจากหลายคลาสทำให้เกิดเมท็อดคอลซ้ำกันหรือกระจายในหลายคลาส โดยสามารถพิจารณาเป็นคุณลักษณะของครอสคัทติ้งคอนเซิร์นได้เช่นกัน (Neto, 2004) และเนื่องด้วยผู้วิจัยต้องการควบคุมให้การเลือกจำนวนซ้ำของแต่ละครอสคัทติ้งคอนเซิร์นมีลักษณะของครอสคัทติ้งคอนเซิร์นคล้ายกันมากที่สุด ผู้วิจัยจึงเลือกครอสคัทติ้งคอนเซิร์นที่มีลักษณะเมท็อดคอล เนื่องจากการจัดการแต่ละเมท็อดคอลด้วยแอสเป็กเป็นการจัดการคำสั่งเพียงบรรทัดเดียว และหากจัดการเมท็อดคอลที่แตกต่างกันด้วยแอสเป็กจะกระทบเพียงค่ามาตรวัดเรสปอนซ์ฟอร์อะคลาส (Response For a Class - RFC) ในจำนวนที่เท่ากัน

3.2. กรณีศึกษาหรือซอฟต์แวร์ที่นำมาใช้สำหรับการทำแอสเป็กรีแฟคทอริงเพื่อจัดการจำนวนซ้ำของโค้ดโคลนที่แตกต่างกัน ผู้วิจัยเลือกจัดการเมท็อดคอลที่ซ้ำกันภายในซอฟต์แวร์เดียว โดยการจัดการแต่ละจำนวนซ้ำของเมท็อดคอลใช้ซอฟต์แวร์ตั้งต้นเดียวกันที่มีโค้ดและฟังก์ชันการทำงานเหมือนกัน เพื่อควบคุมให้ผลมาตรวัดที่แตกต่างกันมีผลมาจากการเปลี่ยนแปลงของตัวแปรต้นและเกิดขึ้นภายใต้สภาพแวดล้อมของซอฟต์แวร์ที่เหมือนกัน

3.3. เครื่องมือสำหรับวัดค่ามาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็ก โดยสำหรับมาตรวัดเชิงวัตถุของ Chidamber และ Kemerer (1993) ทั้ง 6 มาตรวัด ผู้วิจัยเลือกใช้เครื่องมือชื่อซีเคเจเอ็ม (CKJM) เพื่อควบคุมให้มาตรฐานการวัดของแต่ละมาตรวัดเหมือนกันจึงใช้เครื่องมือวัดเพียงเครื่องมือเดียว เช่นกันกับมาตรวัดเชิงแอสเป็กของ Ceccato และ Tonella (2004) ทั้ง 10 มาตรวัด ผู้วิจัยเลือกใช้เครื่องมือชื่อเอโอพีเมตริก (AopMetrics) เป็นเครื่องมือที่พัฒนาขึ้นโดย Ceccato และ Tonella ผู้นำเสนอชุดมาตรวัดเชิงแอสเป็กทั้ง 10 มาตรวัด เครื่องมือเอโอพีเมตริกจึงมีมาตรฐานและความน่าเชื่อถือสำหรับเก็บรวบรวมค่ามาตรวัดเชิงแอสเป็ก

3.3 แนวทางการศึกษาและการทดสอบสมมติฐาน

งานวิจัยนี้เป็นการวิจัยเชิงทดลอง (Experiment Research) ที่ต้องการวิเคราะห์ผลที่ได้เพื่อตอบวัตถุประสงค์ของงานวิจัย คือ เพื่อเปรียบเทียบคุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริงด้วยจำนวนโค้ดโคลนที่แตกต่างกัน โดยพิจารณาจากมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็ก

จากวัตถุประสงค์ของงานวิจัยดังกล่าวผู้วิจัยได้ตั้งสมมติฐานของงานวิจัยดังนี้

1. ผลของจำนวนซ้ำเมท็อดคอลต่อค่าเฉลี่ยของมาตรวัดเชิงวัตถุเวทเทดเมท็อดเปอคลาส (Weighted Method per Class - WMC)

H_0 : จำนวนซ้ำของเมท็อดคอลไม่มีผลต่อค่าเฉลี่ยของมาตรวัดเวทเทดเมท็อดเปอคลาส

H_1 : จำนวนซ้ำของเมท็อดคอลมีผลต่อค่าเฉลี่ยของมาตรวัดเวทเทดเมท็อดเปอคลาส

2. ผลของจำนวนซ้ำเมท็อดคอลต่อค่าเฉลี่ยของมาตรวัดเชิงวัตถุเดพออฟอินเฮอริแทนทรี (Depth of Inheritance Tree - DIT)

H_0 : จำนวนซ้ำของเมท็อดคอลไม่มีผลต่อค่าเฉลี่ยของมาตรวัดเดพออฟอินเฮอริแทนทรี

H_1 : จำนวนซ้ำของเมท็อดคอลมีผลต่อค่าเฉลี่ยของมาตรวัดเดพออฟอินเฮอริแทนทรี

3. ผลของจำนวนซ้ำเมท็อดคอลต่อค่าเฉลี่ยของมาตรวัดเชิงวัตถุนัมเบอร์ออฟชิลเดรน (Number of Children - NOC)

H_0 : จำนวนซ้ำของเมท็อดคอลไม่มีผลต่อค่าเฉลี่ยของมาตรวัดนัมเบอร์ออฟชิลเดรน

H_1 : จำนวนซ้ำของเมท็อดคอลมีผลต่อค่าเฉลี่ยของมาตรวัดนัมเบอร์ออฟชิลเดรน

4. ผลของจำนวนซ้ำเมท็อดคอลต่อค่าเฉลี่ยของมาตรวัดเชิงวัตถุคัพพลิงบีทวีนอ็อบเจกต์คลาส (Coupling Between Object classes - CBO)

H_0 : จำนวนซ้ำของเมท็อดคอลไม่มีผลต่อค่าเฉลี่ยของมาตรวัดคัพพลิงบีทวีนอ็อบเจกต์คลาส

H_1 : จำนวนซ้ำของเมท็อดคอลมีผลต่อค่าเฉลี่ยของมาตรวัดคัพพลิงบีทวีนอ็อบเจกต์คลาส

5. ผลของจำนวนซ้ำเมท็อดคอลต่อค่าเฉลี่ยของมาตรวัดเชิงวัตถุเรสปอนซ์ฟอร์อะคลาส (Response For a Class - RFC)

H_0 : จำนวนซ้ำของเมท็อดคอลไม่มีผลต่อค่าเฉลี่ยของมาตรวัดเรสปอนซ์ฟอร์อะคลาส

H_1 : จำนวนซ้ำของเมท็อดคอลมีผลต่อค่าเฉลี่ยของมาตรวัดเรสปอนซ์ฟอร์อะคลาส

6. ผลของจำนวนซ้ำเมท็อดคอลต่อค่าเฉลี่ยของมาตรวัดเชิงวัตถุแลคออฟโคฮีชันอินเมท็อด (Lack of Cohesion in Methods - LCOM)

H_0 : จำนวนซ้ำของเมท็อดคอลไม่มีผลต่อค่าเฉลี่ยของมาตรวัดแลคออฟโคฮีชันอินเมท็อด

H_1 : จำนวนซ้ำของเมท็อดคอลมีผลต่อค่าเฉลี่ยของมาตรวัดแลคออฟโคฮีชันอินเมท็อด

7. ผลของจำนวนซ้ำเมท็อดคอลต่อค่าเฉลี่ยของมาตรวัดเชิงแอสเป็กเวทเทดโอเปอร์เรชั่นอินมอดูล (Weighted Operations in Module - WOM)

H_0 : จำนวนซ้ำของเมท็อดคอลไม่มีผลต่อค่าเฉลี่ยของมาตรวัดเวทเทดโอเปอร์เรชั่นอินมอดูล

H_1 : จำนวนซ้ำของเมท็อดคอลมีผลต่อค่าเฉลี่ยของมาตรวัดเวทเทดโอเปอร์เรชั่นอินมอดูล

8. ผลของจำนวนซ้ำเมท็อดคอลต่อค่าเฉลี่ยของมาตรวัดเชิงแอสเป็กเดพออฟอินเฮอริแทนทรี (Depth of Inheritance Tree - DIT)

H_0 : จำนวนซ้ำของเมท็อดคอลไม่มีผลต่อค่าเฉลี่ยของมาตรวัดเดพออฟอินเฮอริแทนทรี

H_1 : จำนวนซ้ำของเมท็อดคอลมีผลต่อค่าเฉลี่ยของมาตรวัดเดพออฟอินเฮอริแทนทรี

9. ผลของจำนวนซ้ำเมท็อดคอลต่อค่าเฉลี่ยของมาตรวัดเชิงแอสเป็กนัมเบอร์ออฟชิลเดรน (Number Of Children - NOC)

H_0 : จำนวนซ้ำของเมท็อดคอลไม่มีผลต่อค่าเฉลี่ยของมาตรวัดนัมเบอร์ออฟชิลเดรน

H_1 : จำนวนซ้ำของเมท็อดคอลมีผลต่อค่าเฉลี่ยของมาตรวัดนัมเบอร์ออฟชิลเดรน

10. ผลของจำนวนซ้ำเมท็อดคอลต่อค่าเฉลี่ยของมาตรวัดเชิงแอสเป็กครอสคัทติ้งดีกรีออฟแอนแอสเป็ก (Crosscutting Degree of an Aspect - CDA)

H_0 : จำนวนซ้ำของเมท็อดคอลไม่มีผลต่อค่าเฉลี่ยของมาตรวัดครอสคัทติ้งดีกรีออฟแอนแอสเป็ก

H_1 : จำนวนซ้ำของเมท็อดคอลมีผลต่อค่าเฉลี่ยของมาตรวัดครอสคัทติ้งดีกรีออฟแอนแอสเป็ก

11. ผลของจำนวนซ้ำเมท็อดคอลต่อค่าเฉลี่ยของมาตรวัดเชิงแอสเป็กคัพพลิงออนแอดไวซ์เอ็กซิคิวชั่น (Coupling on Advice Execution - CAE)

H_0 : จำนวนซ้ำของเมท็อดคอลไม่มีผลต่อค่าเฉลี่ยของมาตรวัดคัพพลิงออนแอดไวซ์เอ็กซิคิวชั่น

H_1 : จำนวนซ้ำของเมท็อดคอลมีผลต่อค่าเฉลี่ยของมาตรวัดคัพพลิงออนแอดไวซ์เอ็กซิคิวชั่น

12. ผลของจำนวนซ้ำเมท็อดคอลต่อค่าเฉลี่ยของมาตรวัดเชิงแอสเป็กคัพพลิงออนเมท็อดคอล (Coupling on Method Call - CMC)

H_0 : จำนวนซ้ำของเมท็อดคอลไม่มีผลต่อค่าเฉลี่ยของมาตรวัดคัพพลิงออนเมท็อดคอล

H_1 : จำนวนซ้ำของเมท็อดคอลมีผลต่อค่าเฉลี่ยของมาตรวัดคัพพลิงออนเมท็อดคอล

13. ผลของจำนวนซ้ำเมท็อดคอลต่อค่าเฉลี่ยของมาตรวัดเชิงแอสเป็กคัพพลิงออนฟิลด์แอคเซส (Coupling on Field Access - CFA)

H_0 : จำนวนซ้ำของเมท็อดคอลไม่มีผลต่อค่าเฉลี่ยของมาตรวัดคัพพลิงออนฟิลด์แอคเซส

H_1 : จำนวนซ้ำของเมท็อดคอลมีผลต่อค่าเฉลี่ยของมาตรวัดคัพพลิงออนฟิลด์แอคเซส

14. ผลของจำนวนซ้ำเมท็อดคอลต่อค่าเฉลี่ยของมาตรวัดเชิงแอสเป็กคัพพลิงบีทวีนมอดูล (Coupling between Modules - CBM)

H_0 : จำนวนซ้ำของเมท็อดคอลไม่มีผลต่อค่าเฉลี่ยของมาตรวัดคัพพลิงบีทวีนมอดูล

H_1 : จำนวนซ้ำของเมท็อดคอลมีผลต่อค่าเฉลี่ยของมาตรวัดคัพพลิงบีทวีนมอดูล

15. ผลของจำนวนซ้ำเมท็อดคอลต่อค่าเฉลี่ยของมาตรวัดเชิงแอสเป็กเรสปอนซ์ฟอร์อะมอดูล (Response For a Module - RFM)

H_0 : จำนวนซ้ำของเมท็อดคอลไม่มีผลต่อค่าเฉลี่ยของมาตรวัดเรสปอนซ์ฟอร์อะมอดูล

H_1 : จำนวนซ้ำของเมท็อดคอลมีผลต่อค่าเฉลี่ยของมาตรวัดเรสปอนซ์ฟอร์อะมอดูล

16. ผลของจำนวนซ้ำเมท็อดคอลต่อค่าเฉลี่ยของมาตรวัดเชิงแอสเป็กแลคออฟโคฮีชันอินโอเปอร์เรชั่น (Lack of Cohesion in Operations - LCO)

H_0 : จำนวนซ้ำของเมท็อดคอลไม่มีผลต่อค่าเฉลี่ยของมาตรวัดแลคออฟโคฮีชันอินโอเปอร์เรชั่น

H_1 : จำนวนซ้ำของเมท็อดคอลมีผลต่อค่าเฉลี่ยของมาตรวัดแลคออฟโคฮีชันอินโอเปอร์เรชั่น

3.4 ประชากรและหน่วยตัวอย่าง

ประชากร (Population) หมายถึง ทุกหน่วยในเรื่องที่สนใจศึกษา โดยประชากรอาจเป็นบุคคล องค์กร สัตว์ หรือสิ่งของ (กัลยา วานิชย์บัญชา, 2551b) เนื่องจากงานวิจัยนี้ต้องการศึกษาผลคุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริง โดยจัดการเมท็อดคอลที่มีจำนวนซ้ำของเมท็อดคอลแตกต่างกัน ดังนั้นประชากรของงานวิจัยคือกลุ่มของเมท็อดคอลที่เรียกใช้งานเมท็อดเดียวกันทำให้เกิดโค้ดโคลนภายในซอร์สโค้ดของซอฟต์แวร์ ในทางปฏิบัติผู้วิจัยไม่สามารถนำซอฟต์แวร์ทั้งหมดมาศึกษาเพื่อค้นหาโค้ดโคลนและทำแอสเป็กรีแฟคทอริงได้ ผู้วิจัยจึงเลือกศึกษาจากกรณีศึกษา (Case Study) ที่ได้เลือกมาหนึ่งกรณีศึกษาสำหรับการทำแอสเป็กรีแฟคทอริงเพื่อควบคุมการทดลองให้อยู่ภายใต้สภาพแวดล้อม (Environment) เดียวกัน กรณีศึกษาที่เลือกคือ ซอฟต์แวร์เจฮอตดรอว์ (JHotDraw) เป็นซอฟต์แวร์ที่พัฒนาด้วยการโปรแกรมเชิงวัตถุและเป็นโครงการโอเพนซอร์ส (Open source project) เกี่ยวกับเฟรมเวิร์คกราฟิกสองมิติสำหรับการวาดรูปทรงที่สามารถจัดเป็นโครงสร้าง

นอกจากนี้เจฮอตดรอว์ยังพัฒนาเป็นตัวอย่างสำหรับการใช้ดีไซน์แพตเทิร์น (Design pattern) และอีกเหตุผลสำหรับการเลือกเจฮอตดรอว์เป็นกรณีศึกษา เนื่องจากผู้วิจัยเห็นว่าเป็นซอฟต์แวร์ขนาดใหญ่และหลายงานวิจัยนำมาใช้เป็นกรณีศึกษาสำหรับงานวิจัยที่เกี่ยวข้องกับการโปรแกรมเชิงแอสเป็ก (Ceccato et al., 2006; Marin et al., 2007; Roy et al., 2007) และด้วยข้อจำกัดของเครื่องมือที่ใช้เก็บรวบรวมค่ามาตรวัดทำให้ผู้วิจัยต้องดำเนินการทดลองบนโปรแกรมอีคลิปส์ (Eclipse) ที่ติดตั้งเจดีเค เวอร์ชั่น 1.5 (JDK 1.5) จากการทดลองรันเจฮอตดรอว์ทุกเวอร์ชั่นบนโปรแกรมอีคลิปส์ มีเพียงเจฮอตดรอว์ เวอร์ชั่น 7.1 หรือเวอร์ชั่นต่ำกว่าที่สามารถรันได้ภายใต้เจดีเค เวอร์ชั่น 1.5 ผู้วิจัยจึงเลือกใช้เจฮอตดรอว์ เวอร์ชั่น 7.1 เป็นกรณีศึกษาสำหรับการทำแอสเป็กรีแฟคทอริง รายละเอียดของซอฟต์แวร์เจฮอตดรอว์ เวอร์ชั่น 7.1 แสดงดังตารางที่ 3.2 ดังนั้นหน่วยตัวอย่างในงานวิจัยนี้คือกลุ่มของเมท็อดคอลที่เรียกใช้งานเมท็อดเดียวกันทำให้เกิดโค้ดโคลนภายในซอร์สโค้ดของเจฮอตดรอว์ เวอร์ชั่น 7.1

ตารางที่ 3.2 รายละเอียดของซอฟต์แวร์เจฮอตดรอว์ เวอร์ชั่น 7.1

ข้อมูลทั่วไป	
ชื่อซอฟต์แวร์	เจฮอตดรอว์ (JHotDraw)
เวอร์ชั่น	7.1
ผู้บำรุงรักษา	Werner Randelshofer
ประเภทลิขสิทธิ์	แอลจีพีแอล (LGPL)
ช่วงเวลาพัฒนา	ค.ศ. 2004-2008
ภาษาที่ใช้พัฒนา	จาวา (Java)
ข้อมูลโครงสร้างของซอฟต์แวร์	
จำนวนแพคเกจ	38
จำนวนคลาส	442
จำนวนเมท็อด	4,285
จำนวนบรรทัดของซอร์สโค้ด (Source line of code - SLOC)	53,323

งานวิจัยนี้เลือกเจฮอตดรอว์ เวอร์ชั่น 7.1 เป็นกรณีศึกษาเดียวเนื่องจากในงานวิจัยต้องการศึกษาข้อมูลเบื้องต้น (Exploratory Research) เพื่อประโยชน์สำหรับการวิจัยในอนาคตเท่านั้น ไม่ได้ต้องการสรุปผลให้ครอบคลุมต่อการนำไปประยุกต์ใช้กับกรณีทั่วไป

3.5 เครื่องมือการดำเนินการ

เครื่องมือที่นำมาใช้ในการทดลองเพื่อตอบวัตถุประสงค์งานวิจัยมีดังนี้

1. เครื่องมือสำหรับค้นหาครอสคัทติ้งคอนเซิร์น เนื่องจากผู้วิจัยกำหนดเลือกจัดการครอสคัทติ้งคอนเซิร์นที่มีลักษณะเมท็อดคอล ผู้วิจัยจึงเลือกใช้เทคนิคแฟน-อินอะนาไลซิส (Fan-in analysis) เป็นเทคนิคที่ Marin, Deursen และ Moonen (2007) นำเสนอสำหรับค้นหาครอสคัทติ้งคอนเซิร์น โดยพิจารณาจากจำนวนการเรียกใช้งานของเมท็อดที่ทำให้เกิดการซ้ำกันของเมท็อดคอลกระจายอยู่ภายในคลาสต่างๆ นอกจากนี้ Marin, Deursen และ Moonen (2007) ยังได้นำเสนอเครื่องมือเอฟไอเอ็นที (FINT) เป็นเครื่องมือที่นำเทคนิคแฟน-อินอะนาไลซิสมาประยุกต์ใช้จริง โดยเครื่องมือดังกล่าวสามารถใช้งานร่วมกับโปรแกรมอีคลิปส์ได้ ผู้วิจัยจึงเลือกใช้เครื่องมือเอฟไอเอ็นทีสำหรับค้นหาเมท็อดคอลที่ซ้ำกันภายในเจฮอตดรอว์ เวอร์ชั่น 7.1

2. เครื่องมือสำหรับเก็บรวบรวมมาตรวัดเชิงวัตถุ (Object-oriented Metric) เพื่อนำมาประเมินคุณภาพซอฟต์แวร์ งานวิจัยนี้ใช้ชุดมาตรวัดเชิงวัตถุของ Chidamber และ Kemerer (1993) ที่มีทั้งหมดหกมาตรวัดสำหรับประเมินคุณภาพซอฟต์แวร์ ผู้วิจัยจึงเลือกใช้เครื่องมือซีเคเจเอ็ม (CKJM) เนื่องจากเป็นเครื่องมือที่สามารถเก็บรวบรวมค่ามาตรวัดของ Chidamber และ Kemerer (1993) ได้ครบทุกมาตรวัด ค่ามาตรวัดที่เก็บรวบรวมสามารถพิจารณาจากซอร์สโค้ดที่พัฒนาด้วยภาษาจาวา อีกทั้งเครื่องมือซีเคเจเอ็มยังสามารถใช้งานร่วมกับโปรแกรมอีคลิปส์

3. เครื่องมือสำหรับเก็บรวบรวมมาตรวัดเชิงแอสเป็ก (Aspect-oriented Metric) เพื่อนำมาประเมินคุณภาพซอฟต์แวร์ งานวิจัยนี้ใช้มาตรวัดเชิงแอสเป็กของ Ceccato และ Tonella (2004) ที่มีทั้งหมดสิบมาตรวัดสำหรับประเมินคุณภาพซอฟต์แวร์ ผู้วิจัยจึงเลือกใช้เครื่องมือเอโอพีเมตริก (AopMetrics) เนื่องจากเป็นเครื่องมือที่สามารถเก็บรวบรวมค่ามาตรวัดของ Ceccato และ Tonella (2004) ได้ครบทุกมาตรวัด และพัฒนาโดย Ceccato และ Tonella ซึ่งเป็นผู้นำเสนอชุดมาตรวัด อีกทั้งยังมีหลายงานวิจัยใช้เครื่องมือเอโอพีเมตริกสำหรับเก็บรวบรวมค่ามาตรวัดเชิงแอสเป็ก ได้แก่ Srivisut และ Muenchaisri (2006), Kumar และคณะ (2007), Sirbi และ Kulkarni (2013) และ Mguni และ Ayalew (2013) นอกจากนี้เครื่องมือเอโอพีเมตริกยังสามารถเก็บรวบรวมค่ามาตรวัดจากซอร์สโค้ดที่พัฒนาด้วยภาษาจาวาและภาษาแอสเป็กเจ รวมถึงเป็นเครื่องมือที่สามารถใช้งานร่วมกับโปรแกรมอีคลิปส์ด้วย

3.6 แนวทางการดำเนินงานวิจัย

งานวิจัยนี้ต้องการศึกษาการปรับปรุงซอร์สโค้ดของซอฟต์แวร์ที่มีเมท็อดคอลซ้ำกันด้วยการทำแอสเป็กรีแฟคทอริง และศึกษาผลของจำนวนโค้ดโคลนต่อคุณภาพซอฟต์แวร์จากการพิจารณาด้วยมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็ก แนวทางการดำเนินงานวิจัยเพื่อตอบวัตถุประสงค์งานวิจัยสามารถดำเนินการดังรูปที่ 3.2

รูปที่ 3.2 แนวทางการดำเนินงานวิจัย

1. การกำหนดซอฟต์แวร์ที่ใช้เป็นกรณีศึกษา

ซอฟต์แวร์ที่นำมาใช้เป็นกรณีศึกษาในการทดลอง ผู้วิจัยได้เลือกจากโอเพนซอร์สโปรเจคที่เผยแพร่ให้สามารถดาวน์โหลดได้ โดยกรณีศึกษาที่นำมาใช้คือซอฟต์แวร์เจฮอตดรอว์ เวอร์ชั่น 7.1 เป็นโอเพนซอร์สโปรเจคเกี่ยวกับกราฟิกสองมิติ เจฮอตดรอว์เป็นตัวอย่างสำหรับการใช้ดีไซน์แพตเทิร์น และโอเพนซอร์สโปรเจคดังกล่าวพัฒนาด้วยภาษาจาวา

2. การเลือกหน่วยตัวอย่าง

การเลือกหน่วยตัวอย่างจะเริ่มต้นจากการเลือกเทคนิคสำหรับค้นหาครอสคัทติ้งคอนเซิร์นที่มีภายในซอร์สโค้ดของซอฟต์แวร์หรือกระบวนการแอสเป็กไมน์นิง (Aspect mining) เป็นกระบวนการที่ต้องดำเนินการก่อนนำครอสคัทติ้งคอนเซิร์นที่ได้มาจัดการเป็นแอสเป็กด้วยกระบวนการแอสเป็กรีแฟคทอริง โดยมีงานวิจัยในอดีตนำเสนอเทคนิคสำหรับค้นหาครอสคัทติ้งคอนเซิร์นเอาไว้มากมาย (Kellens et al., 2007) เนื่องจากหน่วยตัวอย่างสำหรับงานวิจัยนี้คือกลุ่มของเมท็อดคอลที่ซ้ำกันที่เรียกใช้งานเมท็อดเดียวกันทำให้เกิดโค้ดโคลนภายในซอร์สโค้ดของเจฮอตดรอว์ เวอร์ชั่น 7.1 ดังนั้นการเลือกหน่วยตัวอย่างเพื่อให้ได้จำนวนซ้ำของเมท็อดคอลที่หลากหลายสำหรับนำมาเปรียบเทียบความแตกต่างของคุณภาพจากการทำแอสเป็กรีแฟคทอริง ผู้วิจัยจึงเลือกใช้เทคนิคแฟน-อินอะนาไลซิส (Fan-in analysis) สำหรับค้นหาเมท็อดคอลซ้ำกันภายในซอร์สโค้ดของเจฮอตดรอว์ เวอร์ชั่น 7.1

โดยมีเครื่องมือเอฟไอเอ็นที (FINT) รองรับสำหรับการใช้เทคนิคแฟน-อินอะนาไลซิส และสามารถใช้งานเครื่องมือเอฟไอเอ็นทีบนโปรแกรมอีคลิปส์ (Eclipse)

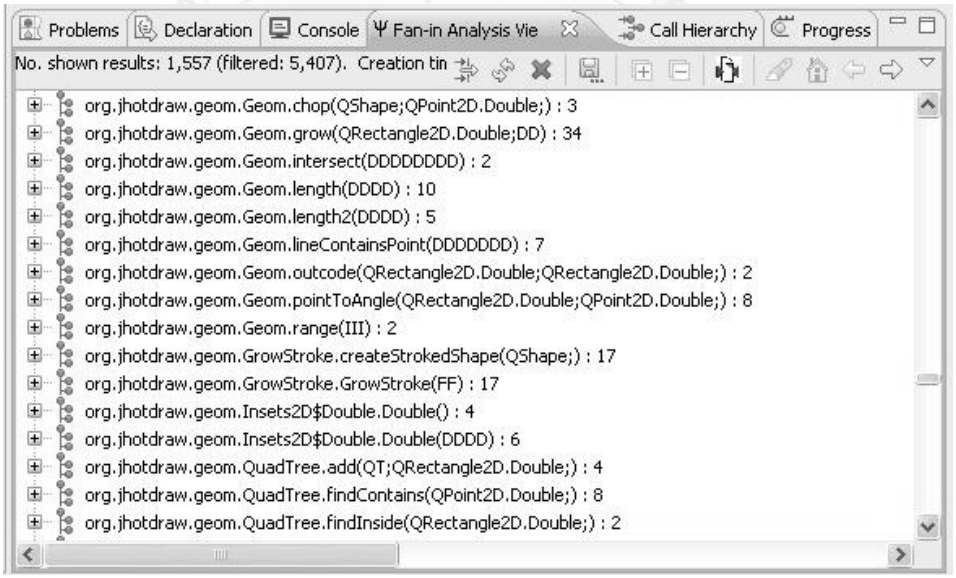
เบื้องต้นก่อนการใช้งานเครื่องมือเอฟไอเอ็นที จำเป็นต้องตั้งค่าสำหรับคัดกรองผลลัพธ์ที่ได้จากการค้นหาด้วยเทคนิคแฟน-อินอะนาไลซิส โดยกำหนดการตั้งค่าทั้งหมดสามส่วน ดังนี้

(1) ส่วนแอกเซสเซอร์ (Accessors) เป็นส่วนสำหรับเลือกพิจารณาหรือไม่พิจารณาเมท็อดคอลที่เรียกใช้งานเมท็อดเกี่ยวกับการเข้าถึงแอตทริบิวต์ของคลาส ได้แก่ เมท็อดประเภทเกตเตอร์ (Getter) และเซตเตอร์ (Setter) ซึ่ง Marin, Deursen และ Moonen (2007) ได้แนะนำให้กำหนดไม่พิจารณาเมท็อดคอลที่เรียกใช้งานเมท็อดประเภทเกตเตอร์ เซตเตอร์

(2) ส่วนเอ็กซ์เทอนอล ไลบรารี่ (External libs) เป็นส่วนสำหรับเลือกพิจารณาหรือไม่พิจารณาเมท็อดคอลที่เรียกใช้งานเมท็อดที่อยู่ภายในไลบรารี่ภายนอกที่นำเข้ามาใช้ในโปรเจค ซึ่ง Marin, Deursen และ Moonen (2007) ได้แนะนำให้กำหนดไม่พิจารณาเมท็อดคอลที่เรียกใช้งานเมท็อดจากไลบรารี่ภายนอก

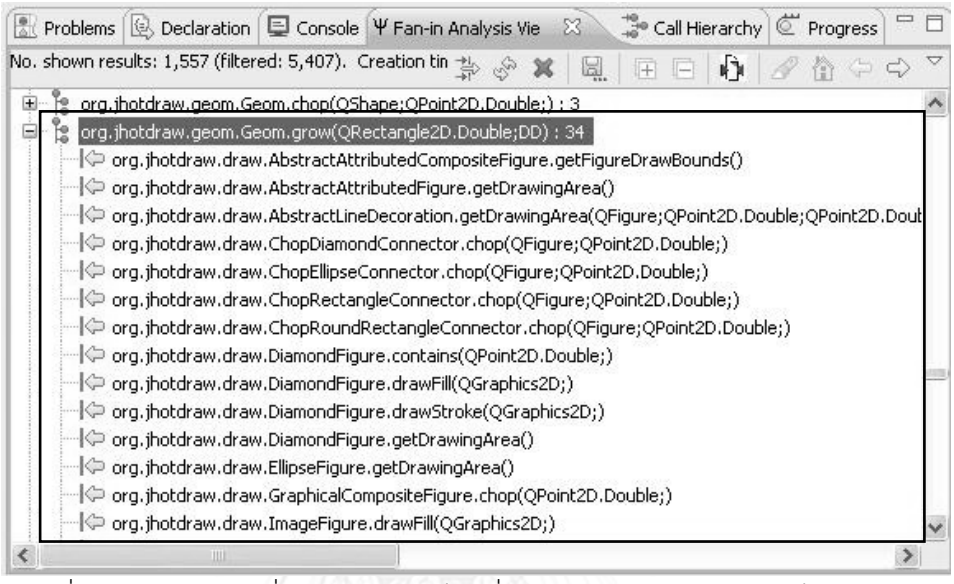
(3) ส่วนค่าเธรโชลด์ (Threshold) เป็นส่วนสำหรับกำหนดขั้นต่ำของจำนวนการเรียกใช้งานเมท็อดที่สนใจพิจารณาเป็นครอสคัทติ้งคอนเซิร์น กล่าวคือเป็นการกำหนดขั้นต่ำของจำนวนซ้ำของเมท็อดคอลที่ต้องการพิจารณา โดยผู้วิจัยเลือกตั้งค่าเธรโชลด์ตั้งแต่ 2 เป็นต้นไป เนื่องจากโค้ดโคลนคือการซ้ำกันของโค้ดตั้งแต่ 2 ตำแหน่งเป็นต้นไป

เมื่อตั้งค่าเบื้องต้นสำหรับเครื่องมือเอฟไอเอ็นทีเสร็จเรียบร้อย จึงนำเครื่องมือมาใช้ค้นหาครอสคัทติ้งคอนเซิร์นที่ต้องการจัดการภายในซอร์สโค้ดของเจฮอตดรอว์ เวอร์ชั่น 7.1 ผลจากการค้นหาจะได้ลิสผลลัพธ์เมท็อดพร้อมทั้งจำนวนการเรียกใช้งานเมท็อดดังกล่าวที่มาจากคลาสอื่น (จำนวนซ้ำของเมท็อดคอล) ดังรูปที่ 3.3



รูปที่ 3.3 ผลลัพธ์จากการใช้เครื่องมือเอฟไอเอ็นที (FINT)

จากรูปที่ 3.4 เป็นตัวอย่างผลลัพธ์จากการค้นหาครอสคัทติ้งคอนเซิร์น เครื่องมือจะแสดงผลลัพธ์การเรียกใช้งานเมท็อด grow() ของคลาส Geom โดยแสดงรายชื่อคลาสและเมท็อดที่ภายในมีการเรียกใช้งานเมท็อด grow() จากตัวอย่างเมท็อด grow() ถูกเรียกใช้งานจากคลาสอื่นๆทั้งหมด 34 ตำแหน่ง จึงมีจำนวนซ้ำของเมท็อดคอลเท่ากับ 34



รูปที่ 3.4 ตัวอย่างรายชื่อคลาสและเมท็อดที่ภายในมีการเรียกใช้งานเมท็อด grow()

จากลิสผลลัพธ์เมท็อดทั้งหมด 1,557 เมท็อดซึ่งเป็นจำนวนประชากรที่ได้จากการค้นหาด้วยเครื่องมือเอฟไอเอ็นที ในการเลือกหน่วยตัวอย่างจะต้องนำผลลัพธ์เมท็อดทั้งหมดมาวิเคราะห์หากลุ่มเมท็อดคอลซ้ำกันที่จะเลือกจัดการด้วยแอสเป็กอีกที โดยต้องนำลิสเมท็อดมาคัดกรองด้วยตัวเองดังขั้นตอนที่ 3 ของการคัดกรองผลลัพธ์ที่ได้แสดงไว้ในบทที่ 2 โดย Marin, Deursen และ Moonen (2007) กล่าวว่าต้องวิเคราะห์ผลลัพธ์เมท็อดที่ควรจะเลือกเป็นครอสคัทติ้งคอนเซิร์นจากการพิจารณาตำแหน่งการเรียกใช้งานเมท็อดที่มาจากคลาสต่างๆ เพื่อให้สามารถจัดการด้วยพอยท์คัทของภาษาแอสเป็กเจได้

ดังนั้นเนื่องด้วยข้อจำกัดของเครื่องมือที่ใช้เก็บรวบรวมค่ามาตรวัดและข้อจำกัดของภาษาแอสเป็กเจ การจัดการเมท็อดคอลบางตำแหน่งจึงไม่สามารถจัดการได้ด้วยแอสเป็ก สามารถแสดงรายละเอียดของเมท็อดคอลที่ไม่สามารถจัดการด้วยแอสเป็กได้ดังตารางที่ 3.3

ตารางที่ 3.3 รายละเอียดเมท็อดคอลที่ไม่สามารถจัดการด้วยแอสเป็ก

ลำดับ	สาเหตุ	ตำแหน่งเมท็อดคอล	ตัวอย่างโค้ด

1	เครื่องมือเอโอพีเมตริกไม่สามารถวัดพอยท์คัทที่ระบุไปยังการเรียกใช้งานเมท็อด	ตำแหน่งของเมท็อดคอลอยู่ก่อนหรือหลังเมท็อดคอลอื่นๆ	owner.willChange();
			owner.restoreTransformTo(..);
			owner.changed();

2	พอยท์คัทของภาษาแอสเป็กเจไม่รองรับการระบุไปยังตำแหน่งของเมท็อดคอล	ตำแหน่งของเมท็อดคอลอยู่ภายในคลาสแอบสแตร็ก (Abstract class)	public *abstract* class AbstractFigure
			extends AbstractBean
			implements Figure {
			public void setVisible(..) {
			changed();
			}
			}

		ตำแหน่งของเมท็อดคอลเป็นอาร์กิวเมนต์ที่ส่งผ่านภายในเมท็อดคอลอีกที	mb.add(createEditMenu(..));
		ตำแหน่งของเมท็อดคอลอยู่ภายใต้คำสั่งรีเทิร์น	return view.drawingToView(..);

		ตำแหน่งของเมท็อดคอลอยู่ภายในคำสั่งเงื่อนไข	if (target.canConnect()) {
			return target;
			}

		ตำแหน่งของเมท็อดคอลอยู่ภายใต้คำสั่งวนลูป	for (int i = 0; i < length; i++) {
			stop = new XMLElement("stop");
			elem.addChild(stop);
			}

		ตำแหน่งของเมท็อดคอลอยู่ภายในเมท็อดที่ซ้อนเมท็อดอีกที	public UndoableEdit setUndoable(..){
			edit = new AbstractUndoableEdit() {
			public void undo() {
			figure.changed();
			}
			}
			}

3	ความซับซ้อนของการเขียนโค้ดภายในเจฮอตดรอว์ เวอร์ชั่น 7.1	-	startConnector = (startFigure == null) ? null : startFigure.findConnector(..);
			setEnabled(isGroupingAction ? canGroup() : canUngroup());

เนื่องจากเจฮอตดรอว์ เวอร์ชั่น 7.1 เป็นซอฟต์แวร์ขนาดใหญ่และด้วยข้อจำกัดของภาษาแอสเป็กเจ จากลิสผลลัพธ์เมท็อดทั้งหมด 1,557 เมท็อด ผู้วิจัยจึงไม่สามารถจัดการการเรียกใช้งานเมท็อดทั้งหมดได้ ผู้วิจัยจึงกำหนดเกณฑ์ในการเลือกเมท็อดที่จะนำมาจัดการการเรียกใช้งานเมท็อดที่ซ้ำกันด้วยแอสเป็ก โดยเกณฑ์สำหรับการเลือกหน่วยตัวอย่างมีดังนี้

(1) การพิจารณาจากตำแหน่งของเมท็อดคอลที่สามารถจัดการด้วยแอสเป็กได้ โดยการพิจารณาพอยท์คัทที่รองรับสำหรับการจัดการด้วยภาษาแอสเป็กเจ เมท็อดคอลที่จะจัดการด้วยแอสเป็กจะต้องมีพอยท์คัทที่รองรับและสามารถระบุไปยังตำแหน่งจอยน์พอยท์ซึ่งเป็นตำแหน่งการทำงานเดิมของเมท็อดคอลภายในคลาส ผู้วิจัยจึงเลือกเมท็อดคอลที่จะนำมาจัดการด้วยแอสเป็ก โดยอ้างอิงจากตำแหน่งของเมท็อดคอลที่สามารถจัดการด้วยแอสเป็กได้ดังนี้

- ตำแหน่งของเมท็อดคอลอยู่ในตอนต้นหรือตอนท้ายของเมท็อด ตัวอย่างจากรูปที่ 3.5 ตอนต้นของเมท็อด updateStartTime มีเมท็อดคอลที่เรียกใช้งานเมท็อด willChange() และตอนท้ายของเมท็อดมีเมท็อดคอลที่เรียกใช้งานเมท็อด changed() เป็นต้น ดังนั้นตำแหน่งของเมท็อดคอลที่เรียกใช้งานทั้งเมท็อด willChange() และ changed() สามารถจัดการด้วยแอสเป็กได้โดยการกำหนดพอยท์คัทเพื่อระบุไปยังตำแหน่งจอยน์พอยท์ ประเภทการทำงานของเมท็อด (Method execution) คือ การทำงานของเมท็อด updateStartTime และใช้โครงสร้างพอยท์คัท execution(MethodSignature)

```
public class TaskFigure{
      public TaskFigure() {}
      public void updateStartTime() {
            willChange();
            int oldValue = getStartTime();
            int newValue = 0;
            for (TaskFigure pre : getPredecessors()) {
                newValue = Math.max(newValue,pre.getStartTime() + pre.getDuration());
            }
            getStartTimeFigure().setText(Integer.toString(newValue));
            if (newValue != oldValue) {
                  for (TaskFigure succ : getSuccessors()) {
                        if (! this.isDependentOf(succ)) {
                              succ.updateStartTime();
                        }
                  }
             }
            changed();
      }
}
```

รูปที่ 3.5 ตัวอย่างเมท็อดคอลที่อยู่ในตอนต้นหรือตอนท้ายของเมท็อด

- ตำแหน่งของเมท็อดคอลอยู่ในตอนต้นหรือตอนท้ายของคอนสตรัคเตอร์คลาส ตัวอย่างจากรูปที่ 3.6 ตอนท้ายของคอนสตรัคเตอร์ของคลาส DrawingPanel มีเมท็อดคอลที่เรียกใช้งานเมท็อด addUndoableEditListener() ดังนั้นตำแหน่งของเมท็อดคอลสามารถจัดการด้วยแอสเป็กได้โดยการกำหนดพอยท์คัทเพื่อระบุไปยังตำแหน่งจอยน์พอยท์ ประเภทการทำงานของคอนสตรัคเตอร์ (Constructor execution) คือ การทำงานของคอนสตรัคเตอร์คลาส DrawingPanel และใช้โครงสร้างพอยท์คัท execution(ConstructorSignature)

```
public class DrawingPanel extends JPanel  {
    private DefaultDrawing drawing;
    private UndoRedoManager undoManager;
    private DrawingEditor editor;
    public DrawingPanel() {
        ...
        drawing = new DefaultDrawing();
        view.setDrawing(drawing);
       drawing.addUndoableEditListener(undoManager);
    }
}
```

รูปที่ 3.6 ตัวอย่างเมท็อดคอลที่อยู่ในตอนต้นหรือตอนท้ายของคอนสตรัคเตอร์คลาส

- ตำแหน่งของเมท็อดคอลอยู่ก่อนหรือหลังการกำหนดค่าตัวแปรอ็อบเจกต์หรือแอตทริบิวต์ของคลาส ตัวอย่างจากรูปที่ 3.7 ภายในเมท็อด bringToFront และ sendToBack มีเมท็อดคอลที่เรียกใช้งานเมท็อด fireAreaInvalidated() โดยเมท็อดคอลมีตำแหน่งอยู่หลังการกำหนดค่าของแอตทริบิวต์คลาส ชื่อ drawArea ดังนั้นการจัดการเมท็อดคอลด้วยแอสเป็กสามารถทำได้ด้วยการกำหนดพอยท์คัทเพื่อระบุไปยังตำแหน่งจอยน์พอยท์ ประเภทการกำหนดค่าฟิลด์ (Field write access) คือ การกำหนดค่าฟิลด์ drawArea และใช้โครงสร้างพอยท์คัท set(FieldSignature)

```
public class DefaultDrawing{
	public Rectangle2D.Double drawArea;
	public void bringToFront(Figure figure) {
		if (basicRemove(figure) != -1) {
			basicAdd(figure);
			invalidateSortOrder();
			drawArea = figure.getDrawingArea();
			fireAreaInvalidated(drawArea);
		}
	}
	public void sendToBack(Figure figure) {
		 if (basicRemove(figure) != -1) {
			basicAdd(0, figure);
			invalidateSortOrder();
			drawArea = figure.getDrawingArea();
			fireAreaInvalidated(drawArea);
		 }
	}
}
```

รูปที่ 3.7 ตัวอย่างเมท็อดคอลที่อยู่ก่อนหรือหลังการกำหนดค่าแอตทริบิวต์ของคลาส

(2) การพิจารณากลุ่มของเมท็อดคอลทั้งหมดที่ซ้ำกันว่าเป็นการเรียกใช้งานเมท็อดเดียวกันไม่ว่าจะเรียกใช้งานเมท็อดโดยตรงหรือเรียกใช้งานเมท็อดผ่านอินเตอร์เฟส (Interface)

(3) การพิจารณากลุ่มเมท็อดคอลทั้งหมดที่ซ้ำกันว่าต้องมีตำแหน่งของเมท็อดคอลกระจัดกระจายอยู่ภายในคลาสตั้งแต่ 2 คลาสขึ้นไป

(4) การกระจายของค่าจำนวนซ้ำเมท็อดคอลที่เลือกเป็นหน่วยตัวอย่าง เนื่องจากงานวิจัยนี้ต้องการเปรียบเทียบคุณภาพจากการทำแอสเป็กรีแฟคทอริงเพื่อจัดการจำนวนซ้ำของเมท็อดคอลที่แตกต่างกัน ดังนั้นจึงจำเป็นต้องมีค่าจำนวนซ้ำของเมท็อดคอลที่หลากหลาย โดยกำหนดให้มีจำนวนซ้ำของเมท็อดคอลที่แตกต่างกันประมาณ 10-15 จำนวน เนื่องจาก Hanenberg, Kleinschmager, และ Walter (2009) เลือกจัดการครอสคัทติ้งคอนเซิร์นทั้งหมดเพียง 9 คอนเซิร์นที่มีจำนวนซ้ำของโค้ดแตกต่างกัน ผู้วิจัยจึงกำหนดจำนวนซ้ำที่ต้องการศึกษาเพิ่มเติมเพื่อให้สามารถนำมาเปรียบเทียบความแตกต่างได้และค่าจำนวนซ้ำต้องมีความหลากหลาย

การเลือกหน่วยตัวอย่างผู้วิจัยเริ่มต้นจากการเลือกเมท็อดที่มีจำนวนการเรียกใช้งานเมท็อดหรือจำนวนซ้ำเมท็อดคอลมากที่สุด โดยตำแหน่งเมท็อดคอลทั้งหมดต้องสามารถจัดการด้วยแอสเป็กได้ จากการค้นหาจากลิสเมท็อดทั้ง 1,557 เมท็อดพบว่า การเรียกใช้งานเมท็อด configureAction() ของคลาส ResourceBundleUtil มีจำนวนซ้ำของเมท็อดคอลที่ซ้ำกัน 71 ตำแหน่ง แต่จากการคัดกรองตามเกณฑ์เมท็อดคอลที่สามารถจัดการด้วยแอสเป็กได้ ทำให้คงเหลือตำแหน่งของเมท็อดคอลทั้งหมด 44 ตำแหน่งซึ่งเป็นจำนวนซ้ำของเมท็อดคอลที่มากสุด ผู้วิจัยจึงเลือกกลุ่มของเมท็อดคอลที่เรียกใช้งานเมท็อด configureAction() ซ้ำกันทั้งหมด 44 ตำแหน่งเป็นหนึ่งหน่วยตัวอย่าง โดยจำนวนซ้ำของเมท็อดคอลมีค่ามากสุดเท่ากับ 44 เพื่อให้ค่าจำนวนซ้ำมีค่าหลากหลายสำหรับการนำมาเปรียบเทียบความแตกต่างของคุณภาพจากการทำแอสเป็กรีแฟคทอริง ดังนั้นผู้วิจัยจึงกำหนดช่วงค่าของจำนวนซ้ำเมท็อดคอลดังตารางที่ 3.4 เพื่อให้จำนวนซ้ำของเมท็อดคอลที่เลือกมาเป็นหน่วยตัวอย่างมีค่าจำนวนซ้ำกระจายอยู่ภายในช่วงค่าจำนวนซ้ำที่แตกต่างกัน โดยมีระยะห่างระหว่างช่วงค่าจำนวนซ้ำเท่ากับ 5

ตารางที่ 3.4 ช่วงค่าของจำนวนซ้ำเมท็อดคอลที่ใช้เลือกหน่วยตัวอย่าง

ลำดับของจำนวนซ้ำเมท็อดคอล	**ช่วงค่าของจำนวนซ้ำเมท็อดคอล**
1	1-5
2	6-10
3	11-15
4	16-20
5	21-25
6	26-30
7	31-35
8	36-40
9	41-45

จากการวิเคราะห์ผลลัพธ์เมท็อดทั้ง 1,557 เมท็อดที่ได้จากการค้นหาด้วยเครื่องมือเอ็ฟไอเอ็นที โดยผู้วิจัยพยายามคัดกรองลิสเมท็อดให้ได้จำนวนซ้ำของเมท็อดคอลครบตามช่วงค่าจำนวนซ้ำของเมท็อดคอลที่กำหนดไว้ในตารางที่ 3.4 ข้างต้น จากผลการคัดกรองผู้วิจัยได้พยายามเลือกเมท็อดที่ได้เพื่อให้ครอบคลุมทุกช่วงค่าจำนวนซ้ำที่แตกต่างกันให้มากที่สุด แต่เนื่องจากจำนวนซ้ำของเมท็อดคอลที่มากขึ้นทำให้การหาจำนวนซ้ำให้ครอบคลุมครบทุกช่วงค่าจำนวนซ้ำของเมท็อดคอลทำได้ยาก ผลลัพธ์ที่ได้จึงได้ค่าจำนวนซ้ำที่ครอบคลุมเพียง 8 ช่วงดังตารางที่ 3.5

ตารางที่ 3.5 ผลลัพธ์ลิสเมท็อดที่ได้จากการคัดกรองแยกตามช่วงค่าจำนวนซ้ำ

ลำดับ	ช่วงค่าของจำนวนซ้ำเมท็อดคอล	ผลลัพธ์จากการคัดกรองลิสเมท็อด	
		เมท็อด	จำนวนซ้ำของเมท็อดคอล
1	1-5	fireAreaInvalidated(..) discardAllEdits()	4 5
2	6-10	addToSelection(..)	9
3	11-15	clearSelection() removeAllChildren() firePropertyChange(..) addUndoableEditListener(..)	12 12 13 15
4	16-20	grow(..)	18
5	21-25	createElement(..)	21
6	26-30	changed()	29
7	31-35	-	-
8	36-40	willChange()	37
9	41-45	configureAction(..)	44

จากผลลัพธ์การเลือกหน่วยตัวอย่างทำให้ได้ลิสเมท็อดดังตารางที่ 3.6 แสดงลิสเมท็อดที่ถูกเรียกใช้งานจากคลาสต่างๆ ทั้งหมด 12 เมท็อด และจำนวนโค้ดโคลนซึ่งเป็นจำนวนซ้ำของเมท็อดคอลที่เรียกใช้งานเมท็อดเดียวกัน มีจำนวนซ้ำของเมท็อดคอลแตกต่างกันทั้งหมด 11 จำนวน โดยจำนวนซ้ำของเมท็อดคอลในตารางเป็นจำนวนที่ผ่านการคัดกรองให้คงเหลือแค่เมท็อดคอลที่สามารถจัดการด้วยแอสเป็กได้ การทำแอสเป็กรีแฟคทอริงในงานวิจัยนี้จึงเป็นการจัดการเมท็อดคอลทั้งหมดในแต่ละจำนวนซ้ำด้วยแอสเป็ก โดยเมท็อดคอลทั้งหมดเป็นการเรียกใช้งานเมท็อดหนึ่งเมท็อดจากลิสผลลัพธ์ในตารางที่ 3.6

ตารางที่ 3.6 ลิสเมท็อดและจำนวนซ้ำเมท็อดคอลสำหรับทำแอสเป็กรีแฟคทอริง

ลำดับ	คลาส	เมท็อดของคลาส	จำนวนซ้ำเมท็อดคอล
1	AbstractFigure	fireAreaInvalidated(..)	4
2	UndoRedoManager	discardAllEdits()	5
3	DefaultDrawingView	addToSelection(..)	9
4	DefaultDrawingView	clearSelection()	12
5	AbstractCompositeFigure	removeAllChildren()	12
6	AbstractBean	firePropertyChange(..)	13
7	AbstractDrawing	addUndoableEditListener(..)	15
8	Geom	grow(..)	18
9	XMLElement	createElement(..)	21
10	AbstractFigure	changed()	29
11	AbstractFigure	willChange()	37
12	ResourceBundleUtil	configureAction(..)	44

3. การทำแอสเป็กรีแฟคทอริง

การทำแอสเป็กรีแฟคทอริงสำหรับจัดการเมท็อดคอลซ้ำกันในแต่ละจำนวนซ้ำตามตารางที่ 3.6 จะใช้ซอฟต์แวร์ตั้งต้นเดียวกันคือเจฮอตดรอว์ เวอร์ชั่น 7.1 โดยก่อนการทำแอสเป็กรีแฟคทอริง ผู้วิจัยจำเป็นต้องแก้ไขซอร์สโค้ดของเจฮอตดรอว์ เวอร์ชั่น 7.1 เพื่อให้สามารถจัดการเมท็อดคอลซ้ำกันทั้งหมดที่ระบุด้วยภาษาแอสเป็กเจได้ การแก้ไขจะแก้เพียงลำดับของโค้ดหรือโครงสร้างของคลาส ดังนั้นการทำงานของซอฟต์แวร์จึงยังคงเดิม รายละเอียดการแก้ไขซอร์สโค้ดของเจฮอตดรอว์ เวอร์ชั่น 7.1 มีรายละเอียดดังนี้

(1) การแก้ไขระดับการเข้าถึง (Access Modifier) ของเมท็อดในลิสตารางที่ 3.6 โดยหากระดับการเข้าถึงของเมท็อดเป็นไพรเวท (Private) หรือโพรเทค (Protected) ต้องแก้ไขเป็นประเภทพับบลิค (Public) เนื่องจากเมท็อดเหล่านี้ถูกเรียกใช้งานจากเมท็อดคอลที่อยู่ภายในคลาสต่างๆ โดยผู้วิจัยเลือกสร้างแอสเป็กภายใต้แพคเกจใหม่ คือ org.jhotdraw.aspect เป็นการควบคุมการสร้างแอสเป็กให้มีแบบแผนเหมือนกัน และการจัดการเมท็อดคอลซ้ำกันหลายตำแหน่งมาไว้ในแอสเป็กเพื่อให้ภายในแอสเป็กยังคงสามารถเรียกใช้งานเมท็อดได้ ระดับการเข้าถึงของเมท็อดจึงต้องแก้ไขเป็นพับบลิค ตัวอย่างดังรูปที่ 3.8

```
package org.jhotdraw.draw;
public abstract class AbstractFigure extends AbstractBean implements Figure {
    protected EventListenerList listenerList = new EventListenerList();
    ...  public
    protected void fireAreaInvalidated(Rectangle2D.Double invalidatedArea) {
        if (listenerList.getListenerCount() > 0) {
            FigureEvent event = null;
            Object[] listeners = listenerList.getListenerList();
            for (int i = listeners.length - 2; i >= 0; i -= 2) {
                if (listeners[i] == FigureListener.class) {
                    if (event == null) {
                        event = new FigureEvent(this, invalidatedArea);
                    }
                    ((FigureListener) listeners[i + 1]).areaInvalidated(event);
                }
            }
        }
    }
}
```

รูปที่ 3.8 การแก้ไขระดับการเข้าถึงของเมท็อด fireAreaInvalidated

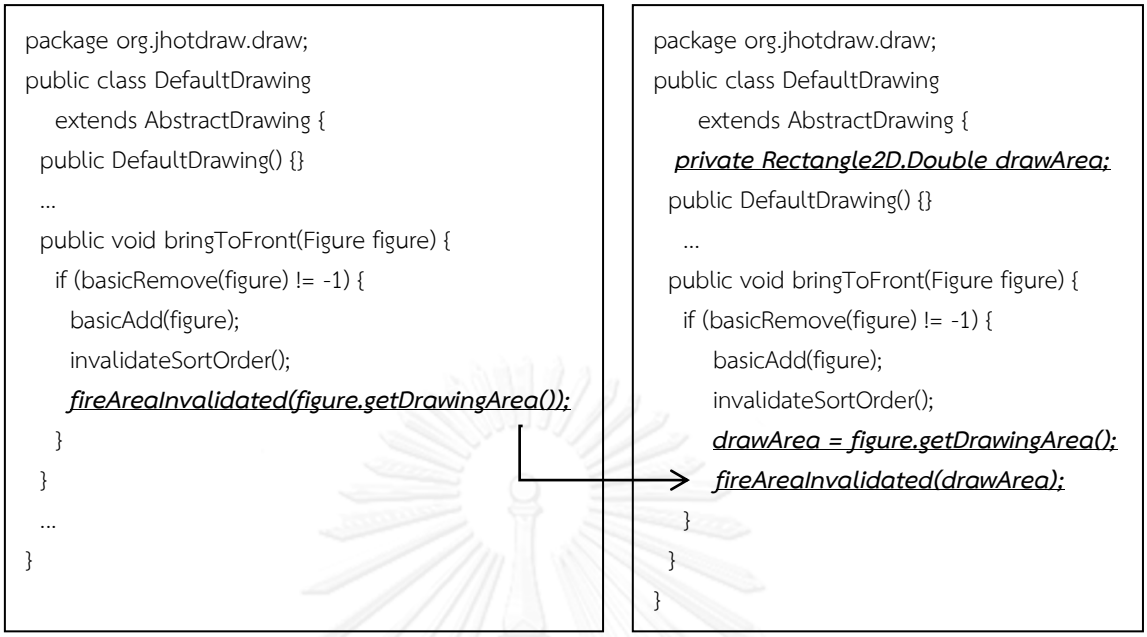
(2) การแก้ไขตัวแปรโลคอล (Local variables) คือตัวแปรที่ประกาศไว้ภายในเมท็อด แก้ไขให้เป็นตัวแปรของคลาสพร้อมกำหนดระดับการเข้าถึงของตัวแปรเป็นแบบไพรเวท (Private) โดยแก้ไขแค่บางตัวแปรโลคอล การแก้ไขมีทั้งหมดสองกรณีดังนี้ กรณีแรกคือตำแหน่งของเมท็อดคอลที่ต้องการจัดการด้วยแอสเป็กอยู่ก่อนหรือหลังการกำหนดค่าของตัวแปร กรณีที่สองคือตัวแปรดังกล่าวต้องนำมาใช้งานในเมท็อดคอล ดังนั้นจากทั้งสองกรณีเพื่อให้แอสเป็กสามารถกำหนดพอยท์คัทเพื่อระบุมายังตำแหน่งของตัวแปรหรือเข้าถึงค่าตัวแปร จึงจำเป็นต้องแก้ไขตัวแปรโลคอลเป็นตัวแปรคลาส ตัวอย่างดังรูปที่ 3.9 การจัดการเมท็อดคอลที่เรียกใช้งานเมท็อด changed() ด้วยแอสเป็ก โดยเรียกใช้งานผ่านตัวแปรอ็อบเจกต์ textOwner เพื่อให้แอสเป็กสามารถเข้าถึงตัวแปร textOwner จึงต้องแก้ไขการประกาศตัวแปรจากการประกาศภายในเมท็อดแก้ไขให้เป็นตัวแปรของคลาสแทน

```
package org.jhotdraw.draw;
public class FontSizeHandle extends LocatorHandle {
    private float oldSize;
    private float newSize;
    private Object restoreData;
    private TextHolderFigure textOwner;
    public void trackStep(Point anchor, Point lead, int modifiersEx) {
        TextHolderFigure textOwner = (TextHolderFigure) getOwner();
        Point2D.Double anchor2D = view.viewToDrawing(anchor);
        Point2D.Double lead2D = view.viewToDrawing(lead);
        if (TRANSFORM.get(textOwner) != null) {
            try {
                TRANSFORM.get(textOwner).inverseTransform(anchor2D, anchor2D);
                TRANSFORM.get(textOwner).inverseTransform(lead2D, lead2D);
            } catch (NoninvertibleTransformException ex) {
                ex.printStackTrace();
            }
        }
        newSize = (float) Math.max(1, oldSize + lead2D.y - anchor2D.y);
        textOwner.willChange();
        textOwner.setFontSize(newSize);
        textOwner.changed();
    }
}
```

รูปที่ 3.9 การแก้ไขการประกาศตัวแปร

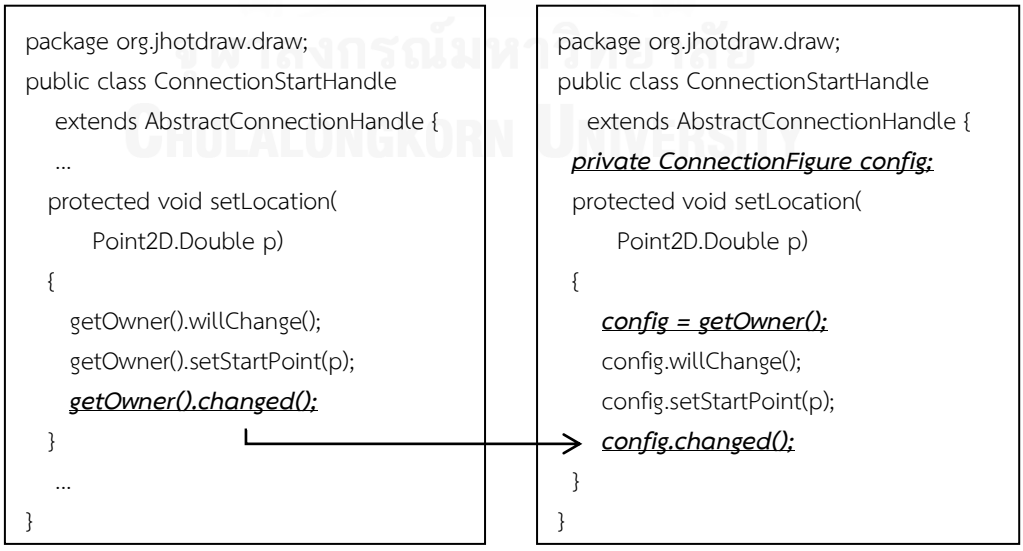
(3) การแก้ไขกรณีคำสั่งเป็นเมท็อดคอลที่ภายในมีการเรียกใช้งานเมท็อดมากกว่า 1 เมท็อด ตัวอย่างเช่น fireAreaInvalidated(figure.getDrawingArea()); และ getOwner().changed(); เป็นต้น การแก้ไขโค้ดทำเพื่อควบคุมเมท็อดคอลที่จัดการด้วยแอสเป็กให้เป็นเมท็อดคอลที่เรียกใช้งานเมท็อดเพียงเมท็อดเดียว สามารถแสดงได้ทั้งหมดสองกรณีดังนี้

กรณีแรกพารามิเตอร์เป็นเมท็อดคอลที่เรียกใช้งานเมท็อดอื่น ส่งผลให้คำสั่งเป็นเมท็อดคอลซ้อนภายในเมท็อดคอลอีกที ดังรูปที่ 3.10 บล็อกด้านซ้ายเป็นซอร์สโค้ดก่อนแก้ไขของคลาส DefaultDrawing โดยภายในคลาสมีเมท็อดคอลที่เรียกใช้งานเมท็อด fireAreaInvalidated() ที่สนใจจัดการด้วยแอสเป็กแต่พารามิเตอร์ที่ส่งเป็นเมท็อดคอลที่เรียกใช้งานเมท็อดอื่น เพื่อควบคุมให้เมท็อดคอลที่จัดการด้วยแอสเป็กให้เป็นการเรียกใช้งานเมท็อด fireAreaInvalidated() เพียงเมท็อดเดียว จำเป็นต้องแก้ไขคลาส DefaultDrawing แสดงด้วยบล็อกทางขวา แก้ไขโดยการสร้างตัวแปรคลาส drawArea สำหรับเก็บค่าจาก figure.getDrawingArea() และส่งเป็นพารามิเตอร์ของเมท็อดคอล



รูปที่ 3.10 การแก้ไขซอร์สโค้ดของคลาส DefaultDrawing

กรณีที่สองการเรียกใช้งานเมท็อดผ่านเมท็อดคอลอื่น ส่งผลให้ภายในคำสั่งประกอบด้วยเมท็อดคอล 2 เมท็อดคอล ดังรูปที่ 3.11 บล็อกทางด้านซ้ายเป็นซอร์สโค้ดก่อนแก้ไขของคลาส ConnectionStartHandle โดยภายในคลาสมีเมท็อดคอลที่เรียกใช้งานเมท็อด changed() ที่สนใจจัดการด้วยแอสเป็ก แต่การเรียกใช้งานเมท็อดดังกล่าวถูกเรียกใช้งานผ่านเมท็อดคอล getOwner() เพื่อควบคุมเมท็อดคอลที่จัดการด้วยแอสเป็กให้เป็นการเรียกใช้งานเมท็อด changed() เพียงเมท็อดเดียว จำเป็นต้องแก้ไขโค้ดของคลาส ConnectionStartHandle แสดงด้วยบล็อกทางขวา แก้ไขโดยสร้างตัวแปรคลาส config สำหรับเก็บค่าจากการเรียกใช้งานเมท็อด getOwner() ทำให้สามารถเรียกใช้งานเมท็อด changed() ผ่านตัวแปรคลาส config แทน



รูปที่ 3.11 การแก้ไขซอร์สโค้ดของคลาส ConnectionStartHandle

จากจำนวนคลาสทั้งหมด 442 คลาสของเจฮอตดรอว์ เวอร์ชั่น 7.1 การปรับปรุงซอร์สโค้ดจะเริ่มต้นจากการแก้ไขระดับการเข้าถึงของเมท็อดในลิสผลลัพธ์ 12 เมท็อด โดยแก้ไขเพียงบางเมท็อดที่มีระดับการเข้าถึงเป็นไพรเวทหรือโพรเทคดังกล่าวไว้ในรูปแบบการแก้ไขที่ 1 จากนั้นจึงมาพิจารณาเมท็อดคอลทั้งหมดในแต่ละจำนวนซ้ำของเมท็อดคอล โดยจะแก้ไขเพียงบางเมท็อดคอลที่มีการเข้าถึงตัวแปรโลคอลหรือเมท็อดคอลมีการเรียกใช้งานเมท็อดมากกว่า 1 เมท็อด จะแก้ไขเมท็อดคอลดังกล่าวด้วยรูปแบบการแก้ไขที่ 2 และ 3 ตามลำดับ จึงทำให้มีบางคลาสภายในซอร์สโค้ดของเจฮอตดรอว์ เวอร์ชั่น 7.1 ที่ได้รับแก้ไข โดยในกรณีที่ภายในคลาสมีตำแหน่งเมท็อดคอลหลายตำแหน่งที่เรียกใช้งานเมท็อดจากลิสผลลัพธ์ 12 เมท็อดอยู่ภายในคลาสเดียวกันและเป็นเมท็อดคอลที่ต้องได้รับการแก้ไข จะส่งผลให้ภายในคลาสดังกล่าวมีการแก้ไขเมท็อดคอลมากกว่า 1 ตำแหน่ง จากการแก้ไขซอร์สโค้ดของเจฮอตดรอว์ เวอร์ชั่น 7.1 ในแต่ละจำนวนซ้ำของเมท็อดคอลจึงสามารถแสดงจำนวนคลาสที่ได้รับการแก้ไขแยกตามรูปแบบการแก้ไขได้ดังตารางที่ 3.7 โดยรูปแบบการแก้ไขระบุด้วยหมายเลขภายในตารางดังนี้

(1) การแก้ไขระดับการเข้าถึง (Access Modifier) ของเมท็อด

(2) การแก้ไขตัวแปรโลคอล (Local variables) เป็นตัวแปรคลาส

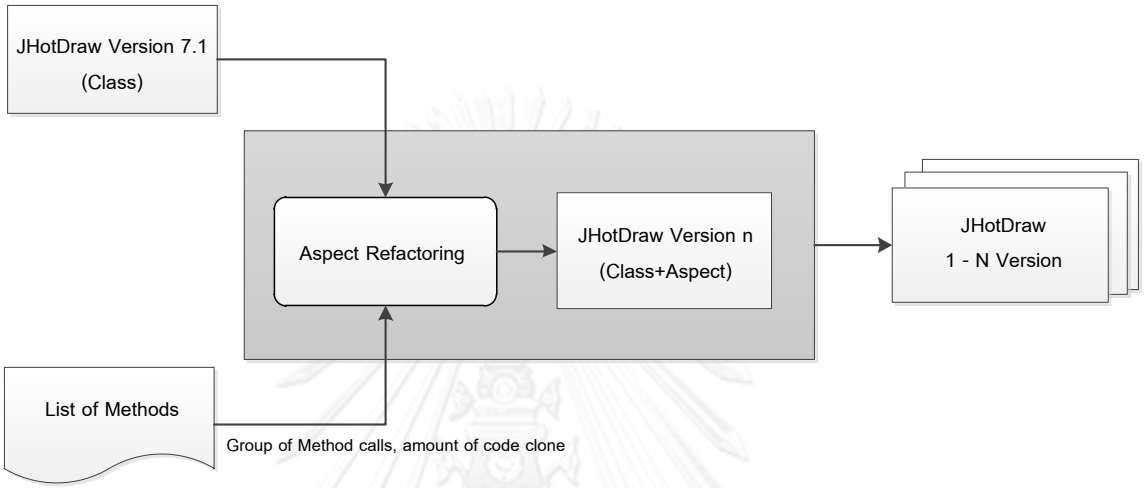
(3) การแก้ไขกรณีคำสั่งเป็นเมท็อดคอลที่มีการเรียกใช้งานเมท็อดมากกว่า 1 เมท็อด

เมื่อปรับปรุงซอร์สโค้ดเรียบร้อยแล้ว ผู้วิจัยจึงกำหนดให้ซอร์สโค้ดที่ได้รับการแก้ไขเป็นซอร์สโค้ดของเจฮอตดรอว์ เวอร์ชั่น 7.1 ที่ใช้เป็นซอฟต์แวร์ตั้งต้นสำหรับการทำแอสเป็กรีแฟคทอริง

ตารางที่ 3.7 รายละเอียดการแก้ไขซอร์สโค้ดของเจฮอตดรอว์ เวอร์ชั่น 7.1

จำนวนซ้ำเมท็อดคอล	**รูปแบบการแก้ไขซอร์สโค้ดของเจฮอตดรอว์เวอร์ชั่น 7.1 (จำนวนคลาส)**		
	(1)	**(2)**	**(3)**
4	1	-	2
5	-	-	-
9	-	-	4
12	-	7	4
12	-	2	5
13	1	6	-
15	-	5	5
18	-	10	2
21	-	2	-
29	-	7	3
37	-	9	-
44	-	26	-

กระบวนการแอสเป็กรีแฟคทอริง (Aspect refactoring) เป็นกระบวนการสำหรับจัดการครอสคัทติ้งคอนเซิร์นเป็นแอสเป็กด้วยการใช้โครงสร้างภาษาของการโปรแกรมเชิงแอสเป็ก การทำแอสเป็กรีแฟคทอริงในแต่ละครั้งเลือกจัดการเมท็อดคอลที่ซ้ำกันที่เกิดจากการเรียกใช้งานเมท็อดตามลิสผลลัพธ์ที่ได้จากขั้นตอนสองของการค้นหาครอสคัทติ้งคอนเซิร์นและมีจำนวนโค้ดโคลนตามที่ระบุ ขั้นตอนโดยรวมของการทำแอสเป็กรีแฟคทอริงแสดงดังรูปที่ 3.12



รูปที่ 3.12 ขั้นตอนการทำแอสเป็กรีแฟคทอริง

จากขั้นตอนการเลือกหน่วยตัวอย่างทำให้ได้ลิสเมท็อดพร้อมผลลัพธ์การเรียกใช้งานเมท็อดจากคลาสอื่นๆและจำนวนซ้ำของเมท็อดคอล การทำแอสเป็กรีแฟคทอริงในแต่ละครั้งจึงกำหนดให้ใช้ซอฟต์แวร์ตั้งต้นเดียวกันคือเจฮอตดรอว์ เวอร์ชั่น 7.1 ที่ได้ปรับปรุงซอร์สโค้ดเรียบร้อยแล้ว จากนั้นจึงทำแอสเป็กรีแฟคทอริงโดยเลือกจัดการหนึ่งเมท็อดจากลิสผลลัพธ์ที่ได้ในขั้นตอนที่สองเพื่อจัดการเมท็อดคอลที่ซ้ำกันทั้งหมดตามจำนวนโค้ดโคลนที่ระบุไว้ด้วยแอสเป็ก การทำแอสเป็กรีแฟคทอริงอ้างอิงจากเทคนิคเอ็กซ์แทรคเมท็อดคอล (Extract method call) ของ Laddad (2003) เมื่อทำแอสเป็กรีแฟคทอริงเสร็จเรียบร้อยจะได้ซอฟต์แวร์เจฮอตดรอว์เวอร์ชั่นใหม่ที่ภายในซอร์สโค้ดมีไฟล์คลาสและไฟล์แอสเป็ก จากนั้นดำเนินการทำแอสเป็กรีแฟคทอริงซ้ำจนกระทั่งจัดการครบตามลิสผลลัพธ์เมท็อดที่ได้จากขั้นตอนก่อนหน้า สุดท้ายของขั้นตอนนี้จึงได้ซอฟต์แวร์เจฮอตดรอว์เวอร์ชั่นต่างๆ ที่แต่ละเวอร์ชั่นเป็นการจัดการเมท็อดคอลที่มีจำนวนซ้ำของเมท็อดคอลแตกต่างกัน

4. การตรวจสอบความถูกต้องของซอฟต์แวร์

หลังจากทำแอสเป็กรีแฟคทอริงเสร็จเรียบร้อยแล้ว นำเจฮอตดรอว์ทุกเวอร์ชั่นมาตรวจสอบความถูกต้อง เพื่อเปรียบเทียบผลลัพธ์การทำงานของซอฟต์แวร์หลังทำแอสเป็กรีแฟคทอริงกับซอฟต์แวร์ตั้งต้นเจฮอตดรอว์ เวอร์ชั่น 7.1 เพื่อให้แน่ใจว่าหลังจากปรับปรุงซอฟต์แวร์การทำงานของซอฟต์แวร์ยังคงเดิม

5. การวัดคุณภาพซอฟต์แวร์

เมื่อการทำงานของเจฮอตดรอว์ทุกเวอร์ชั่นถูกต้องตรงกันกับซอฟต์แวร์ตั้งต้นแล้ว ขั้นตอนนี้จึงนำเจฮอตดรอว์แต่ละเวอร์ชั่นมาเก็บรวบรวมค่ามาตรวัดของมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็กเพื่อนำไปประเมินคุณภาพซอฟต์แวร์ รายละเอียดของขั้นตอนมีดังนี้

5.1. วัดมาตรวัดเชิงวัตถุ นำเจฮอตดรอว์ทุกเวอร์ชั่นที่ได้จากการทำแอสเป็กรีแฟคทอริงและเจฮอตดรอว์ เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้นมาเก็บรวบรวมค่ามาตรวัดเชิงวัตถุของ Chidamber และ Kemerer (1993) ทั้งหมดหกมาตรวัดด้วยเครื่องมือซีเคเจเอ็ม (CKJM) เครื่องมือดังกล่าวจะใช้งานผ่านบิวด์ทูล (Build Tools) ชื่ออาปาเช่แอนท์ (Apache Ant) ด้วยการระบุชุดคำสั่งของแอนท์

5.2. วัดมาตรวัดเชิงแอสเป็ก นำเจฮอตดรอว์ทุกเวอร์ชั่นที่ได้จากการทำแอสเป็กรีแฟคทอริงและเจฮอตดรอว์ เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้นมาเก็บรวบรวมค่ามาตรวัดเชิงแอสเป็กของ Ceccato และ Tonella (2004) ทั้งหมดสิบมาตรวัดด้วยเครื่องมือเอโอพีเมตริก (AopMetrics) โดยใช้งานผ่านบิวด์ทูลอาปาเช่แอนท์เช่นกัน

6. การวิเคราะห์ผลการทดลอง

เบื้องต้นนำค่ามาตรวัดเชิงวัตถุและค่ามาตรวัดเชิงแอสเป็กของเจฮอตดรอว์ทุกเวอร์ชั่นและเจฮอตดรอว์ เวอร์ชั่น 7.1 มาหาค่าสถิติเชิงพรรณนา (Descriptive statistics) เพื่อพิจารณาความแตกต่างระหว่างการจัดการเมท็อดคอลที่มีจำนวนซ้ำมากหรือน้อย

3.7 ประเด็นของความถูกต้อง (Validity) และความน่าเชื่อถือ (Reliability) ของข้อมูลที่เก็บ

เนื่องจากงานวิจัยเปรียบเทียบคุณภาพซอฟต์แวร์จากการจัดการเมท็อดคอลที่มีจำนวนซ้ำของเมท็อดคอลที่แตกต่างกัน ผู้วิจัยจึงพยายามควบคุมปัจจัยที่เกี่ยวข้องเพื่อให้ได้ข้อมูลที่ถูกต้อง (Valid) และเชื่อถือได้ (Reliable) สำหรับตอบวัตถุประสงค์ของงานวิจัย ดังนี้

(1) การเลือกจัดการครอสคัทติ้งคอนเซิร์น งานวิจัยนี้เป็นการเปรียบเทียบคุณภาพซอฟต์แวร์ที่จัดการครอสคัทติ้งคอนเซิร์น โดยพิจารณาความแตกต่างจากจำนวนโค้ดโคลนที่เกิดจากการอิมพลีเม้นต์ครอสคัทติ้งคอนเซิร์น ดังนั้นเพื่อควบคุมให้ค่ามาตรวัดที่ได้เกิดความแตกต่างมาจากจำนวนโค้ดโคลนอย่างแท้จริง ผู้วิจัยจึงได้ควบคุมปัจจัยดังกล่าวด้วยการควบคุมลักษณะของครอสคัทติ้งคอนเซิร์นที่เลือกจัดการด้วยแอสเป็ก โดยกำหนดให้จัดการครอสคัทติ้งคอนเซิร์นที่มีลักษณะเมท็อดคอลเพียงอย่างเดียว ทำให้ค่ามาตรวัดที่ได้เกิดความแตกต่างมาจากจำนวนซ้ำของเมท็อดคอล และไม่ได้มีผลมาจากโค้ดที่เกี่ยวข้องกับครอสคัทติ้งคอนเซิร์นมีความแตกต่างกัน

(2) การทำแอสเป็กรีแฟคทอริงสำหรับในแต่ละจำนวนโค้ดโคลนมีความจำเป็นต้องควบคุมสภาพแวดล้อมต่างๆในการทดลองให้มีความเหมือนกันมากที่สุด เพื่อไม่ให้ปัจจัยอื่นๆมากระทบต่อค่ามาตรวัด ดังนั้นสำหรับการทำแอสเป็กรีแฟคทอริงในแต่ละจำนวนซ้ำของเมท็อดคอลจึงกำหนดให้ใช้ซอฟต์แวร์ตั้งต้นเดียวกันคือ เจฮอตดรอว์ เวอร์ชั่น 7.1 เพื่อควบคุมให้ซอฟต์แวร์ก่อนปรับปรุงมีโค้ดการทำงานและฟังก์ชันการทำงานเหมือนกัน เป็นการควบคุมให้ได้ผลมาตรวัดที่สามารถนำมาเปรียบเทียบได้อย่างถูกต้อง

(3) การตรวจสอบการทำงานของซอฟต์แวร์ก่อนและหลังปรับปรุง เนื่องจากการทำแอสเป็กรีแฟคทอริงเป็นการจัดเรียงโค้ดใหม่โดยยังคงพฤติกรรมเดิมของซอฟต์แวร์เอาไว้ ดังนั้นเพื่อให้แน่ใจว่าการทำงานของซอฟต์แวร์ยังคงเดิมหลังจากจัดการเมท็อดคอลที่ซ้ำด้วยแอสเป็ก ผู้วิจัยจึงใช้ฟังก์ชันการดีบัก (Debugging) ที่มีในโปรแกรมอีคลิปส์รันเพื่อตรวจสอบค่าของตัวแปรและอ็อบเจกต์ว่าถูกต้องคงเดิมเหมือนซอฟต์แวร์ตั้งต้น เพื่อให้แน่ใจว่าค่ามาตรวัดที่วัดหลังจากทำแอสเป็กรีแฟคทอริงเชื่อถือได้

3.8 กรอบการวิเคราะห์ข้อมูล

เนื่องจากงานวิจัยมีวัตถุประสงค์ทั้งหมด 3 ข้อ ได้แก่ (1) เพื่อเปรียบเทียบมาตรวัดเชิงวัตถุของซอฟต์แวร์ที่ปรับปรุงด้วยการทำแอสเป็กรีแฟคทอริงจากจำนวนโค้ดโคลนที่แตกต่างกัน (2) เพื่อเปรียบเทียบมาตรวัดเชิงแอสเป็กของซอฟต์แวร์ที่ปรับปรุงด้วยการทำแอสเป็กรีแฟคทอริงจากจำนวนโค้ดโคลนที่แตกต่างกัน และ (3) เพื่อวิเคราะห์คุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริงด้วยจำนวนโค้ดโคลนที่แตกต่างกัน

สำหรับการตอบวัตถุประสงค์ในข้อ 1 และ 2 การวิเคราะห์ข้อมูลเพื่อตอบวัตถุประสงค์ทั้งสองข้อมีแนวทางการพิจารณาเหมือนกัน โดยสามารถแบ่งการวิเคราะห์ออกเป็น 2 ส่วน ดังนี้

(1) การเปรียบเทียบค่ามาตรวัดก่อนและหลังทำแอสเป็กรีแฟคทอริง เพื่อพิจารณามาตรวัดที่ได้รับผลกระทบจากการทำแอสเป็กรีแฟคทอริง โดยการนำค่ามาตรวัดของคลาสทั้งหมดในเจฮอตดรอว์แต่ละเวอร์ชั่นมาเทียบกับค่ามาตรวัดของคลาสทั้งหมดในเจฮอตดรอว์ เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้น เพื่อแสดงคลาสที่มีค่าเปลี่ยนแปลงหลังทำแอสเป็กรีแฟคทอริง เมื่อพิจารณาเจฮอตดรอว์ทุกเวอร์ชั่นทำให้ทราบมาตรวัดที่ได้รับผลกระทบเมื่อทำแอสเป็กรีแฟคทอริงโดยการจัดการเมท็อดคอลซ้ำกันทั้งหมดด้วยแอสเป็ก

(2) การเปรียบเทียบค่าเฉลี่ยมาตรวัดที่คำนวณจากคลาสทั้งหมดของเจฮอตดรอว์แต่ละเวอร์ชั่น โดยเบื้องต้นผู้วิจัยนำค่ามาตรวัดของคลาสทั้งหมดในเจฮอตดรอว์แต่ละเวอร์ชั่นมาวิเคราะห์ข้อมูลในลักษณะสถิติเชิงพรรณนา (Descriptive statistic) เมื่อได้ข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์ทุกเวอร์ชั่นแล้ว จึงนำค่าเฉลี่ย (Mean) ของแต่ละมาตรวัดมาเปรียบเทียบกันระหว่างเวอร์ชั่นของเจฮอตดรอว์ เพื่อพิจารณาความแตกต่างหรือแนวโน้มของค่าเฉลี่ยมาตรวัดจากการจัดการเมท็อดคอลที่มีจำนวนซ้ำน้อยหรือมากด้วยแอสเป็ก จากการวิเคราะห์สามารถแสดงผลของจำนวนซ้ำเมท็อดคอลต่อมาตรวัดซอฟต์แวร์ ทั้งมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็ก

สำหรับการตอบวัตถุประสงค์ข้อ 3 จากการวิเคราะห์มาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็กที่ตอบวัตถุประสงค์ข้อ 1 และ 2 ทำให้ทราบลิสมาตรวัดที่ได้รับผลกระทบจากการจัดการเมท็อดคอลด้วยแอสเป็กในจำนวนซ้ำของเมท็อดคอลที่แตกต่างกัน โดยสามารถนำลิสมาตรวัดมาวิเคราะห์คุณภาพซอฟต์แวร์ด้วยการพิจารณาจากความสัมพันธ์ระหว่างมาตรวัดและคุณภาพซอฟต์แวร์ แสดงให้เห็นมาตรวัดที่ส่งผลให้คุณภาพซอฟต์แวร์เปลี่ยนแปลง

บทที่ 4
ผลการวิเคราะห์ข้อมูล

4.1 บทนำ

ในบทนี้จะนำเสนอถึงผลและการวิเคราะห์ข้อมูลที่ได้จากการทดลอง เพื่อตอบวัตถุประสงค์ของงานวิจัย ได้แก่ (1) เปรียบเทียบมาตรวัดเชิงวัตถุของซอฟต์แวร์ที่ปรับปรุงด้วยการทำแอสเป็กรีแฟคทอริงจากจำนวนโค้ดโคลนที่แตกต่างกัน (2) เปรียบเทียบมาตรวัดเชิงแอสเป็กของซอฟต์แวร์ที่ปรับปรุงด้วยการทำแอสเป็กรีแฟคทอริงจากจำนวนโค้ดโคลนที่แตกต่างกัน และ (3) วิเคราะห์คุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริงด้วยจำนวนโค้ดโคลนที่แตกต่างกัน ซึ่งจะประกอบด้วยหัวข้อย่อยดังนี้ (1) ผลการทำแอสเป็กรีแฟคทอริง (2) ผลการวิเคราะห์ข้อมูลเบื้องต้น (3) การเปรียบเทียบค่ามาตรวัดเชิงวัตถุ (4) การเปรียบเทียบค่ามาตรวัดเชิงแอสเป็ก และ (5) การอภิปรายผลและการศึกษาเพิ่มเติม

4.2 ผลการทำแอสเป็กรีแฟคทอริง

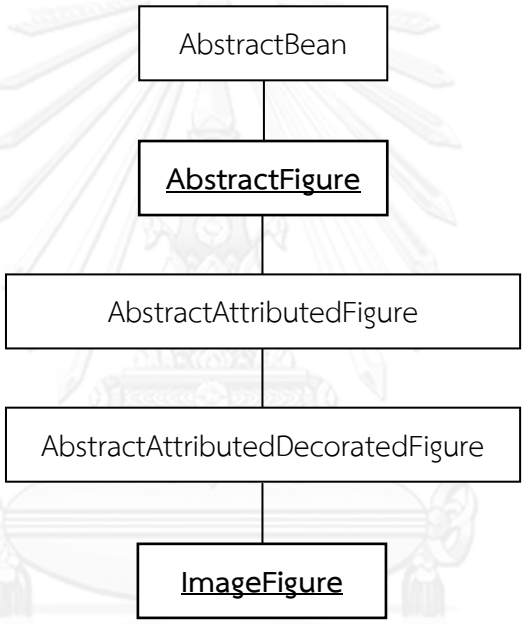
การทำแอสเป็กรีแฟคทอริงสำหรับงานวิจัยนี้ ผู้วิจัยเลือกจัดการเมท็อดคอลที่ซ้ำกันภายในซอร์สโค้ดด้วยแอสเป็ก กรณีศึกษาที่ใช้ในการทดลองคือเจฮอตดรอว์ เวอร์ชั่น 7.1 จากการค้นหาครอสคัทติ้งคอนเซิร์นภายในซอร์สโค้ดของเจฮอตดรอว์ เวอร์ชั่น 7.1 ทำให้ได้รายชื่อเมท็อดทั้งหมด 12 เมท็อด โดยแต่ละเมท็อดมีจำนวนซ้ำของเมท็อดคอลแตกต่างกัน 11 จำนวน ดังตารางที่ 4.1

ตารางที่ 4.1 ลิสเมท็อดและจำนวนซ้ำเมท็อดคอลสำหรับทำแอสเป็กรีแฟคทอริง

ลำดับ	คลาส	เมท็อดของคลาส	จำนวนซ้ำเมท็อดคอล
1	AbstractFigure	fireAreaInvalidated(..)	4
2	UndoRedoManager	discardAllEdits()	5
3	DefaultDrawingView	addToSelection(..)	9
4	DefaultDrawingView	clearSelection()	12
5	AbstractCompositeFigure	removeAllChildren()	12
6	AbstractBean	firePropertyChange(..)	13
7	AbstractDrawing	addUndoableEditListener(..)	15
8	Geom	grow(..)	18
9	XMLElement	createElement(..)	21
10	AbstractFigure	changed()	29
11	AbstractFigure	willChange()	37
12	ResourceBundleUtil	configureAction(..)	44

จากตารางที่ 4.1 แต่ละจำนวนซ้ำเมท็อดคอลอาจประกอบด้วยเมท็อดคอลที่มีลักษณะแตกต่างกัน ผู้วิจัยจึงวิเคราะห์เมท็อดคอลทั้งหมดของแต่ละจำนวนซ้ำเมท็อดคอล และจำแนกรูปแบบของลักษณะเมท็อดคอลทั้งหมด 5 รูปแบบ ดังนี้

(1) ลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดจากภายในคลาสโดยตรง เนื่องจากภายในคลาสอิมพลีเม้นต์เมท็อดที่ต้องการเรียกใช้งานไว้อยู่แล้ว หรือเป็นคลาสลูกอยู่ในลำดับชั้นคลาสเดียวกันกับคลาสแม่ที่อิมพลีเม้นต์เมท็อด จึงสามารถเรียกใช้งานเมท็อดได้จากภายในลำดับชั้นคลาสเดียวกัน ตัวอย่างดังรูปที่ 4.1 แสดงลำดับชั้นคลาสจากคลาสแม่ไปยังคลาสลูก โดยภายในลำดับชั้นคลาสมีคลาสแม่ชื่อ AbstractFigure ที่อิมพลีเม้นต์การทำงานของเมท็อด changed() ส่งผลให้คลาสลูกชื่อ ImageFigure สามารถเรียกใช้งานเมท็อด changed() ได้จากภายในคลาสโดยตรง ดังตัวอย่างคลาส ImageFigure รูปที่ 4.2



รูปที่ 4.1 คลาสแม่ AbstractFigure และคลาสลูก ImageFigure ภายในลำดับชั้นคลาสเดียวกัน

```
public class ImageFigure extends AbstractAttributedDecoratedFigure
        implements ImageHolderFigure {
    public ImageFigure() {
        this(0,0,0,0);
    }
    public ImageFigure(double x, double y, double width, double height) {
        rectangle = new Rectangle2D.Double(x, y, width, height);
    }
    public void setImage(byte[] imageData, BufferedImage bufferedImage) {
        willChange();
        this.imageData = imageData;
        this.bufferedImage = bufferedImage;
        changed();
    }
   ...
}
```

รูปที่ 4.2 ตัวอย่างลักษณะเมท็อดคอลที่เรียกใช้งานเมท็อดจากภายในคลาสโดยตรง

(2) ลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดผ่านชื่อคลาสโดยตรง เนื่องจากชื่อคลาสดังกล่าวเป็นคลาสที่อิมพลีเม้นต์การทำงานของเมท็อดที่ต้องการใช้งาน ตัวอย่างของลักษณะเมท็อดคอลแสดงดังรูปที่ 4.3 ภายในคลาส NodeFigure มีการเรียกใช้งานเมท็อด grow() ของคลาส Geom ด้วยการเรียกใช้งานเมท็อดผ่านชื่อคลาส Geom และภายในคลาส Geom มีการอิมพลีเม้นต์การทำงานของเมท็อด grow() ดังรูปที่ 4.4

```
package org.jhotdraw.samples.net.figures;
import java.awt.geom.*;
public class NodeFigure extends TextFigure {
    public Rectangle2D.Double b;
    public NodeFigure() {    ...    }
    ...
    public Rectangle2D.Double getFigureDrawingArea() {
        b = super.getFigureDrawingArea();
        Geom.grow(b, 10d, 10d);
        return b;
    }
    ...
}
```

รูปที่ 4.3 ตัวอย่างลักษณะเมท็อดคอลที่เรียกใช้งานเมท็อดผ่านชื่อคลาสโดยตรง

```
package org.jhotdraw.geom;
import java.awt.*;
import java.awt.geom.*;
public class Geom {
    private Geom() {}
     ...
    public static void grow(Rectangle2D.Double r, double h, double v) {
        r.x -= h;
        r.y -= v;
        r.width += h * 2d;
        r.height += v * 2d;
    }
}
```

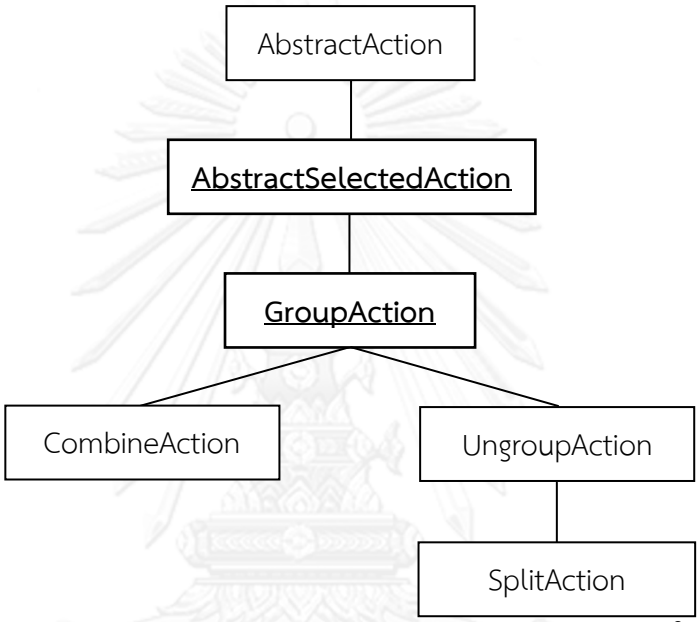
รูปที่ 4.4 คลาส Geom มีการอิมพลีเม้นต์เมท็อด grow()

(3) ลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจ็กต์คลาส ตัวอย่างลักษณะเมท็อดคอลแสดงดังรูปที่ 4.5 ภายในคลาส ConnectionStartHandle มีการเรียกใช้งานเมท็อด changed() โดยเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์ของคลาส ConnectionFigure ชื่อ config ที่ประกาศไว้ภายในคลาส ConnectionStartHandle

```
package org.jhotdraw.draw;
public class ConnectionStartHandle extends AbstractConnectionHandle {
    private ConnectionFigure config;
    public ConnectionStartHandle(ConnectionFigure owner) {
        super(owner);
    }
    ...
    protected void setLocation(Point2D.Double p) {
            config = getOwner();
            config.willChange();
            config.setStartPoint(p);
            config.changed();
    }
    ...
}
```

รูปที่ 4.5 ตัวอย่างลักษณะเมท็อดคอลที่เรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์

(4) ลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์คลาสของคลาสแม่ ลักษณะนี้จะคล้ายกับลักษณะเมท็อดคอลรูปแบบ 3 คือการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์คลาส จากรูปที่ 4.6 แสดงลำดับชั้นคลาส โดยคลาสแม่คือคลาส AbstractSelectedAction และคลาสลูกที่สนใจคือคลาส GroupAction ตัวอย่างลักษณะเมท็อดคอลแสดงดังรูปที่ 4.7 ภายในคลาสลูก GroupAction มีการเรียกใช้งานเมท็อด configureAction() ผ่านตัวแปรอ็อบเจกต์ของคลาส ResourceBundleUtil ชื่อ labels แต่ตัวแปรอ็อบเจกต์ดังกล่าวประกาศไว้ภายในคลาสแม่ AbstractSelectedAction ดังรูป 4.8



รูปที่ 4.6 คลาสแม่ AbstractSelectedAction และคลาสลูก GroupAction ในลำดับชั้นคลาส

```
public class GroupAction extends AbstractSelectedAction {
        ...
  public GroupAction(DrawingEditor editor, CompositeFigure prototype, Boolean isGroupingAction) {
      super(editor);
      this.prototype = prototype;
      this.isGroupingAction = isGroupingAction;
      labels.configureAction(this, ID);
  }
        ...
}
```

รูปที่ 4.7 ตัวอย่างลักษณะเมท็อดคอลที่เรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์ของคลาสแม่

```
public abstract class AbstractSelectedAction extends AbstractAction  {
    private DrawingEditor editor;
    protected ResourceBundleUtil labels =
            ResourceBundleUtil.getLAFBundle("org.jhotdraw.draw.Labels", Locale.getDefault());
    public AbstractSelectedAction(DrawingEditor editor) {
        setEditor(editor);
        updateEnabledState();
    }
}
```

รูปที่ 4.8 การประกาศตัวแปรอ็อบเจกต์ labels ไว้ในคลาสแม่ AbstractSelectedAction

(5) ลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์ที่เป็นพารามิเตอร์ ตัวอย่างลักษณะเมท็อดคอลแสดงดังรูปที่ 4.9 ภายในคลาส AttributeKey มีการเรียกใช้งานเมท็อด changed() โดยเรียกใช้งานผ่านตัวแปรอ็อบเจกต์ซึ่งเป็นพารามิเตอร์ที่รับเข้ามาของเมท็อด ตัวแปรอ็อบเจกต์ดังกล่าว คือ ตัวแปรอ็อบเจกต์ f

```
package org.jhotdraw.draw;
public class  AttributeKey<T> {
    ...
    public AttributeKey(String key, T defaultValue, boolean isNullValueAllowed) {
        this.key = key;
        this.defaultValue = defaultValue;
        this.isNullValueAllowed = isNullValueAllowed;
    }
    public void set(Figure f, T value) {
        f.willChange();
        basicSet(f, value);
        f.changed();
    }
}
```

รูปที่ 4.9 ตัวอย่างลักษณะเมท็อดคอลที่เรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์ที่เป็นพารามิเตอร์

การทำแอสเป็กรีแฟคทอริงโดยจัดการเมท็อดคอลซ้ำกันด้วยแอสเป็ก งานวิจัยนี้เปรียบเทียบจำนวนซ้ำของเมท็อดคอลที่แตกต่างกันทั้งหมด 11 จำนวนดังตารางที่ 4.1 ข้างต้น โดยแต่ละจำนวนซ้ำเมท็อดคอลอาจประกอบด้วยลักษณะเมท็อดคอลหลายลักษณะแตกต่างกัน ผู้วิจัยจึงวิเคราะห์และจำแนกรูปแบบของลักษณะเมท็อดคอลในแต่ละจำนวนซ้ำเมท็อดคอลตาม 5 รูปแบบข้างต้นดังตารางที่ 4.2 โดยรูปแบบของลักษณะเมท็อดคอลระบุด้วยหมายเลขในตารางดังนี้

(1) ลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดจากภายในคลาสโดยตรง
(2) ลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดผ่านชื่อคลาสโดยตรง
(3) ลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจ็กต์คลาส
(4) ลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์คลาสของคลาสแม่
(5) ลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์ที่เป็นพารามิเตอร์

ตารางที่ 4.2 จำนวนรูปแบบของลักษณะเมท็อดคอลในแต่ละจำนวนซ้ำของเมท็อดคอล

จำนวนซ้ำของเมท็อดคอล	**รูปแบบของลักษณะเมท็อดคอล**					**รวม**
	(1)	**(2)**	**(3)**	**(4)**	**(5)**	
4	4	-	-	-	-	4
5	-	-	5	-	-	5
9	-	-	4	-	5	9
12	2	-	5	-	5	12
12	-	-	12	-	-	12
13	13	-		-	-	13
15	-		10	-	5	15
18	-	18	-	-	-	18
21	-	-	-	-	21	21
29	12	-	15	-	2	29
37	17	-	18	-	2	37
44	-	-	36	8	-	44

ดังนั้นเพื่อตอบวัตถุประสงค์ของงานวิจัย ผู้วิจัยจึงต้องดำเนินการทดลองโดยมีขั้นตอนต่างๆ สำหรับเก็บรวบรวมค่ามาตรวัดที่นำมาใช้เปรียบเทียบระหว่างการจัดการเมท็อดคอลในจำนวนซ้ำที่แตกต่างกัน ขั้นตอนการเก็บข้อมูลค่ามาตรวัดมีรายละเอียดดังนี้

4.2.1 การจัดการเมท็อดคอลซ้ำกันทั้งหมดด้วยแอสเป็ก

การทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลซ้ำกันทั้งหมดด้วยแอสเป็ก โดยมีจำนวนซ้ำของเมท็อดคอลแตกต่างกันทั้งหมด 11 จำนวน แต่ละจำนวนซ้ำของเมท็อดคอลเป็นการเรียกใช้งานเมท็อดที่แตกต่างกันทั้งหมด 12 เมท็อดดังตารางที่ 4.1 ข้างต้น การทำแอสเป็กรีแฟคทอริงในแต่ละครั้งกำหนดให้ใช้ซอร์สโค้ดตั้งต้นเดียวกันคือ ซอร์สโค้ดที่ปรับปรุงแล้วของเจฮอตดรอว์เวอร์ชั่น 7.1 และเลือกจัดการกลุ่มของเมท็อดคอลที่เรียกใช้งานเมท็อดเดียวกันจากในลิสผลลัพธ์ตาราง 4.1 ดังนั้นหลังทำแอสเป็กรีแฟคทอริงครบเรียบร้อยจึงได้เจฮอตดรอว์ทั้งหมด 12 เวอร์ชั่นที่จัดการเมท็อดคอลที่มีจำนวนซ้ำแตกต่างกัน สามารถแสดงเวอร์ชั่นของเจฮอตดรอว์ได้ดังตารางที่ 4.3

ตารางที่ 4.3 เวอร์ชั่นของเจฮอตดรอว์ที่จัดการจำนวนซ้ำของเมท็อดคอลแตกต่างกันด้วยแอสเป็ก

เจฮอตดรอว์	**จำนวนซ้ำของเมท็อดคอล**
เวอร์ชั่น 7.1 (ซอฟต์แวร์ตั้งต้น)	0
เวอร์ชั่น 1	4
เวอร์ชั่น 2	5
เวอร์ชั่น 3	9
เวอร์ชั่น 4	12
เวอร์ชั่น 5	12
เวอร์ชั่น 6	13
เวอร์ชั่น 7	15
เวอร์ชั่น 8	18
เวอร์ชั่น 9	21
เวอร์ชั่น 10	29
เวอร์ชั่น 11	37
เวอร์ชั่น 12	44

การทำแอสเป็กรีแฟคทอริงสำหรับจัดการเมท็อดคอลซ้ำกันทั้งหมดด้วยแอสเป็กอ้างอิงจากเทคนิคเอ็กซ์แทรคเมท็อดคอล (Extract method call) ของ Laddad (2003) โดยได้กล่าวเอาไว้ในบทที่ 2 แต่เนื่องจากเทคนิคดังกล่าวเป็นขั้นตอนสำหรับจัดการเมท็อดคอลที่ไม่มีความซับซ้อน คือ สามารถดึงเมท็อดคอลออกมาจากคลาสและนำมาไว้ในแอดไวซ์ของแอสเป็กได้เลยดังตัวอย่างที่แสดงในบทที่ 2 และเป็นเมท็อดคอลที่เรียกใช้งานเมท็อดจากชื่อคลาสโดยตรง แสดงตัวอย่างเมท็อดคอลดังรูปที่ 4.10 เมท็อดคอลเรียกใช้งานเมท็อด checkPermission() จากชื่อคลาส AccessController การจัดการเมท็อดคอลจึงสามารถดึงเมท็อดคอลออกมาจากคลาสและเพิ่มภายในแอดไวซ์ได้โดยตรง

```
AccessController.checkPermission(new BankingPermission("accountOperation"));
```

ชื่อคลาส

รูปที่ 4.10 ตัวอย่างเมท็อดคอลที่เรียกใช้งานเมท็อดจากชื่อคลาสโดยตรง

เนื่องจากลักษณะเมท็อดคอลที่เลือกจัดการด้วยแอสเป็กสำหรับเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นมีลักษณะการเรียกใช้งานเมท็อดหลากหลายดังที่กล่าวข้างต้น โดยอาจไม่สามารถใช้เทคนิคเอ็กซ์แทรคเมท็อดคอลในการจัดการได้เพียงอย่างเดียวแต่ต้องใช้ขั้นตอนอื่นเพิ่มเติม สามารถแสดงลักษณะเมท็อดคอลที่ซับซ้อนรวมถึงวิธีการจัดการเมท็อดคอลด้วยแอสเป็ก แบ่งออกเป็น 3 กรณี ดังนี้

(1) ลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดจากภายในคลาสโดยตรง ดังตัวอย่างเมท็อดคอลในรูปที่ 4.11 ภายในคลาส ImageFigure มีเมท็อดคอลที่เรียกใช้งานเมท็อด changed() ซ้ำกันทั้งหมด 3 ตำแหน่ง

```
package org.jhotdraw.draw;
public class ImageFigure extends AbstractAttributedDecoratedFigure
        implements ImageHolderFigure {
    private byte[] imageData;
    private BufferedImage bufferedImage;
    ...
    public void setImage(byte[] imageData, BufferedImage bufferedImage) {
        willChange();
        this.imageData = imageData;
        this.bufferedImage = bufferedImage;
        changed();
    }
    public void setImageData(byte[] imageData) {
        willChange();
        this.imageData = imageData;
        this.bufferedImage = null;
        changed();
    }
    public void setBufferedImage(BufferedImage image) {
        willChange();
        this.imageData = null;
        this.bufferedImage = image;
        changed();
    }
    ...
}
```

รูปที่ 4.11 ตัวอย่างลักษณะเมท็อดคอลที่เรียกใช้งานเมท็อดจากภายในคลาสโดยตรง

การจัดการเมท็อดคอลทั้ง 3 ตำแหน่งด้วยแอสเป็กสามารถอ้างอิงจากเทคนิคเอ็กซ์แทรคเมท็อดคอล โดยเริ่มจากการสร้างพอยท์คัท changedExecution เพื่อใช้ระบุไปยังตำแหน่งเดิมของเมท็อดคอลเป็นตำแหน่งที่ต้องการให้เมท็อดคอลทำงาน จากตัวอย่างตำแหน่งเดิมของเมท็อดคอลคือตอนท้ายของเมท็อด ดังนั้นจึงกำหนดพอยท์คัทให้ระบุไปยังตำแหน่งจอยน์พอยท์ คือ การทำงานของเมท็อดทั้งสามเมท็อด ด้วยโครงสร้างพอยท์คัทคือ execution(MethodSignature) และสร้างแอดไวซ์ประเภทอาฟเตอร์ (After) แสดงดังรูปที่ 4.12 จากพอยท์คัทและแอดไวซ์ที่สร้างเป็นการกำหนดให้โค้ดภายในแอดไวซ์ของแอสเป็กทำงานหลังจากเมท็อดที่ระบุไว้ในพอยท์คัททำงานเสร็จสิ้น

```
package org.jhotdraw.aspect;
public aspect  AspectChanged {
    pointcut changedExecution ()
          : execution(void ImageFigure.setBufferedImage(..))
          || execution(void ImageFigure.setImage(..))
          || execution(void ImageFigure.setImageData(..));
    after() : changedExecution(){
    }
}
```

รูปที่ 4.12 การสร้างพอยท์คัทและแอดไวซ์

การดึงเมท็อดคอลทั้งสามตำแหน่งจากคลาสออกมาไว้ในแอดไวซ์ของแอสเป็ก เนื่องจากไม่สามารถนำเมท็อดคอลออกมาไว้ภายในแอดไวซ์ได้โดยตรงจึงต้องเพิ่มเติมจากเทคนิคเอ็กซ์แทรคเมท็อดคอลคือ การประกาศตัวแปรอ็อบเจกต์ในแอสเป็กเพื่อเก็บค่าตัวแปรอ็อบเจกต์ที่สามารถเรียกใช้งานเมท็อด changed() และเพิ่มคำสั่งภายในแอดไวซ์เพื่อเก็บค่าตัวแปรอ็อบเจกต์ที่ส่งมาจากพอยท์คัท คำสั่งสำหรับเก็บค่าตัวแปรอ็อบเจกต์ของคลาส ImageFigure ที่ส่งมาจากพอยท์คัท changed Execution คือคำสั่ง Object obj = thisJoinPoint.getThis(); ดังรูปที่ 4.13 ทำให้ในแอดไวซ์สามารถเรียกใช้งานเมท็อด changed() ได้โดยเรียกใช้งานผ่านตัวแปรอ็อบเจกต์ที่ประกาศไว้ภายในแอสเป็ก

```
package org.jhotdraw.aspect;
public aspect  AspectChanged {
    private AbstractFigure tmpObj;
    pointcut changedExecution ()
          : execution(void ImageFigure.setBufferedImage(..))
          || execution(void ImageFigure.setImage(..))
          || execution(void ImageFigure.setImageData(..));
    after() : changedExecution(){
          Object obj = thisJoinPoint.getThis();
          tmpObj = (AbstractFigure)obj;
          tmpObj.changed();
    }
}
```

รูปที่ 4.13 การเก็บค่าตัวแปรอ็อบเจกต์สำหรับนำมาเรียกใช้งานเมท็อด changed()

ขั้นตอนสุดท้ายของเทคนิคเอ็กซ์แทรคเมท็อดคอลคือ การใช้ไวลด์การ์ด (Wildcard) สำหรับปรับปรุงพอยท์คัท ดังรูปที่ 4.14 ผลจากการใช้ไวลด์การ์ดทำให้การแก้ไขหรือเพิ่มฟังก์ชันในอนาคตทำได้ง่ายขึ้น เช่น คลาส SVGImageFigure มีการเรียกใช้งานเมท็อด changed() ในตอนท้ายของทั้งสามเมท็อดที่มีชื่อเมท็อดเหมือนกันกับคลาส ImageFigure การใช้ไวลด์การ์ดส่งผลให้สามารถ

ใช้พอยท์คัทและแอดไวซ์ที่สร้างไว้แล้วกับการเรียกใช้งานเมท็อด changed() ของคลาส SVGImage Figure ได้เลย

```
package org.jhotdraw.aspect;
public aspect  AspectChanged {
    private AbstractFigure tmpObj;
    pointcut changedExecution () : execution(void *ImageFigure.set*Image*(..));
    after() : changedExecution(){
        Object obj = thisJoinPoint.getThis();
        tmpObj = (AbstractFigure)obj;
        tmpObj.changed();
    }
}
```

รูปที่ 4.14 การใช้ไวลด์การ์ดปรับปรุงพอยท์คัท

(2) ลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์ของคลาส หรือเมท็อดคอลมีการส่งอาร์กิวเมนต์เป็นตัวแปรอ็อบเจกต์หรือแอตทริบิวต์ของคลาส ดังตัวอย่างเมท็อดคอลรูปที่ 4.15 ภายในคลาส DrawingPanel มีเมท็อดคอลที่ต้องการจัดการด้วยแอสเป็กคือ เมท็อดคอลที่เรียกใช้งานเมท็อด addUndoableEditListener() ผ่านตัวแปรอ็อบเจกต์ drawing ของคลาส DefaultDrawing และมีการส่งอาร์กิวเมนต์คือตัวแปรอ็อบเจกต์ undoManager ของคลาส UndoRedoManager

```
public class DrawingPanel extends JPanel  {
    private DefaultDrawing drawing;
    private UndoRedoManager undoManager;
    private DrawingEditor editor;
    public DrawingPanel() {
      ResourceBundleUtil labels = ResourceBundleUtil.getLAFBundle(
"org.jhotdraw.draw.Labels");
      initComponents();
      undoManager = new UndoRedoManager();
      editor = new DefaultDrawingEditor();
      editor.add(view);
      drawing = new DefaultDrawing();
      view.setDrawing(drawing);
      drawing.addUndoableEditListener(undoManager);
   }
}
```

รูปที่ 4.15 ตัวอย่างลักษณะเมท็อดคอลที่เรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์และอาร์กิวเมนต์เป็นตัวแปรอ็อบเจกต์

การจัดการเมท็อดคอลด้วยแอสเป็ก ภายในแอสเป็กจึงต้องสามารถเข้าถึงตัวแปรอ็อบเจกต์ที่ต้องนำมาใช้ภายในเมท็อดคอลได้ โดยการประกาศตัวแปรในแอสเป็กและเก็บค่าของตัวแปรอ็อบเจกต์คลาสมาไว้ในตัวแปรที่สร้าง จากตัวอย่างตัวแปรอ็อบเจกต์ของคลาสที่ต้องใช้ในเมท็อดคอลมีสองตัวแปร ได้แก่ drawing และ undoManager ภายในแอสเป็กจึงประกาศสองตัวแปรสำหรับเก็บค่า ได้แก่ ตัวแปร tmpDraw และ tmpUndo จากรูปที่ 4.16 แสดงการเก็บค่าตัวแปรอ็อบเจกต์ drawing ของคลาสมาไว้ในตัวแปรอ็อบเจกต์ tmpDraw ของแอสเป็ก โดยสร้างพอยท์คัท collectDrawing เพื่อระบุไปยังตำแหน่งจอยน์พอยท์ที่เป็นการกำหนดค่าของตัวแปร drawing ในคลาส หรือจอยน์พอยท์ประเภท ฟิลด์ ไรต์ แอกเซส (Field write access) กำหนดด้วยโครงสร้างพอยท์คัทคือ set(FieldSignature) และสร้างแอดไวซ์ประเภทอาฟเตอร์ จากการกำหนดพอยท์คัทและแอดไวซ์ เมื่อตัวแปร drawing ในคลาส DrawingPanel ถูกกำหนดค่าให้กับตัวแปรเรียบร้อยแล้ว โค้ดในแอดไวซ์ของพอยท์คัท collectDrawing จึงทำงานเพื่อรับค่าตัวแปร drawing ของคลาสมากำหนดให้ตัวแปร tmpDraw ของแอสเป็ก

```
package org.jhotdraw.aspect;
public aspect  AspectAddUndoableEditListener {
    private AbstractDrawing tmpDraw;
    pointcut collectDrawing() : set(private * DrawingPanel.drawing);
    after() : collectDrawing(){
        Object tmp[] = thisJoinPoint.getArgs();
        tmpDraw = (AbstractDrawing)tmp[0];
    }
}
```

รูปที่ 4.16 การเก็บค่าตัวแปรอ็อบเจกต์ drawing มาไว้ในแอสเป็ก

ส่วนการเก็บค่าของตัวแปรอ็อบเจกต์ undoManager ของคลาสมาไว้ในตัวแปรอ็อบเจกต์ tmpUndo ของแอสเป็กเพื่อส่งเป็นอาร์กิวเมนต์ของเมท็อดคอล การเก็บค่าจะทำในลักษณะเดียวกันแต่แก้ไขพอยท์คัทให้ระบุไปยังตำแหน่งการกำหนดค่าตัวแปร undoManager ของคลาสแทน ดังนั้นเมื่อซอฟต์แวร์ดำเนินการมาถึงการกำหนดค่าตัวแปร undoManager หลังจากกำหนดค่าตัวแปรเรียบร้อยแล้วโค้ดภายในแอดไวซ์ของพอยท์คัท collectUndoManager จึงทำงานเพื่อรับค่าตัวแปร undoManager ของคลาสมากำหนดให้ตัวแปร tmpUndo ของแอสเป็ก ดังรูปที่ 4.17

```
package org.jhotdraw.aspect;
public aspect  AspectAddUndoableEditListener {
    private UndoRedoManager tmpUndo;
    pointcut collectUndoManager() : set(private * DrawingPanel.undoManager);
    after() : collectUndoManager (){
        Object tmp[] = thisJoinPoint.getArgs();
        tmpUndo = (UndoRedoManager)tmp[0];
    }
}
```

รูปที่ 4.17 การเก็บค่าตัวแปรอ็อบเจกต์ undoManager สำหรับส่งเป็นอาร์กิวเมนต์ในเมท็อดคอล

เมื่อสามารถเก็บค่าทั้งสองตัวแปรอ็อบเจกต์ภายในแอสเป็กแล้ว การนำเมท็อดคอลออกจากคลาสและนำมาไว้ในแอสเป็กจึงสามารถอ้างอิงเทคนิคเอ็กซ์แทรคเมท็อดคอล โดยการสร้างพอยท์คัทเพื่อระบุไปยังตำแหน่งเดิมที่เมท็อดคอลทำงานภายในคลาส จากตัวอย่างเมท็อดคอลอยู่ในตอนท้ายของคอนสตรัคเตอร์ของคลาส DrawingPanel ตำแหน่งดังกล่าวคือจอยน์พอยท์ประเภทคอนสตรัคเตอร์เอ็กซิคิวชั่น (Constructor execution) ต่อมาจึงสร้างแอดไวซ์ประเภทอาฟเตอร์และนำเมท็อดคอลมาไว้ในแอดไวซ์ โดยเรียกใช้งานเมท็อดผ่านตัวแปร tmpDraw และส่งอาร์กิวเมนต์ด้วยตัวแปร tmpUndo ดังรูปที่ 4.18 ดังนั้นเมื่อซอฟต์แวร์ดำเนินการมาถึงการสร้างคอนสตรัคเตอร์ของคลาส DrawingPanel หลังคอนสตรัคเตอร์ทำงานเสร็จเรียบร้อย เมท็อดคอลในแอดไวซ์จึงทำงาน

```
public aspect  AspectAddUndoableEditListener {
    private AbstractDrawing tmpDraw;
    private UndoRedoManager tmpUndo;
    pointcut collectDrawing() : set(private * DrawingPanel.drawing);
    after() : collectDrawing(){
        Object tmp[] = thisJoinPoint.getArgs();
        tmpDraw = (AbstractDrawing)tmp[0];
    }
    pointcut collectUndoManager() : set(private * DrawingPanel.undoManager);
    after() : collectUndoManager (){
        Object tmp[] = thisJoinPoint.getArgs();
        tmpUndo = (UndoRedoManager)tmp[0];
    }
    pointcut addUndoableExecution() : execution(public DrawingPanel.new());
    after () : addUndoableExecution (){
        tmpDraw.addUndoableEditListener(tmpUndo);
    }
}
```

รูปที่ 4.18 การจัดการเมท็อดคอลที่เรียกใช้งานเมท็อด addUndoableEditListener() ด้วยแอสเป็ก

การเรียกใช้งานเมท็อด addUndoableEditListener() มีตำแหน่งซ้ำกันทั้งหมด 15 ตำแหน่งที่สนใจจัดการด้วยแอสเป็ก และเมท็อดคอลทั้งหมดซ้ำกันภายในคลาสทั้งหมด 10 คลาส การจัดการเมท็อดคอลทั้ง 15 ตำแหน่งด้วยแอสเป็ก สามารถแสดงได้ดังรูปที่ 4.19 ด้วยการใช้ไวลด์การ์ด (Wildcard) ทำให้พอยท์คัทที่สร้างสามารถครอบคลุมทั้ง 10 คลาสที่มีเมท็อดคอลซ้ำกันได้

```
package org.jhotdraw.aspect;
public aspect AspectAddUndoableEditListener {
    private AbstractDrawing tmpDraw;
    private UndoRedoManager tmpUndo;
    pointcut collectDrawing() : set(private * *Panel.drawing)
        || execution(void *Panel.setDrawing(..))
        || set(private * *View.draw);
    after() : collectDrawing(){
        Object tmp[] = thisJoinPoint.getArgs();
        tmpDraw = (AbstractDrawing)tmp[0];
    }
    pointcut collectUndoManager() : set(private * *Panel.undoManager)
        || set(private * *View.undo);
    after() : collectUndoManager(){
        Object tmp[] = thisJoinPoint.getArgs();
        tmpUndo = (UndoRedoManager)tmp[0];
    }
    pointcut addUndoableExecution()
          : execution(public *Panel.new()) && !within(*PropertiesPanel)
        ||  execution(void *Panel.setDrawing(..))
        ||  set(private * *View.draw) && withincode(void *View.init());
    after () : addUndoableExecution(){
        tmpDraw.addUndoableEditListener(tmpUndo);
    }
}
```

รูปที่ 4.19 การใช้ไวลด์การ์ดปรับปรุงพอยท์คัท

(3) ลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์ที่เป็นพารามิเตอร์ ดังตัวอย่างเมท็อดคอลในรูปที่ 4.20 ภายในคลาส GroupAction มีเมท็อดคอลที่ต้องการจัดการด้วยแอสเป็กคือ เมท็อดคอลที่เรียกใช้งานเมท็อด addToSelection() ผ่านตัวแปรอ็อบเจกต์ view และมีการส่งอาร์กิวเมนต์ด้วยตัวแปรอ็อบเจกต์ group โดยตัวแปรอ็อบเจกต์ทั้งสองตัวแปรเป็นพารามิเตอร์ที่รับมาจากเมท็อด groupFigures

```
public class GroupAction extends AbstractSelectedAction {
    public LinkedList<Figure> figures;
    public final static String ID = "selectionGroup";
    private CompositeFigure prototype;
    private boolean isGroupingAction;
    ...
    public void groupFigures(DrawingView  view, CompositeFigure group,
Collection<Figure> figures) {
        Collection<Figure> sorted = view.getDrawing().sort(figures);
         view.getDrawing().basicRemoveAll(figures);
         view.clearSelection();
         view.getDrawing().add(group);
        group.willChange();
        for (Figure f : sorted) {
            group.basicAdd(f);
        }
        group.changed();
        view.addToSelection(group);
    }
}
```

รูปที่ 4.20 ตัวอย่างลักษณะเมท็อดคอลที่เรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์ที่เป็นพารามิเตอร์

การจัดการเมท็อดคอลด้วยแอสเป็ก ภายในแอสเป็กจึงต้องสามารถเข้าถึงตัวแปรอ็อบเจกต์ view และ group ด้วยการเก็บค่าของตัวแปรอ็อบเจกต์คลาสมาไว้ในตัวแปรของแอสเป็ก จากตัวอย่างตัวแปรอ็อบเจกต์ของคลาสที่ต้องใช้ในเมท็อดคอลมีสองตัวแปร ได้แก่ view และ group ภายในแอสเป็กจึงประกาศสองตัวแปรได้แก่ tmpView และ tmpCreatedFigure เพื่อเก็บค่าตัวแปรอ็อบเจกต์ของคลาสตามลำดับ จากรูปที่ 4.21 แสดงการเก็บค่าของทั้งสองตัวแปรอ็อบเจกต์ภายในแอสเป็กด้วยการกำหนดพอยท์คัทระบุไปยังการทำงานของเมท็อด groupFigures และสร้างแอดไวซ์ประเภทบีฟอร์ ดังนั้นก่อนการทำงานของเมท็อด groupFigures โค้ดภายในแอดไวซ์ collectArgument จะทำงานก่อนเพื่อเก็บตัวแปรอ็อบเจกต์ที่เป็นพารามิเตอร์ของเมท็อด groupFigures มาไว้ในตัวแปรของแอสเป็ก คำสั่งที่ใช้สำหรับเก็บค่าตัวแปรอ็อบเจกต์คือ Object tmp[] = thisJoinPoint.getArgs(); คำสั่งนี้จะเก็บรวบรวมพารามิเตอร์ทั้งหมดที่รับมาของเมท็อด groupFigures

```
package org.jhotdraw.aspect;
public aspect  AspectAddToSelection {
    private Figure tmpCreatedFigure;
    private DrawingView tmpView;
    pointcut collectArgument() :  execution(void GroupAction.groupFigures(..));
    before() : collectArgument (){
        Object tmp[] = thisJoinPoint.getArgs();
        tmpView = (DrawingView)tmp[0];
        tmpCreatedFigure = (Figure)tmp[1];
    }
}
```

รูปที่ 4.21 การเก็บตัวแปรอ็อบเจกต์ที่เป็นพารามิเตอร์ของเมท็อดภายในแอสเป็ก

เมื่อสามารถเก็บค่าทั้งสองตัวแปรอ็อบเจกต์ไว้ภายในแอสเป็ก การดึงเมท็อดคอลจากคลาสมาไว้ในแอสเป็กจึงสามารถอ้างอิงเทคนิคเอ็กซ์แทรคเมท็อดคอล โดยการสร้างพอยท์คัทเพื่อระบุไปยังตำแหน่งเดิมที่เมท็อดคอลทำงานภายในคลาส จากตัวอย่างเมท็อดคอลอยู่ในตอนท้ายของเมท็อด groupFigures จึงสร้างพอยท์คัทเพื่อระบุไปยังการทำงานของเมท็อด groupFigures ต่อมาสร้างแอดไวซ์ประเภทอาฟเตอร์และนำเมท็อดคอลมาไว้ภายในแอดไวซ์ โดยเรียกใช้งานเมท็อดผ่านตัวแปร tmpView และส่งอาร์กิวเมนต์ด้วยตัวแปร tmpCreatedFigure ดังรูปที่ 4.22 ดังนั้นเมท็อดคอลภายในแอดไวซ์จะทำงานหลังจากการทำงานของเมท็อด groupFigures ของคลาส GroupAction ทำงานเสร็จเรียบร้อย

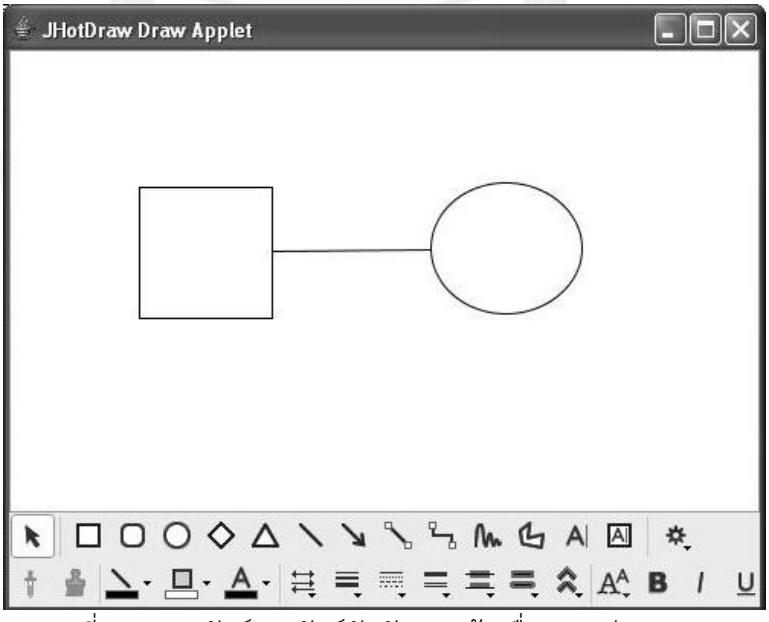
```
public aspect  AspectAddToSelection {
    private Figure tmpCreatedFigure;
    private DrawingView tmpView;
    pointcut collectArgument() :  execution(void GroupAction.groupFigures(..));
    before() : collectArgument(){
        Object tmp[] = thisJoinPoint.getArgs();
        tmpView = (DrawingView)tmp[0];
        tmpCreatedFigure = (Figure)tmp[1];
    }
    pointcut addToSelectionExecution() : execution(void GroupAction.groupFigures(..));
    after() : addToSelectionExecution(){
        tmpView.addToSelection(tmpCreatedFigure);
    }
}
```

รูปที่ 4.22 การจัดการเมท็อดคอลที่เรียกใช้งานเมท็อด addToSelection() ด้วยแอสเป็ก

4.2.2 การตรวจสอบความถูกต้องของเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น

หลังจากการทำแอสเป็กรีแฟคทอริงเพื่อปรับปรุงซอร์สโค้ดของเจฮอตดรอว์ เวอร์ชั่น 7.1 โดยจัดการเมท็อดคอลซ้ำกันด้วยแอสเป็กทำให้ได้เจฮอตดรอว์ครบทั้ง 12 เวอร์ชั่นที่จัดการจำนวนซ้ำของเมท็อดคอลแตกต่างกัน เนื่องจากการรีแฟคทอริงเป็นกระบวนการหรือกลุ่มเทคนิคสำหรับจัดเรียงโค้ดใหม่โดยยังคงพฤติกรรมเดิมของซอฟต์แวร์เอาไว้ หลังจากทำแอสเป็กรีแฟคทอริงผู้วิจัยจึงตรวจสอบพฤติกรรมของเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นว่ายังคงเหมือนกับเจฮอตดรอว์ เวอร์ชั่น 7.1 การทำแอสเป็กรีแฟคทอริงในงานวิจัยนี้เป็นการจัดการเมท็อดคอลด้วยแอสเป็ก ดังนั้นผู้วิจัยจึงทำการดีบัก (Debugging) ในแต่ละจุดที่จัดการนำเมท็อดคอลออกมาไว้ในแอสเป็กเพื่อดูความถูกต้องของลำดับการทำงานระหว่างคลาสและแอสเป็ก ความถูกต้องของการทำงานของเมท็อดคอลภายในแอสเป็ก และพิจารณาผลลัพธ์พฤติกรรมของซอฟต์แวร์ว่ายังคงเดิม การตรวจสอบความผิดพลาดด้วยการดีบักสำหรับเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น ผู้วิจัยได้เลือกใช้ฟังก์ชันการดีบักที่รองรับมาภายในโปรแกรมอีคลิปส์ (Eclipse)

ตัวอย่างที่นำมาแสดงถึงขั้นตอนการตรวจสอบพฤติกรรมของเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นหลังทำแอสเป็กรีแฟคทอริงคือ ตัวอย่างการดีบักของเจฮอตดรอว์ เวอร์ชั่น 10 เป็นเวอร์ชั่นที่จัดการเมท็อดคอลที่เรียกใช้งานเมท็อด changed() ด้วยแอสเป็ก แสดงตัวอย่างการตรวจสอบการทำงานของเมท็อดคอล 1 ตำแหน่งที่จัดการด้วยแอสเป็กคือ เมท็อดคอลที่มีตำแหน่งอยู่ตอนท้ายของเมท็อด updateConnection ในคลาส LineConnectionFigure ก่อนการทำแอสเป็กรีแฟคทอริงได้ทำการรันเจฮอตดรอว์ เวอร์ชั่น 7.1 เพื่อดูผลลัพธ์ของการดำเนินการเมท็อด updateConnection ของคลาส LineConnectionFigure โดยรันไฟล์ DrawApplet และดำเนินการฟังก์ชันอัพเดตเส้นเชื่อมระหว่างรูปทรงด้วยการวางสองรูปทรงและสร้างเส้นเชื่อมระหว่างสองรูปทรง ผลลัพธ์จากการดำเนินการเมท็อด updateConnection ทำให้ได้ผลลัพธ์แสดงดังรูปที่ 4.23 แสดงเส้นเชื่อมที่เชื่อมระหว่างสองรูปทรงได้สำเร็จ



รูปที่ 4.23 ผลลัพธ์ของฟังก์ชันอัพเดตเส้นเชื่อมระหว่างรูปทรง

หลังการทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลที่เรียกใช้งานเมท็อด changed() ในตอนท้ายของเมท็อด updateConnection ในคลาส LineConnectionFigure โดยนำเมท็อดคอลออกมาไว้ภายในแอสเป็ก AspectChanged การตรวจสอบด้วยการดีบักพิจารณา 3 ประเด็นดังนี้

(1) ความถูกต้องของลำดับการทำงานระหว่างคลาสและแอสเป็ก จากตัวอย่างลำดับการทำงานเป็นดังนี้ ลำดับแรกดำเนินการเมท็อด updateConnection เสร็จเรียบร้อย ลำดับสองแอสเป็กจึงเข้ามาขัดขวางการทำงานของคลาส โดยเมท็อดคอลที่เรียกใช้งานเมท็อด changed() ภายในแอดไวซ์ทำงาน การดีบักเพื่อตรวจสอบความถูกต้องของลำดับการทำงานจึงต้องกำหนดจุดพัก (Breakpoint) จากตัวอย่างกำหนดจุดพักทั้งหมดสองตำแหน่งคือ จุดพักแรกอยู่ภายในเมท็อด updateConnection ของคลาส LineConnectionFigure แสดงดังรูปที่ 4.24 จุดนี้กำหนดเพื่อตรวจสอบว่าเข้ามาทำงานภายในเมท็อดของคลาสเป็นลำดับแรก จุดพักสองอยู่ภายในแอดไวซ์ changedExecution ของแอสเป็ก AspectChanged แสดงดังรูปที่ 4.25 จุดนี้กำหนดเพื่อตรวจสอบว่าเมท็อคอลที่เรียกใช้เมท็อด change() ทำงานในลำดับต่อมา จากนั้นจึงรันดีบักด้วยไฟล์ DrawApplet และดำเนินการฟังก์ชันอัพเดตเส้นเชื่อมระหว่างรูปทรง พบว่าลำดับการทำงานระหว่างคลาสและแอสเป็กถูกต้อง

LineConnectionFigure.java | AspectChanged.aj

```
public void updateConnection() {
    willChange();
    if (getStartConnector() != null) {
        Point2D.Double start = getStartConnector().findStart(this);
        if(start != null) {
            setStartPoint(start);
        }
    }
    if (getEndConnector() != null) {
        Point2D.Double end = getEndConnector().findEnd(this);

        if(end != null) {
            setEndPoint(end);
        }
    }
    //changed();
}
```

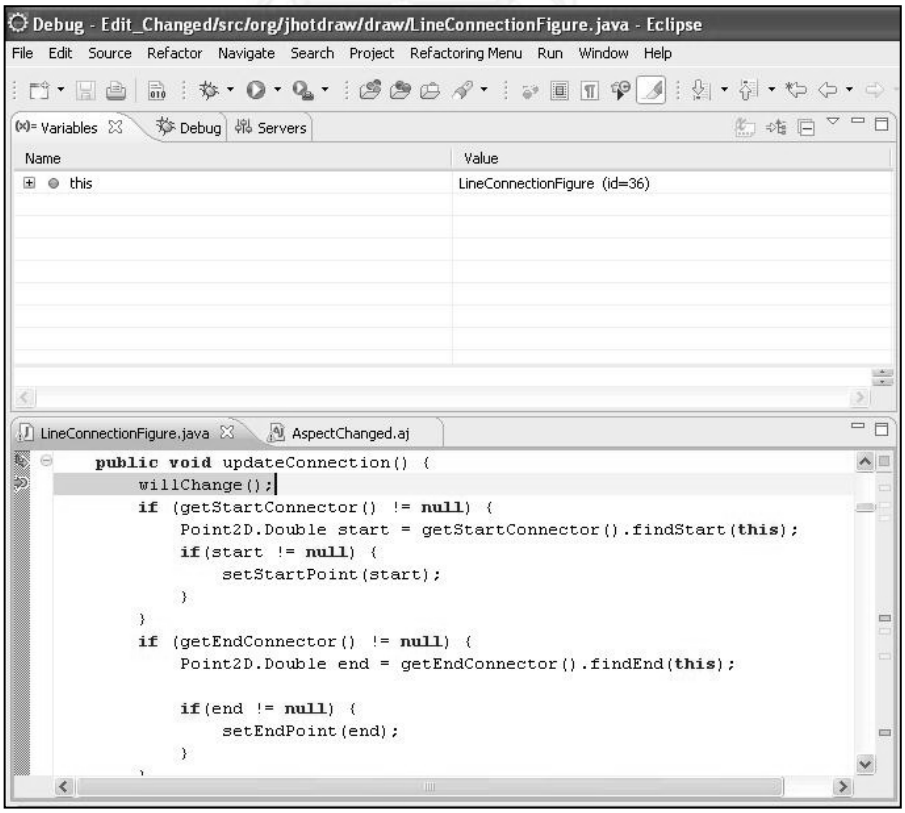
รูปที่ 4.24 การกำหนดจุดพักภายในเมท็อด updateConnection ของคลาส

LineConnectionFigure.java | AspectChanged.aj

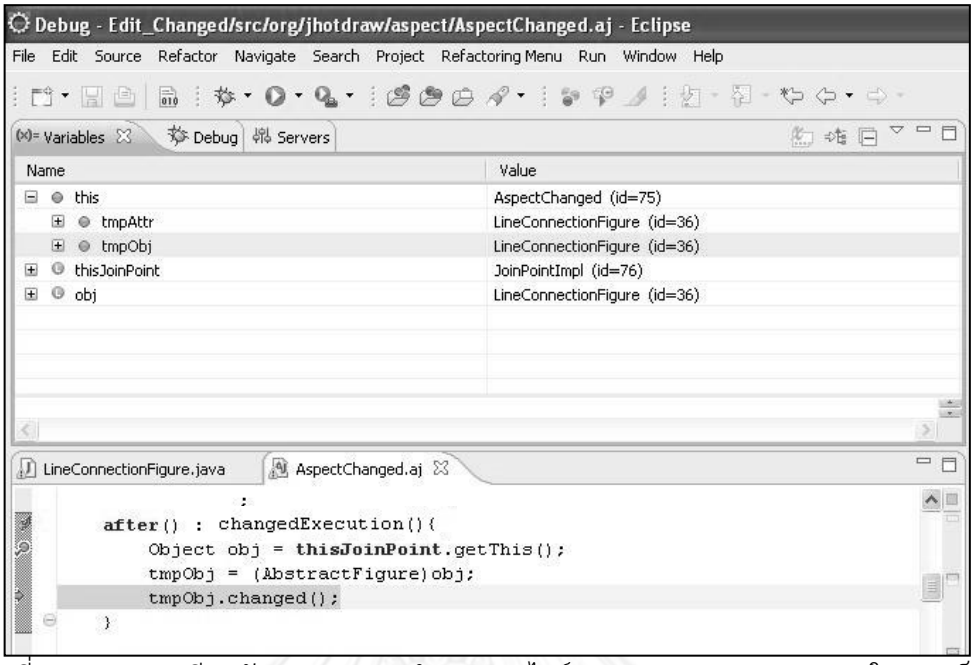
```
privileged public aspect AspectChanged {

    public AbstractFigure tmpObj;
    public Figure tmpAttr;

    pointcut changedExecution(): execution(void LineConnectionFigure.updateConnection())
                                && !within(org.jhotdraw.draw.LabeledLineConnectionFigure);
    after() : changedExecution(){
        Object obj = thisJoinPoint.getThis();
        tmpObj = (AbstractFigure)obj;
        tmpObj.changed();
    }
```

รูปที่ 4.25 การกำหนดจุดพักภายในแอดไวซ์ changedExecution ของแอสเป็ก

(2) ความถูกต้องของตัวแปรอ็อบเจกต์ภายในเมท็อดคอล เนื่องจากตัวอย่างเป็นเมท็อดคอลที่เรียกใช้งานเมท็อดจากในคลาสโดยตรง ก่อนการทำแอสเป็กรีแฟคทอริง เมท็อด changed() ถูกเรียกใช้งานผ่านอ็อบเจกต์ของคลาส LineConnectionFigure ดังนั้นหลังทำแอสเป็กรีแฟคทอริงภายในแอสเป็กควรเรียกใช้เมท็อด changed() ผ่านอ็อบเจกต์คลาสเดียวกัน การตรวจสอบความถูกต้องจึงเปรียบเทียบจากรหัสตัวแปรอ็อบเจกต์ที่รับมาจากพอยท์คัทภายในแอสเป็กว่าตรงกับรหัสอ็อบเจกต์ของคลาสเพื่อยืนยันว่าการเรียกใช้งานเมท็อด changed() เป็นการเรียกใช้งานจากตัวแปรอ็อบเจกต์เดียวกัน ตัวอย่างการดีบักด้วยการรัน DrawApplet และดำเนินการฟังก์ชันอัพเดตเส้นเชื่อมระหว่างรูปทรง เมื่อซอฟต์แวร์รันมาถึงจุดพักที่กำหนด หน้าต่างรายละเอียดการดีบักจะปรากฏและแสดงรายละเอียดตามลำดับการทำงานของคลาสและแอสเป็ก โดยลำดับแรกคือการทำงานของเมท็อด updateConnection ในคลาส LineConnectionFigure รายละเอียดการดีบักแสดงดังรูปที่ 4.26 เมท็อด updateConnection ทำงานภายใต้อ็อบเจกต์คลาส LineConnectionFigure ที่มีรหัสอ็อบเจกต์เท่ากับ 36 และลำดับการทำงานต่อมาหลังจากเมท็อด updateConnection ทำงานเสร็จคือ เมท็อดคอลภายในแอดไวซ์ changedExecution ของแอสเป็กที่ทำงานเพื่อเรียกใช้งานเมท็อด changed() ต้องเรียกใช้งานผ่านอ็อบเจกต์เดียวกัน รายละเอียดการดีบักของลำดับสองแสดงดังรูปที่ 4.27 แสดงตัวแปรอ็อบเจกต์ tmpObj ที่เรียกใช้งานเมท็อด changed() ซึ่งเป็นอ็อบเจกต์ของคลาส LineConnectionFigure ที่มีรหัสอ็อบเจกต์เท่ากับ 36 เช่นกัน จากตัวอย่างเมท็อด changed() จึงถูกเรียกใช้งานผ่านอ็อบเจกต์คลาสเดียวกัน เมท็อดคอลภายในแอสเป็กจึงเป็นเมท็อดคอลเดียวกันเมื่อเทียบกับเมท็อดคอลเดิมภายในคลาสก่อนการทำแอสเป็กรีแฟคทอริง



รูปที่ 4.26 รายละเอียดตัวแปรของการทำงานเมท็อด updateConnection ภายในคลาส



รูปที่ 4.27 รายละเอียดตัวแปรของการทำงานแอดไวซ์ changedExecution ภายในแอสเป็ก

(3) ความถูกต้องของพฤติกรรมซอฟต์แวร์ โดยตรวจสอบพฤติกรรมของซอฟต์แวร์หลังทำแอสเป็กรีแฟคทอริงเทียบกับซอฟต์แวร์ตั้งต้น จากตัวอย่างตรวจสอบผลลัพธ์ที่ได้จากการทำงานฟังก์ชันอัพเดตเส้นเชื่อมระหว่างรูปทรงของเจฮอตดรอว์ เวอร์ชั่น 10 เทียบกับผลลัพธ์ที่ได้ของฟังก์ชันเดียวกันของเจฮอตดรอว์ เวอร์ชั่น 7.1 โดยจากการรันไฟล์ DrawApplet ของเจฮอตดรอว์ เวอร์ชั่น 10 และวางสองรูปทรงพร้อมสร้างเส้นเชื่อมระหว่างสองรูปทรง ผลลัพธ์ที่ได้คือสามารถสร้างเส้นเชื่อมระหว่างสองรูปทรงได้สำเร็จเช่นเดียวกับผลลัพธ์ที่ได้จากการรันของเจฮอตดรอว์ เวอร์ชั่น 7.1 ที่แสดงดังรูปที่ 4.23 ข้างต้น

4.2.3 การเก็บรวบรวมค่ามาตรวัดของซอฟต์แวร์ก่อนและหลังทำแอสเป็กรีแฟคทอริง

เมื่อตรวจสอบความถูกต้องของเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นเรียบร้อยแล้ว จากนั้นนำเจฮอตดรอว์ทุกเวอร์ชั่นมาเก็บค่ามาตรวัดเชิงวัตถุและค่ามาตรวัดเชิงแอสเป็กดังนี้

(1) การเก็บค่ามาตรวัดเชิงวัตถุ

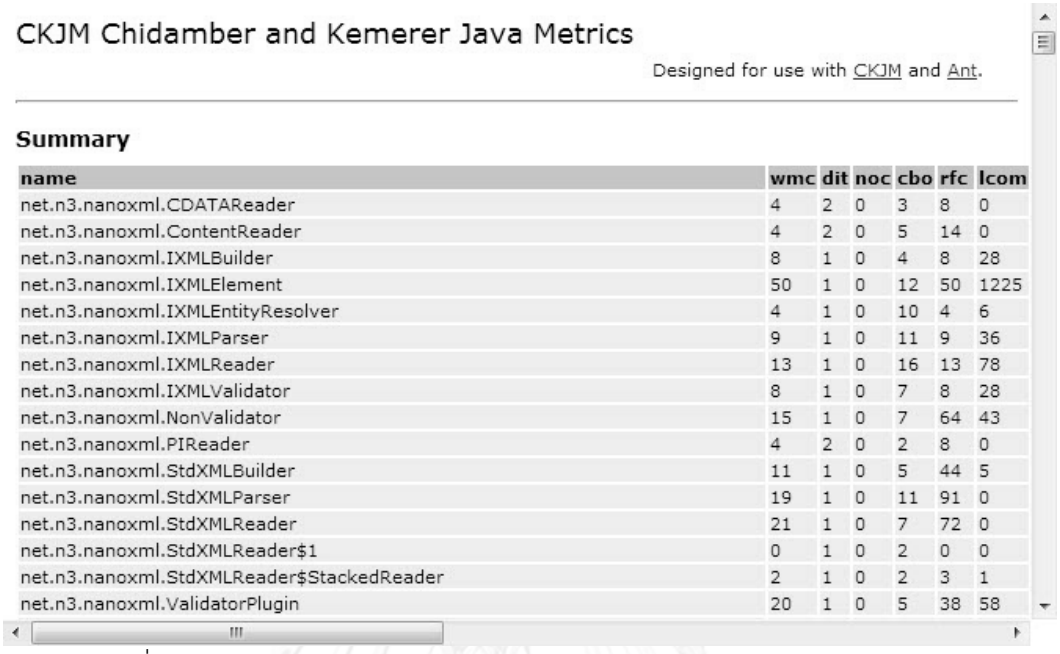
นำเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นและเจฮอตดรอว์ เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้นมาเก็บรวบรวมค่ามาตรวัดเชิงวัตถุของ Chidamber และ Kemerer (1993) ทั้งหมดหกมาตรวัดด้วยเครื่องมือซีเคเจเอ็ม (CKJM) เครื่องมือดังกล่าวใช้งานผ่านบิวด์ทูล (Build Tools) ชื่ออาปาเช่แอนท์ (Apache Ant) ด้วยการระบุชุดคำสั่งของแอนท์ภายในไฟล์ Build.xml ดังนั้นโปรเจคของเจฮอตดรอว์ที่ต้องการเก็บรวบรวมค่ามาตรวัดต้องสร้างไฟล์ Build.xml ไว้ภายในโปรเจค ตัวอย่างชุดคำสั่งภายในไฟล์ Build.xml สำหรับเก็บค่ามาตรวัดเชิงวัตถุของโปรเจคแสดงดังรูปที่ 4.28 ไฟล์ Build.xml จึงสร้างขึ้นมาสำหรับแต่ละโปรเจค เมื่อสร้างไฟล์ Build.xml ภายในโปรเจคเรียบร้อยแล้ว จึงใช้คำสั่งบิวด์โปรเจค (Build Project) เพื่อรันไฟล์ Build.xml

```
<project name="myproject" default="ckjm">
  <property name="src" location="src"/>
  <property name="build" location="build"/>
  <property name="libraries" value="lib"/>
  <path id="classpath.common">
        <fileset dir="${libraries}">
              <include name="*.jar"/>
        </fileset>
  </path>
  <target name="init">
        <mkdir dir="${build}/classes"/>
  </target>
  <target name="compile" depends="init" description="compile the source, build library ">
        <javac srcdir="${src}" destdir="${build}/classes" debug="true">
              <classpath refid="classpath.common"/>
        </javac>
  </target>
  <target name="ckjm" depends="compile" description="compute CK metrics">
        <taskdef name="ckjm" classname="gr.spinellis.ckjm.ant.CkjmTask">
        <classpath>
              <pathelement location="build/ckjm_ext_20.jar"/>
        </classpath>
        </taskdef>
        <ckjm outputfile="build/ckjm.xml" format="xml" classdir="build/classes">
        <include name="**/*.class" />
              <!-- <exclude name="**/*Test.class" /> -->
        </ckjm>
        <xslt in="build/ckjm.xml" style="xsl/ckjm.xsl" out="build/ckjm.html" />
  </target>
</project>
```

รูปที่ 4.28 ตัวอย่างไฟล์ Build.xml สำหรับเก็บรวบรวมค่ามาตรวัดเชิงวัตถุ

ผลลัพธ์จากการรันไฟล์ Build.xml ภายในโปรเจคเจฮอตดรอว์ คือ ไฟล์ HTML ที่แสดงค่ามาตรวัดรายคลาสของโปรเจคเจฮอตดรอว์ เนื่องจากมาตรวัดเชิงวัตถุพิจารณาเพียงไฟล์คลาส โดยภายในโปรเจคเจฮอตดรอว์แต่ละเวอร์ชั่นจะมีคลาสทั้งหมด 442 คลาส ดังนั้นภายในไฟล์ HTML จึงแสดงค่ามาตรวัดทั้งหกมาตรวัดของคลาส 442 คลาส โดยมาตรวัดทั้งหกมาตรวัดของ Chidamber และ Kemerer (1993) ได้แก่ WMC, DIT, NOC, CBO, RFC และ LCOM จากรูปที่ 4.29 แสดงตัวอย่างไฟล์ HTML ที่แสดงให้เห็นค่ามาตรวัดทั้งหกมาตรวัดของคลาสเพียงบางส่วนจากคลาสทั้งหมดภายในโปรเจค เนื่องจากในไฟล์ HTML จะมีคลาสที่มีสัญลักษณ์ ‘$’ ปนอยู่ด้วย ทำให้จำนวนคลาสภายในไฟล์ HTML มีมากกว่า 442 คลาส โดยจากการพิจารณาคลาสที่ไม่มีสัญลักษณ์ปนและตรวจสอบค่าของทั้งหกมาตรวัดของคลาสพบว่า ค่ามาตรวัดที่ได้ตรงกับหลักการนับจากนิยามของ

มาตรวัด ผู้วิจัยจึงคัดลอกค่ามาตรวัดของคลาสทั้งหมดจากในไฟล์ HTML ลงไฟล์ไมโครซอฟท์เอกซ์เซล (Microsoft Excel) เพื่อคัดกรองคลาสที่มีสัญลักษณ์ ‘$’ ออก ให้เหลือคลาสจำนวน 442 คลาส



CKJM Chidamber and Kemerer Java Metrics

Designed for use with CKJM and Ant.

Summary

name	wmc	dit	noc	cbo	rfc	lcom
net.n3.nanoxml.CDATAReader	4	2	0	3	8	0
net.n3.nanoxml.ContentReader	4	2	0	5	14	0
net.n3.nanoxml.IXMLBuilder	8	1	0	4	8	28
net.n3.nanoxml.IXMLElement	50	1	0	12	50	1225
net.n3.nanoxml.IXMLEntityResolver	4	1	0	10	4	6
net.n3.nanoxml.IXMLParser	9	1	0	11	9	36
net.n3.nanoxml.IXMLReader	13	1	0	16	13	78
net.n3.nanoxml.IXMLValidator	8	1	0	7	8	28
net.n3.nanoxml.NonValidator	15	1	0	7	64	43
net.n3.nanoxml.PIReader	4	2	0	2	8	0
net.n3.nanoxml.StdXMLBuilder	11	1	0	5	44	5
net.n3.nanoxml.StdXMLParser	19	1	0	11	91	0
net.n3.nanoxml.StdXMLReader	21	1	0	7	72	0
net.n3.nanoxml.StdXMLReader$1	0	1	0	2	0	0
net.n3.nanoxml.StdXMLReader$StackedReader	2	1	0	2	3	1
net.n3.nanoxml.ValidatorPlugin	20	1	0	5	38	58

รูปที่ 4.29 ไฟล์ HTML แสดงค่ามาตรวัดเชิงวัตถุของแต่ละคลาสภายในโปรเจค

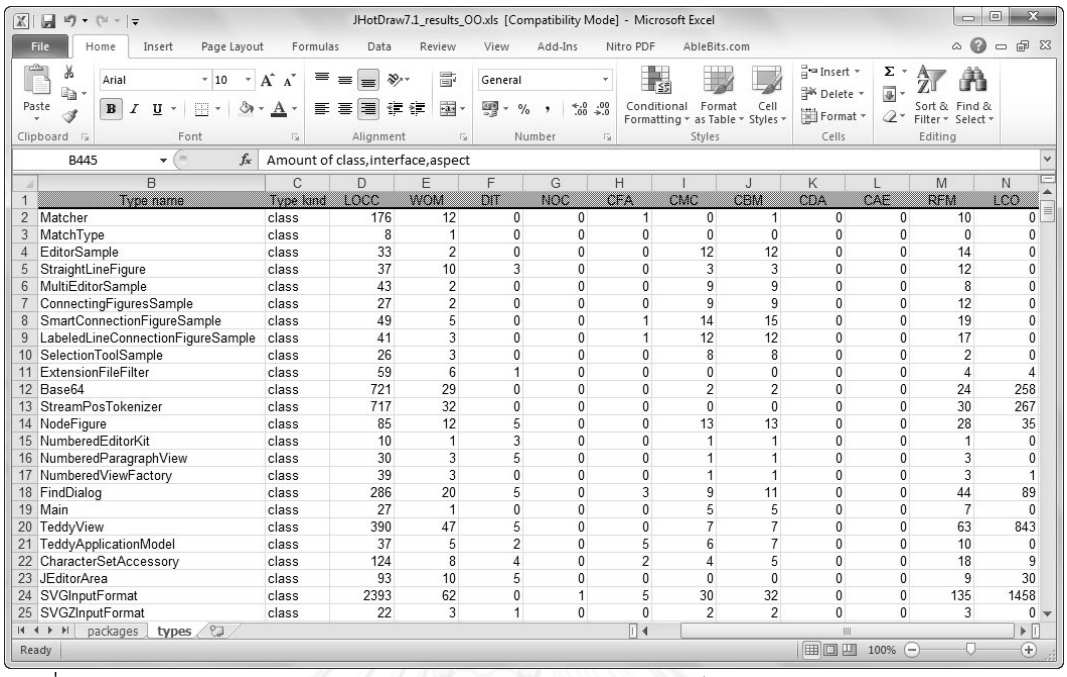
(2) การเก็บค่ามาตรวัดเชิงแอสเป็ก

นำเจฮอตดรอว์ทุกเวอร์ชั่นและเจฮอตดรอว์ 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้นมาเก็บรวบรวมค่ามาตรวัดเชิงแอสเป็กของ Ceccato และ Tonella (2004) ทั้งหมดสิบมาตรวัด ด้วยเครื่องมือเอโอพีเมตริก (AopMetrics) โดยใช้งานผ่านบิวด์ทูลอาปาเช่แอนท์เช่นกัน ดังนั้นจึงต้องสร้างไฟล์ Build.xml ภายในโปรเจคของเจฮอตดรอว์ทุกเวอร์ชั่นเช่นกัน ชุดคำสั่งสำหรับเก็บรวบรวมค่ามาตรวัดเชิงแอสเป็กภายในไฟล์ Build.xml แสดงดังรูป 4.30

```
<project name="aop-metrics" default="run">
    <property name="testworkdir" value="build/testworkdir"/>
    <property name="srcmeasure" value="srcmeasure"/>
    <property name="src" location="src"/>
    <property name="build" location="build"/>
    <property name="libraries" value="lib"/>
    <property name="tools" value="tools"/>
    <property name="src" location="src"/>
    <path id="classpath.common">
        <fileset dir="${libraries}">
            <include name="*.jar"/>
        </fileset>
    </path>
    <target name="run" description="Computes metrics on the project.">
    <taskdef name="aopmetrics" classname="org.tigris.aopmetrics.AopMetricsTask">
        <classpath refid="classpath.common"/>
        <classpath location="build/aop-metrics.jar"/>
    </taskdef>
    <aopmetrics workdir="${testworkdir}" sourcelevel="1.5" export="xls"
         resultsfile="build/aop-metrics-results.xls">
        <fileset dir="${src}" includes="**/*.java"/>
        <fileset dir="${src}" includes="**/*.aj"/>
        <classpath refid="classpath.common"/>
    </aopmetrics>
    </target>
</project>
```

รูปที่ 4.30 ตัวอย่างไฟล์ Build.xml สำหรับเก็บรวบรวมค่ามาตรวัดเชิงแอสเป็ก

ผลลัพธ์จากการรันไฟล์ Build.xml ภายในโปรเจคเจฮอตดรอว์คือ ไฟล์ไมโครซอฟท์เอกซ์เซล (Microsoft Excel) ที่แสดงค่ามาตรวัดรายคลาสและแอสเป็กของโปรเจคเจฮอตดรอว์ เนื่องจากมาตรวัดเชิงแอสเป็กพิจารณาทั้งไฟล์คลาสและแอสเป็ก ดังนั้นจำนวนคลาสและแอสเป็กของเจฮอตดรอว์แต่ละเวอร์ชั่นที่เก็บรวบรวมค่ามาตรวัดมีจำนวนทั้งสิ้น 443 คลาส/แอสเป็ก โดยแยกเป็นคลาส 442 คลาสและแอสเป็ก 1 แอสเป็ก ส่วนเจฮอตดรอว์ เวอร์ชั่น 7.1 มีคลาส 442 คลาสเช่นเดิม โดยมาตรวัดที่แสดงมีทั้งหมดสิบมาตรวัดของ Ceccato และ Tonella (2004) ได้แก่ WOM, DIT, NOC, CDA, CAE, CMC, CFA, CBM, RFM และ LCO จากรูปที่ 4.31 แสดงตัวอย่างไฟล์ไมโครซอฟท์เอกซ์เซลที่แสดงให้เห็นค่ามาตรวัดทั้งสิบมาตรวัดของคลาสเพียงบางส่วนของโปรเจคเจฮอตดรอว์ และเนื่องจากในไฟล์ไมโครซอฟท์เอกซ์เซลมีคลาสที่มีสัญลักษณ์ ‘$’ ปนอยู่ด้วยเช่นกัน ผู้วิจัยจึงคัดกรองคลาสที่มีสัญลักษณ์ ‘$’ ออกเพื่อให้เหลือเพียงคลาสและแอสเป็กที่ถูกต้อง



JHotDraw7.1_results_OO.xls [Compatibility Mode] - Microsoft Excel

B445: Amount of class,interface,aspect

	Type name	Type kind	LOCC	WOM	DIT	NOC	CFA	CMC	CBM	CDA	CAE	RFM	LCO
2	Matcher	class	176	12	0	0	1	0	1	0	0	10	0
3	MatchType	class	8	1	0	0	0	0	0	0	0	0	0
4	EditorSample	class	33	2	0	0	0	12	12	0	0	14	0
5	StraightLineFigure	class	37	10	3	0	0	3	3	0	0	12	0
6	MultiEditorSample	class	43	2	0	0	0	9	9	0	0	8	0
7	ConnectingFiguresSample	class	27	2	0	0	0	9	9	0	0	12	0
8	SmartConnectionFigureSample	class	49	5	0	0	1	14	15	0	0	19	0
9	LabeledLineConnectionFigureSample	class	41	3	0	0	1	12	12	0	0	17	0
10	SelectionToolSample	class	26	3	0	0	0	8	8	0	0	2	0
11	ExtensionFileFilter	class	59	6	1	0	0	0	0	0	0	4	4
12	Base64	class	721	29	0	0	0	2	2	0	0	24	258
13	StreamPosTokenizer	class	717	32	0	0	0	0	0	0	0	30	267
14	NodeFigure	class	85	12	5	0	0	13	13	0	0	28	35
15	NumberedEditorKit	class	10	1	3	0	0	1	1	0	0	1	0
16	NumberedParagraphView	class	30	3	5	0	0	1	1	0	0	3	0
17	NumberedViewFactory	class	39	3	0	0	0	1	1	0	0	3	1
18	FindDialog	class	286	20	5	0	3	9	11	0	0	44	89
19	Main	class	27	1	0	0	0	5	5	0	0	7	0
20	TeddyView	class	390	47	5	0	0	7	7	0	0	63	843
21	TeddyApplicationModel	class	37	5	2	0	5	6	7	0	0	10	0
22	CharacterSetAccessory	class	124	8	4	0	2	4	5	0	0	18	9
23	JEditorArea	class	93	10	5	0	0	0	0	0	0	9	30
24	SVGInputFormat	class	2393	62	0	1	5	30	32	0	0	135	1458
25	SVGZInputFormat	class	22	3	1	0	0	2	2	0	0	3	0

รูปที่ 4.31 ไฟล์ไมโครซอฟท์เอกซ์เซลแสดงค่ามาตรวัดเชิงแอสเป็กของแต่ละคลาสภายในโปรเจค

4.3 ผลการวิเคราะห์ข้อมูลเบื้องต้น

งานวิจัยนี้ต้องการเปรียบเทียบคุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริง โดยจัดการเมท็อดคอลที่มีจำนวนซ้ำของเมท็อดคอลแตกต่างกัน จากการดำเนินตามแผนการทดลองได้เจฮอตดรอว์ทั้งหมด 12 เวอร์ชั่นดังตารางที่ 4.4 โดยมีจำนวนซ้ำของเมท็อดคอลแตกต่างกันทั้งหมด 11 จำนวน และมีสองเวอร์ชั่นที่มีจำนวนซ้ำของเมท็อดคอลเท่ากัน ตัวอย่างข้อมูลในตารางที่ 4.4 เช่น เจฮอตดรอว์ เวอร์ชั่น 1 มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 4 ตำแหน่งด้วยแอสเป็ก เป็นเมท็อดคอลที่เรียกใช้งานเมท็อด fireAreaInvalidated() เป็นต้น ส่วนเจฮอตดรอว์ เวอร์ชั่น 7.1 เป็นซอฟต์แวร์ตั้งต้นที่ไม่ได้ปรับปรุงซอร์สโค้ด การเปรียบเทียบคุณภาพซอฟต์แวร์จึงพิจารณาจากค่ามาตรวัดเชิงวัตถุและค่ามาตรวัดเชิงแอสเป็กของเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น รวมถึงเจฮอตดรอว์ เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้น

ตารางที่ 4.4 เวอร์ชั่นของเจฮอตดรอว์และจำนวนซ้ำของเมท็อดคอลที่จัดการด้วยแอสเป็ก

เจฮอตดรอว์	จำนวนซ้ำเมท็อดคอล	เมท็อด
เวอร์ชั่น 7.1	0	-
เวอร์ชั่น 1	4	fireAreaInvalidated(..)
เวอร์ชั่น 2	5	discardAllEdits()
เวอร์ชั่น 3	9	addToSelection(..)
เวอร์ชั่น 4	12	clearSelection()
เวอร์ชั่น 5	12	removeAllChildren()
เวอร์ชั่น 6	13	firePropertyChange(..)
เวอร์ชั่น 7	15	addUndoableEditListener(..)
เวอร์ชั่น 8	18	grow(..)
เวอร์ชั่น 9	21	createElement(..)
เวอร์ชั่น 10	29	changed()
เวอร์ชั่น 11	37	willChange()
เวอร์ชั่น 12	44	configureAction(..)

4.3.1 ผลการวิเคราะห์ข้อมูลสถิติเชิงพรรณนาจากมาตรวัดเชิงวัตถุ

จากการวัดค่ามาตรวัดเชิงวัตถุของเจฮอตดรอว์ทุกเวอร์ชั่นด้วยเครื่องมือซีเคเจเอ็ม (CKJM) ซึ่งเป็นมาตรวัดเชิงวัตถุของ Chidamber และ Kemerer (1993) มี 6 มาตรวัด ได้แก่ WMC, DIT, NOC, CBO, RFC และ LCOM โดยค่ามาตรวัดทั้งหกมาตรวัดเป็นค่ามาตรวัดรายคลาส คือวัดจากแต่ละคลาสที่มีในเจฮอตดรอว์ เวอร์ชั่น 7.1 และเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น ซึ่งทุกเวอร์ชั่นมีจำนวนคลาสเท่ากันคือ 442 คลาส ผู้วิจัยจึงนำค่ามาตรวัดทั้ง 6 มาตรวัดของ 442 คลาสมาคำนวณเพื่อแสดงข้อมูลสถิติเชิงพรรณนา (Descriptive Statistics) ข้อมูลสถิติเชิงพรรณนาที่พิจารณา ได้แก่

- ✓ ค่าต่ำสุด (Minimum) คือ ค่ามาตรวัดต่ำสุดโดยพิจารณาจากทั้ง 442 คลาสภายในเจฮอตดรอว์
- ✓ ค่าสูงสุด (Maximum) คือ ค่ามาตรวัดสูงสุดโดยพิจารณาจากทั้ง 442 คลาสภายในเจฮอตดรอว์
- ✓ ค่าเฉลี่ย (Mean) คือ ค่าเฉลี่ยของค่ามาตรวัดจากทั้ง 442 คลาส
- ✓ ค่าเบี่ยงเบนมาตรฐาน (Standard Deviation) คือ รากที่สองของค่าแปรปรวน

เบื้องต้นแสดงข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์ เวอร์ชั่น 7.1 ที่ไม่ได้ปรับปรุงด้วยการทำแอสเป็กรีแฟคทอริง และเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นที่ปรับปรุงด้วยการทำแอสเป็กรีแฟคทอริงโดยจัดการจำนวนซ้ำของเมท็อดคอลแตกต่างกัน สามารถแสดงข้อมูลสถิติแยกตามมาตรวัดทั้งหกมาตรวัดได้ดังนี้

(1) เจฮอตดรอว์ เวอร์ชั่น 7.1 เป็นซอฟต์แวร์ตั้งต้น สามารถแสดงข้อมูลสถิติเชิงพรรณนาแยกตามมาตรวัดซีเคทั้ง 6 มาตรวัดดังตารางที่ 4.5

ตารางที่ 4.5 ข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์ เวอร์ชั่น 7.1

มาตรวัดซีเค	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WMC	0	73	12.34	12.145
DIT	0	7	1.05	1.452
NOC	0	21	0.55	2.031
CBO	0	265	14.47	21.980
RFC	0	246	37.54	37.648
LCOM	0	2240	107.21	258.762

(2) เจฮอตดรอว์ เวอร์ชั่น 1 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 4 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดซีเคทั้ง 6 มาตรวัดดังตารางที่ 4.6

ตารางที่ 4.6 ข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์ เวอร์ชั่น 1

มาตรวัดซีเค	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WMC	0	73	12.34	12.145
DIT	0	7	1.05	1.452
NOC	0	21	0.55	2.031
CBO	0	265	14.47	21.980
RFC	0	246	37.54	37.644
LCOM	0	2240	107.21	258.762

(3) เจฮอตดรอว์ เวอร์ชั่น 2 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 5 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดซีเคทั้ง 6 มาตรวัดดังตารางที่ 4.7

ตารางที่ 4.7 ข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์ เวอร์ชั่น 2

มาตรวัดซีเค	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WMC	0	73	12.34	12.145
DIT	0	7	1.05	1.452
NOC	0	21	0.55	2.031
CBO	0	265	14.47	21.980
RFC	0	246	37.53	37.635
LCOM	0	2240	107.21	258.762

(4) เจฮอตดรอว์ เวอร์ชั่น 3 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 9 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดซีเคทั้ง 6 มาตรวัดดังตารางที่ 4.8

ตารางที่ 4.8 ข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์ เวอร์ชั่น 3

มาตรวัดซีเค	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WMC	0	73	12.34	12.145
DIT	0	7	1.05	1.452
NOC	0	21	0.55	2.031
CBO	0	265	14.47	21.980
RFC	0	246	37.52	37.643
LCOM	0	2240	107.21	258.762

(5) เจฮอตดรอว์ เวอร์ชั่น 4 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 12 ตำแหน่งด้วยแอสเป็ก และเนื่องจากจำนวนซ้ำของเมท็อดคอลดังกล่าวมีทั้งหมดสองกรณีที่จัดการในจำนวนซ้ำที่เท่ากัน โดยกรณีนี้เป็นการจัดการเมท็อดคอลที่เรียกใช้งานเมท็อด ClearSelection() สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดซีเคทั้ง 6 มาตรวัดดังตารางที่ 4.9

ตารางที่ 4.9 ข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์ เวอร์ชั่น 4

มาตรวัดซีเค	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WMC	0	73	12.34	12.145
DIT	0	7	1.05	1.452
NOC	0	21	0.55	2.031
CBO	0	265	14.47	21.980
RFC	0	246	37.53	37.637
LCOM	0	2240	107.21	258.762

(6) เจฮอตดรอว์ เวอร์ชั่น 5 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 12 ตำแหน่งด้วยแอสเป็กเช่นกัน โดยกรณีนี้เป็นการจัดการเมท็อดคอลที่เรียกใช้งานเมท็อด RemoveAllChildren() สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดซีเคทั้ง 6 มาตรวัดดังตารางที่ 4.10

ตารางที่ 4.10 ข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์ เวอร์ชั่น 5

มาตรวัดซีเค	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WMC	0	73	12.34	12.145
DIT	0	7	1.05	1.452
NOC	0	21	0.55	2.031
CBO	0	265	14.47	21.980
RFC	0	246	37.52	37.638
LCOM	0	2240	107.21	258.762

(7) เจฮอตดรอว์ เวอร์ชั่น 6 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 13 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดซีเคทั้ง 6 มาตรวัดดังตารางที่ 4.11

ตารางที่ 4.11 ข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์ เวอร์ชั่น 6

มาตรวัดซีเค	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WMC	0	73	12.34	12.145
DIT	0	7	1.05	1.452
NOC	0	21	0.55	2.031
CBO	0	265	14.47	21.980
RFC	0	246	37.53	37.642
LCOM	0	2240	107.21	258.762

(8) เจฮอตดรอว์ เวอร์ชั่น 7 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 15 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดซีเคทั้ง 6 มาตรวัดดังตารางที่ 4.12

ตารางที่ 4.12 ข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์ เวอร์ชั่น 7

มาตรวัดซีเค	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WMC	0	73	12.34	12.145
DIT	0	7	1.05	1.452
NOC	0	21	0.55	2.031
CBO	0	265	14.47	21.980
RFC	0	246	37.51	37.596
LCOM	0	2240	107.21	258.762

(9) เจฮอตดรอว์ เวอร์ชั่น 8 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 18 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดซีเคทั้ง 6 มาตรวัดดังตารางที่ 4.13

ตารางที่ 4.13 ข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์ เวอร์ชั่น 8

มาตรวัดซีเค	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WMC	0	73	12.34	12.145
DIT	0	7	1.05	1.452
NOC	0	21	0.55	2.031
CBO	0	265	14.43	21.950
RFC	0	246	37.51	37.622
LCOM	0	2240	107.21	258.762

(10) เจฮอตดรอว์ เวอร์ชั่น 9 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 21 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดซีเคทั้ง 6 มาตรวัดดังตารางที่ 4.14

ตารางที่ 4.14 ข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์ เวอร์ชั่น 9

มาตรวัดซีเค	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WMC	0	73	12.34	12.145
DIT	0	7	1.05	1.452
NOC	0	21	0.55	2.031
CBO	0	265	14.47	21.980
RFC	0	246	37.54	37.648
LCOM	0	2240	107.21	258.762

(11) เจฮอตดรอว์ เวอร์ชั่น 10 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 29 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดซีเคทั้ง 6 มาตรวัดดังตารางที่ 4.15

ตารางที่ 4.15 ข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์ เวอร์ชั่น 10

มาตรวัดซีเค	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WMC	0	73	12.34	12.145
DIT	0	7	1.05	1.452
NOC	0	21	0.55	2.031
CBO	0	265	14.47	21.980
RFC	0	246	37.51	37.627
LCOM	0	2240	107.21	258.762

(12) เจฮอตดรอว์ เวอร์ชั่น 11 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 37 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดซีเคทั้ง 6 มาตรวัดดังตารางที่ 4.16

ตารางที่ 4.16 ข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์ เวอร์ชั่น 11

มาตรวัดซีเค	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WMC	0	73	12.34	12.145
DIT	0	7	1.05	1.452
NOC	0	21	0.55	2.031
CBO	0	265	14.47	21.980
RFC	0	246	37.49	37.618
LCOM	0	2240	107.21	258.762

(13) เจฮอตดรอว์ เวอร์ชั่น 12 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 44 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดซีเคทั้ง 6 มาตรวัดดังตารางที่ 4.17

ตารางที่ 4.17 ข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์ เวอร์ชั่น 12

มาตรวัดซีเค	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WMC	0	73	12.34	12.145
DIT	0	7	1.05	1.452
NOC	0	21	0.55	2.031
CBO	0	265	14.46	21.947
RFC	0	246	37.44	37.685
LCOM	0	2240	107.22	258.758

4.3.2 ผลการวิเคราะห์ข้อมูลสถิติเชิงพรรณนาจากมาตรวัดเชิงแอสเป็ก

จากการเก็บรวบรวมค่ามาตรวัดเชิงแอสเป็กของเจฮอตดรอว์ทุกเวอร์ชั่นด้วยเครื่องมือเอโอพีเมตริก (AopMetrics) ซึ่งเป็นมาตรวัดเชิงแอสเป็กของ Ceccato และ Tonella (2004) มี 10 มาตรวัดได้แก่ WOM, DIT, NOC, CFA, CMC, CBM, CDA, CAE, RFM และ LCO เนื่องจากค่ามาตรวัดทั้งสิบมาตรวัดเป็นค่ามาตรวัดรายคลาสหรือรายแอสเป็ก คือวัดจากแต่ละคลาสหรือแอสเป็กที่มีในเจฮอตดรอว์ เวอร์ชั่น 7.1 และเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น โดยเจฮอตดรอว์ เวอร์ชั่น 7.1 เป็นซอฟต์แวร์ตั้งต้นที่ไม่ได้ปรับปรุงซอร์สโค้ดทำให้ภายในซอร์สโค้ดมีเพียงคลาส 442 คลาส ส่วนเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น แต่ละเวอร์ชั่นมีคลาส 442 คลาสและแอสเป็ก 1 แอสเป็กสำหรับวัดค่ามาตรวัด ผู้วิจัยจึงนำค่ามาตรวัดทั้ง 10 มาตรวัดของคลาสและแอสเป็กทั้งหมดมาคำนวณเพื่อแสดงข้อมูลสถิติเชิงพรรณนา (Descriptive Statistics) ข้อมูลสถิติเชิงพรรณนาที่พิจารณา ได้แก่

- ✓ ค่าต่ำสุด (Minimum) คือค่ามาตรวัดต่ำสุดโดยพิจารณาจากคลาสและแอสเป็กทั้งหมดภายในเจฮอตดรอว์
- ✓ ค่าสูงสุด (Maximum) คือค่ามาตรวัดสูงสุดโดยพิจารณาจากคลาสและแอสเป็กทั้งหมดภายในเจฮอตดรอว์
- ✓ ค่าเฉลี่ย (Mean) คือค่าเฉลี่ยของค่ามาตรวัดจากจำนวนคลาสและแอสเป็กทั้งหมด
- ✓ ค่าเบี่ยงเบนมาตรฐาน (Standard Deviation) คือรากที่สองของค่าแปรปรวน

เบื้องต้นแสดงข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์ เวอร์ชั่น 7.1 และเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น สามารถแสดงข้อมูลสถิติแยกตามมาตรวัดทั้งสิบมาตรวัดได้ดังนี้

(1) เจฮอตดรอว์ เวอร์ชั่น 7.1 เป็นซอฟต์แวร์ตั้งต้น สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดเชิงแอสเป็กทั้ง 10 มาตรวัดดังตารางที่ 4.18

ตารางที่ 4.18 ข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์ เวอร์ชั่น 7.1

มาตรวัดแอสเป็ก	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WOM	0	90	12.59	12.746
DIT	0	6	1.53	1.589
NOC	0	21	0.55	2.031
CFA	0	23	0.68	2.252
CMC	0	44	4.95	6.951
CBM	0	47	5.26	7.268
CDA	0	0	0.00	0.000
CAE	0	0	0.00	0.000
RFM	0	145	19.26	22.340
LCO	0	3650	87.46	281.127

(2) เจฮอตดรอว์ เวอร์ชั่น 1 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 4 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดเชิงแอสเป็กทั้ง 10 มาตรวัดดังตารางที่ 4.19

ตารางที่ 4.19 ข้อมูลสถิติเชิงพรรณนาแยกตามมาตรวัดเชิงแอสเป็กของเจฮอตดรอว์ เวอร์ชั่น 1

มาตรวัดแอสเป็ก	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WOM	0	90	12.56	12.744
DIT	0	6	1.52	1.589
NOC	0	21	0.55	2.029
CFA	0	23	0.68	2.249
CMC	0	44	4.95	6.944
CBM	0	47	5.25	7.261
CDA	0	2	0.00	0.095
CAE	0	1	0.00	0.067
RFM	0	145	19.22	22.337
LCO	0	3650	87.26	280.840

(3) เจฮอตดรอว์ เวอร์ชั่น 2 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 5 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดเชิงแอสเป็กทั้ง 10 มาตรวัดดังตารางที่ 4.20

ตารางที่ 4.20 ข้อมูลสถิติเชิงพรรณนาแยกตามมาตรวัดเชิงแอสเป็กของเจฮอตดรอว์ เวอร์ชั่น 2

มาตรวัดแอสเป็ก	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WOM	0	90	12.56	12.742
DIT	0	6	1.52	1.589
NOC	0	21	0.55	2.029
CFA	0	23	0.68	2.249
CMC	0	44	4.95	6.944
CBM	0	47	5.25	7.261
CDA	0	5	0.01	0.238
CAE	0	1	0.01	0.106
RFM	0	145	19.22	22.333
LCO	0	3650	87.26	280.840

(4) เจฮอตดรอว์ เวอร์ชั่น 3 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 9 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดเชิงแอสเป็กทั้ง 10 มาตรวัดดังตารางที่ 4.21

ตารางที่ 4.21 ข้อมูลสถิติเชิงพรรณนาแยกตามมาตรวัดเชิงแอสเป็กของเจฮอตดรอว์ เวอร์ชั่น 3

มาตรวัดแอสเป็ก	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WOM	0	90	12.57	12.737
DIT	0	6	1.52	1.589
NOC	0	21	0.55	2.029
CFA	0	23	0.68	2.249
CMC	0	44	4.94	6.945
CBM	0	47	5.24	7.261
CDA	0	9	0.02	0.428
CAE	0	1	0.02	0.141
RFM	0	145	19.25	22.351
LCO	0	3650	87.26	280.840

(5) เจฮอตดรอว์ เวอร์ชั่น 4 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 12 ตำแหน่งด้วยแอสเป็ก เนื่องจากจำนวนซ้ำของเมท็อดคอลดังกล่าวมีทั้งหมดสองกรณีที่จัดการในจำนวนซ้ำที่เท่ากัน กรณีนี้จัดการเมท็อดคอลที่เรียกใช้งานเมท็อด ClearSelection() สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดเชิงแอสเป็กทั้ง 10 มาตรวัดดังตารางที่ 4.22

ตารางที่ 4.22 ข้อมูลสถิติเชิงพรรณนาแยกตามมาตรวัดเชิงแอสเป็กของเจฮอตดรอว์ เวอร์ชั่น 4

มาตรวัดแอสเป็ก	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WOM	0	90	12.57	12.737
DIT	0	6	1.52	1.589
NOC	0	21	0.55	2.029
CFA	0	23	0.68	2.249
CMC	0	44	4.95	6.942
CBM	0	47	5.25	7.259
CDA	0	11	0.02	0.523
CAE	0	1	0.02	0.156
RFM	0	146	19.26	22.381
LCO	0	3650	87.26	280.840

(6) เจฮอตดรอว์ เวอร์ชั่น 5 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 12 ตำแหน่งด้วยแอสเป็ก กรณีนี้เป็นการจัดการเมท็อดคอลที่เรียกใช้งานเมท็อด RemoveAllChildren() สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดเชิงแอสเป็กทั้ง 10 มาตรวัดดังตารางที่ 4.23

ตารางที่ 4.23 ข้อมูลสถิติเชิงพรรณนาแยกตามมาตรวัดเชิงแอสเป็กของเจฮอตดรอว์ เวอร์ชั่น 5

มาตรวัดแอสเป็ก	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WOM	0	90	12.56	12.744
DIT	0	6	1.52	1.589
NOC	0	21	0.55	2.029
CFA	0	23	0.68	2.249
CMC	0	44	4.95	6.944
CBM	0	47	5.25	7.261
CDA	0	9	0.02	0.428
CAE	0	1	0.02	0.141
RFM	0	145	19.22	22.341
LCO	0	3650	87.26	280.840

(7) เจฮอตดรอว์ เวอร์ชั่น 6 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 13 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดเชิงแอสเป็กทั้ง 10 มาตรวัดดังตารางที่ 4.24

ตารางที่ 4.24 ข้อมูลสถิติเชิงพรรณนาแยกตามมาตรวัดเชิงแอสเป็กของเจฮอตดรอว์ เวอร์ชั่น 6

มาตรวัดแอสเป็ก	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WOM	0	90	12.56	12.740
DIT	0	6	1.52	1.589
NOC	0	21	0.55	2.029
CFA	0	23	0.68	2.249
CMC	0	44	4.95	6.943
CBM	0	47	5.25	7.260
CDA	0	6	0.01	0.285
CAE	0	1	0.01	0.116
RFM	0	145	19.25	22.354
LCO	0	3650	87.26	280.840

(8) เจฮอตดรอว์ เวอร์ชั่น 7 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 15 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดเชิงแอสเป็กทั้ง 10 มาตรวัดดังตารางที่ 4.25

ตารางที่ 4.25 ข้อมูลสถิติเชิงพรรณนาแยกตามมาตรวัดเชิงแอสเป็กของเจฮอตดรอว์ เวอร์ชั่น 7

มาตรวัดแอสเป็ก	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WOM	0	90	12.56	12.740
DIT	0	6	1.52	1.589
NOC	0	21	0.55	2.029
CFA	0	23	0.68	2.249
CMC	0	44	4.95	6.944
CBM	0	47	5.25	7.261
CDA	0	10	0.02	0.475
CAE	0	1	0.02	0.149
RFM	0	145	19.27	22.397
LCO	0	3650	87.26	280.840

(9) เจฮอตดรอว์ เวอร์ชั่น 8 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 18 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดเชิงแอสเป็กทั้ง 10 มาตรวัดดังตารางที่ 4.26

ตารางที่ 4.26 ข้อมูลสถิติเชิงพรรณนาแยกตามมาตรวัดเชิงแอสเป็กของเจฮอตดรอว์ เวอร์ชั่น 8

มาตรวัดแอสเป็ก	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WOM	0	90	12.57	12.737
DIT	0	6	1.52	1.589
NOC	0	21	0.55	2.029
CFA	0	23	0.68	2.249
CMC	0	44	4.93	6.924
CBM	0	47	5.23	7.241
CDA	0	12	0.03	0.570
CAE	0	1	0.03	0.163
RFM	0	145	19.30	22.423
LCO	0	3650	87.26	280.840

(10) เจฮอตดรอว์ เวอร์ชั่น 9 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 21 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดเชิงแอสเป็กทั้ง 10 มาตรวัดดังตารางที่ 4.27

ตารางที่ 4.27 ข้อมูลสถิติเชิงพรรณนาแยกตามมาตรวัดเชิงแอสเป็กของเจฮอตดรอว์ เวอร์ชั่น 9

มาตรวัดแอสเป็ก	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WOM	0	90	12.56	12.744
DIT	0	6	1.52	1.589
NOC	0	21	0.55	2.029
CFA	0	23	0.68	2.250
CMC	0	44	4.95	6.943
CBM	0	47	5.26	7.259
CDA	0	2	0.00	0.095
CAE	0	1	0.00	0.067
RFM	0	145	19.23	22.348
LCO	0	3650	87.26	280.840

(11) เจฮอตดรอว์ เวอร์ชั่น 10 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 29 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดเชิงแอสเป็กทั้ง 10 มาตรวัดดังตารางที่ 4.28

ตารางที่ 4.28 ข้อมูลสถิติเชิงพรรณนาแยกตามมาตรวัดเชิงแอสเป็กของเจฮอตดรอว์ เวอร์ชั่น 10

มาตรวัดแอสเป็ก	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WOM	0	90	12.57	12.738
DIT	0	6	1.52	1.589
NOC	0	21	0.55	2.029
CFA	0	23	0.68	2.249
CMC	0	44	4.95	6.944
CBM	0	47	5.25	7.260
CDA	0	23	0.05	1.093
CAE	0	1	0.05	0.222
RFM	0	145	19.26	22.389
LCO	0	3650	87.26	280.840

(12) เจฮอตดรอว์ เวอร์ชั่น 11 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 37 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดเชิงแอสเป็กทั้ง 10 มาตรวัดดังตารางที่ 4.29

ตารางที่ 4.29 ข้อมูลสถิติเชิงพรรณนาแยกตามมาตรวัดเชิงแอสเป็กของเจฮอตดรอว์ เวอร์ชั่น 11

มาตรวัดแอสเป็ก	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WOM	0	90	12.57	12.737
DIT	0	6	1.52	1.589
NOC	0	21	0.55	2.029
CFA	0	23	0.68	2.249
CMC	0	44	4.95	6.944
CBM	0	47	5.25	7.260
CDA	0	26	0.06	1.235
CAE	0	1	0.06	0.235
RFM	0	145	19.27	22.392
LCO	0	3650	87.26	280.840

(13) เจฮอตดรอว์ เวอร์ชั่น 12 ภายในซอร์สโค้ดของซอฟต์แวร์มีการจัดการเมท็อดคอลซ้ำกันทั้งหมด 44 ตำแหน่งด้วยแอสเป็ก สามารถแสดงข้อมูลสถิติเชิงพรรณาแยกตามมาตรวัดเชิงแอสเป็กทั้ง 10 มาตรวัดดังตารางที่ 4.30

ตารางที่ 4.30 ข้อมูลสถิติเชิงพรรณนาแยกตามมาตรวัดเชิงแอสเป็กของเจฮอตดรอว์ เวอร์ชั่น 12

มาตรวัดแอสเป็ก	ค่าต่ำสุด	ค่าสูงสุด	ค่าเฉลี่ย	ค่าเบี่ยงเบนมาตรฐาน
WOM	0	90	12.56	12.742
DIT	0	6	1.52	1.589
NOC	0	21	0.55	2.029
CFA	0	23	0.68	2.249
CMC	0	44	4.95	6.945
CBM	0	47	5.25	7.261
CDA	0	44	0.10	2.091
CAE	0	1	0.10	0.299
RFM	0	145	19.22	22.333
LCO	0	3650	87.26	280.840

4.4 การเปรียบเทียบค่ามาตรวัดเชิงวัตถุ

ผลการทดลองทำให้ได้เจฮอตดรอว์ทั้งหมด 12 เวอร์ชั่น ที่จัดการจำนวนซ้ำของเมท็อดคอลแตกต่างกันด้วยแอสเป็ก จำนวนซ้ำที่พิจารณาในงานวิจัยได้แก่ 4, 5, 9, 12, 13, 15, 18, 21, 29, 37 และ 44 โดยการวัดค่ามาตรวัดเชิงวัตถุของเจฮอตดรอว์จะไม่พิจารณาไฟล์แอสเป็ก ดังนั้นค่ามาตรวัดซีเคทั้ง 6 มาตรวัดจึงวัดแต่ไฟล์คลาส ซึ่งเจฮอตดรอว์แต่ละเวอร์ชั่นมีคลาสทั้งหมด 442 คลาส เบื้องต้นผู้วิจัยเปรียบเทียบค่ามาตรวัดรายคลาสของเจฮอตดรอว์แต่ละเวอร์ชั่นกับเจฮอตดรอว์ 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้นที่ไม่ได้ปรับปรุงซอร์สโค้ด เพื่อพิจารณาค่ามาตรวัดของคลาสก่อนและหลังทำแอสเป็กรีแฟคทอริง จากการพิจารณาคลาสทั้งหมดของเจฮอตดรอว์แต่ละเวอร์ชั่น สามารถแสดงมาตรวัดที่มีค่ามาตรวัดของคลาสเปลี่ยนแปลงหลังทำแอสเป็กรีแฟคทอริงได้ดังตารางที่ 4.31

ตารางที่ 4.31 มาตรวัดซีเคซึ่งมีคลาสที่ค่ามาตรวัดเปลี่ยนแปลงหลังทำแอสเป็กรีแฟคทอริง

เจฮอตดรอว์	**จำนวนซ้ำเมท็อดคอล**	**มาตรวัดซีเค**					
		WMC	DIT	NOC	CBO	RFC	LCOM
เวอร์ชั่น 1	4	-	-	-	-	✓	-
เวอร์ชั่น 2	5	-	-	-	-	✓	-
เวอร์ชั่น 3	9	-	-	-	-	✓	-
เวอร์ชั่น 4	12	-	-	-	-	✓	-
เวอร์ชั่น 5	12	-	-	-	-	✓	-
เวอร์ชั่น 6	13	-	-	-	-	✓	-
เวอร์ชั่น 7	15	-	-	-	-	✓	-
เวอร์ชั่น 8	18	-	-	-	✓	✓	-
เวอร์ชั่น 9	21	-	-	-	-	-	-
เวอร์ชั่น 10	29	-	-	-	-	✓	-
เวอร์ชั่น 11	37	-	-	-	-	✓	-
เวอร์ชั่น 12	44	-	-	-	✓	✓	✓

หมายเหตุ

เครื่องหมาย - แสดงถึง ไม่มีคลาสที่ค่ามาตรวัดเปลี่ยนแปลงหลังทำแอสเป็กรีแฟคทอริง

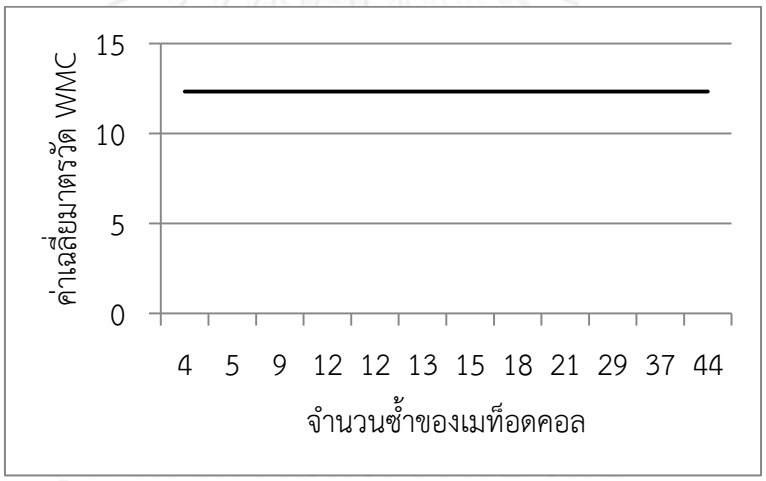
เครื่องหมาย ✓ แสดงถึง มีคลาสที่ค่ามาตรวัดเปลี่ยนแปลงหลังทำแอสเป็กรีแฟคทอริง

การพิจารณาข้อมูลค่ามาตรวัดของแต่ละมาตรวัดเชิงวัตถุสามารถนำมาพิจารณาตามกรอบการวิเคราะห์ข้อมูลทั้งหมดสามส่วน ได้แก่ (1) การเปรียบเทียบค่ามาตรวัดก่อนและหลังทำแอสเป็กรีแฟคทอริง (2) การเปรียบเทียบค่าเฉลี่ยมาตรวัดระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น และ (3) การวิเคราะห์ผลมาตรวัดต่อคุณภาพซอฟต์แวร์ รายละเอียดการวิเคราะห์ข้อมูลของแต่ละมาตรวัดมีดังนี้

(1) เวทเทดเมท็อดเปอคลาส (Weighted Method per Class - WMC)

จากตารางที่ 4.31 เป็นการเปรียบเทียบเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นกับเจฮอตดรอว์ เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้น โดยเปรียบเทียบค่ามาตรวัด WMC รายคลาส พบว่าเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นไม่มีคลาสที่มีค่ามาตรวัด WMC เปลี่ยนแปลงหลังทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลซ้ำกันด้วยแอสเป็ก

ผลการเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นจึงให้ผลเช่นเดียวกัน โดยนำค่าเฉลี่ยของมาตรวัด WMC จากข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์แต่ละเวอร์ชั่น (ตารางที่ 4.6 - 4.17) มาเปรียบเทียบ แสดงด้วยกราฟดังรูปที่ 4.32 พบว่าจำนวนซ้ำของเมท็อดคอลไม่ส่งผลให้ค่ามาตรวัด WMC เปลี่ยนแปลงไม่ว่าจะจัดการเมท็อดคอลในจำนวนซ้ำที่น้อยหรือมากด้วยแอสเป็ก โดยค่าเฉลี่ยของมาตรวัด WMC มีค่าคงเดิมเท่ากับ 12.34



รูปที่ 4.32 ค่าเฉลี่ยของมาตรวัด WMC ของเจฮอตดรอว์แต่ละเวอร์ชั่นที่จัดการจำนวนซ้ำเมท็อดคอลแตกต่างกัน

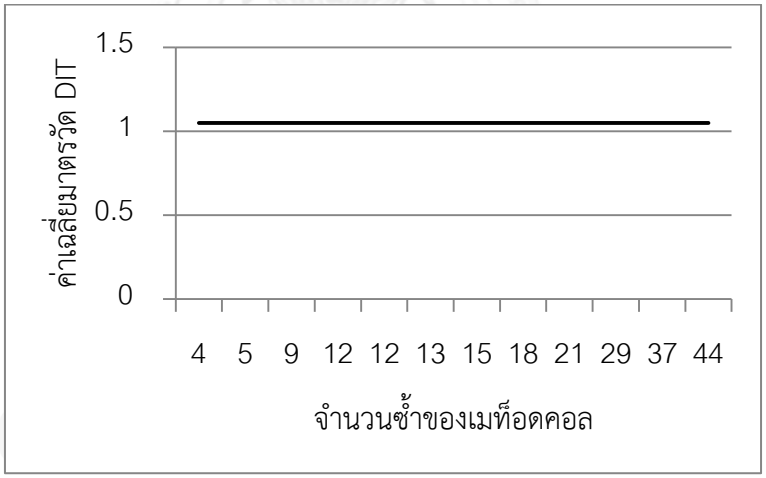
ค่ามาตรวัด WMC ไม่เปลี่ยนแปลงหลังทำแอสเป็กรีแฟคทอริง เนื่องจากมาตรวัด WMC พิจารณาจากจำนวนเมท็อดที่อิมพลีเม้นต์ภายในคลาส ดังนั้นค่ามาตรวัด WMC ของคลาสจะมีค่าเพิ่มขึ้นหรือลดลงจึงต้องมีการเปลี่ยนแปลงจำนวนเมท็อดที่อิมพลีเม้นต์ภายในคลาส ได้แก่ การเพิ่มเมท็อดหรือนำเมท็อดออกจากคลาส แต่เนื่องจากงานวิจัยนี้เลือกทำแอสเป็กรีแฟคทอริงโดยสนใจจัดการเพียงเมท็อดคอลที่ซ้ำกัน ทำให้มีแค่เมท็อดคอลซึ่งเป็นเพียงคำสั่งภายในคลาสถูกนำออกจากคลาสจึงไม่ส่งผลต่อจำนวนเมท็อดที่อิมพลีเม้นต์ภายในคลาส

จากตารางที่ 2.3 ความสัมพันธ์ระหว่างมาตรวัดเชิงวัตถุกับคุณภาพซอฟต์แวร์ ที่อ้างอิงจากงานวิจัยของ Kulkarni, Kalshetty และ Arde (2010) โดยมาตรวัด WMC สามารถส่งผลกระทบต่อคุณภาพซอฟต์แวร์ 2 ด้าน ได้แก่ ความสามารถการบำรุงรักษาและความสามารถการนำกลับมาใช้ใหม่ แต่จากผลการเปรียบเทียบค่ามาตรวัด WMC แสดงให้เห็นว่าการทำแอสเป็กรีแฟคทอริงไม่ทำให้ค่ามาตรวัด WMC ของคลาสเปลี่ยนแปลง ดังนั้นการทำแอสเป็กรีแฟคทอริงจึงไม่ส่งผลต่อคุณภาพซอฟต์แวร์ทั้งสองด้านด้วยเช่นกัน

(2) เดพออฟอินเฮอริแทนทรี (Depth of Inheritance Tree - DIT)

จากตารางที่ 4.31 เป็นการเปรียบเทียบเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นกับเจฮอตดรอว์ เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้น โดยเปรียบเทียบค่ามาตรวัด DIT รายคลาส พบว่าเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นไม่มีคลาสที่มีค่ามาตรวัด DIT เปลี่ยนแปลงหลังทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลซ้ำกันด้วยแอสเป็ก

ผลการเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นจึงให้ผลเช่นเดียวกัน โดยนำค่าเฉลี่ยของมาตรวัด DIT จากข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์แต่ละเวอร์ชั่น (ตารางที่ 4.6 – 4.17) มาเปรียบเทียบ แสดงด้วยกราฟดังรูปที่ 4.33 พบว่าจำนวนซ้ำของเมท็อดคอลไม่ส่งผลให้ค่ามาตรวัด DIT เกิดการเปลี่ยนแปลงไม่ว่าจะจัดการเมท็อดคอลในจำนวนซ้ำที่น้อยหรือมากด้วยแอสเป็ก โดยค่าเฉลี่ยของมาตรวัด DIT มีค่าคงเดิมเท่ากับ 1.05



รูปที่ 4.33 ค่าเฉลี่ยของมาตรวัด DIT ของเจฮอตดรอว์แต่ละเวอร์ชั่นที่จัดการจำนวนซ้ำเมท็อดคอลแตกต่างกัน

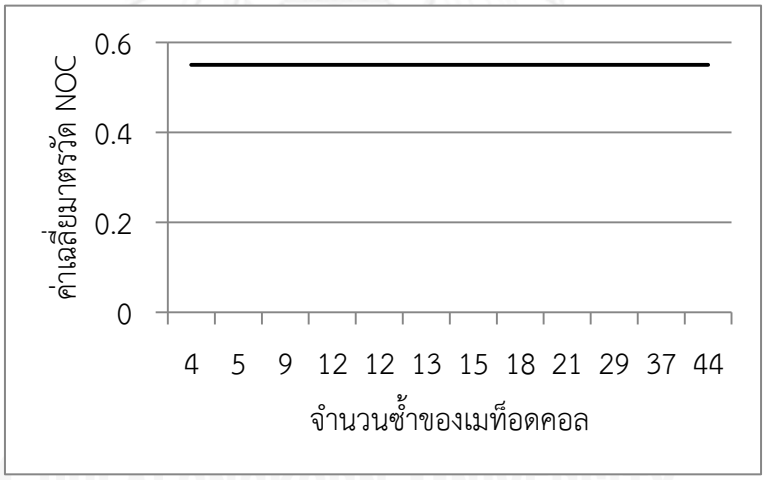
ค่ามาตรวัด DIT ไม่เปลี่ยนแปลงหลังทำแอสเป็กรีแฟคทอริง เนื่องจากมาตรวัด DIT พิจารณาจากระดับความลึกของคลาสภายในลำดับชั้นการรับทอด (Inheritance hierarchy) โดยนับจากคลาสโหนดไปยังรูทโหนดของลำดับชั้น ดังนั้นค่ามาตรวัด DIT ของคลาสจะมีค่าเพิ่มขึ้นหรือลดลงจึงต้องมีการเปลี่ยนแปลงภายในลำดับชั้นของคลาส เช่น การเพิ่มหรือลดคลาสภายในลำดับชั้นการรับทอดที่ส่งผลให้จำนวนชั้นในลำดับชั้นการรับทอดเปลี่ยนแปลง แต่เนื่องจากงานวิจัยนี้เลือกทำแอสเป็กรีแฟคทอริงโดยสนใจการจัดการเพียงเมท็อดคอลที่ซ้ำกันจึงทำให้ไม่ส่งผลต่อลำดับชั้นการรับทอด

จากตารางที่ 2.3 ความสัมพันธ์ระหว่างมาตรวัดเชิงวัตถุกับคุณภาพซอฟต์แวร์ ที่อ้างอิงจากงานวิจัยของ Kulkarni, Kalshetty และ Arde (2010) โดยมาตรวัด DIT สามารถส่งผลกระทบต่อคุณภาพซอฟต์แวร์ 2 ด้าน ได้แก่ ความสามารถในการทำความเข้าใจและความสามารถการนำกลับมาใช้ใหม่ แต่จากผลการเปรียบเทียบค่ามาตรวัด DIT แสดงให้เห็นว่าการทำแอสเป็กรีแฟคทอริงไม่ทำให้ค่ามาตรวัด DIT ของคลาสเปลี่ยนแปลง ดังนั้นการทำแอสเป็กรีแฟคทอริงจึงไม่ส่งผลต่อคุณภาพซอฟต์แวร์ทั้งสองด้าน

(3) นัมเบอร์ออฟชิลเดรน (Number of Children - NOC)

จากตารางที่ 4.31 เปรียบเทียบเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นกับเจฮอตดรอว์ เวอรชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้น โดยเปรียบเทียบค่ามาตรวัด NOC รายคลาส พบว่าเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นไม่มีคลาสที่มีค่ามาตรวัด NOC เปลี่ยนแปลงหลังทำแอสเป็กรีแฟคทอริง

ผลการเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นจึงให้ผลเช่นเดียวกัน โดยนำค่าเฉลี่ยของมาตรวัด NOC จากข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์แต่ละเวอร์ชั่น (ตารางที่ 4.6 – 4.17) มาเปรียบเทียบ แสดงด้วยกราฟดังรูปที่ 4.34 พบว่าจำนวนซ้ำของเมท็อดคอลไม่ส่งผลให้ค่ามาตรวัด NOC เปลี่ยนแปลงไม่ว่าจะจัดการเมท็อดคอลในจำนวนซ้ำที่น้อยหรือมากด้วยแอสเป็ก โดยค่าเฉลี่ยของมาตรวัด NOC มีค่าคงเดิมเท่ากับ 0.55



รูปที่ 4.34 ค่าเฉลี่ยของมาตรวัด NOC ของเจฮอตดรอว์แต่ละเวอร์ชั่นที่จัดการจำนวนซ้ำเมท็อดคอลแตกต่างกัน

ค่ามาตรวัด NOC ไม่เปลี่ยนแปลงหลังจากทำแอสเป็กรีแฟคทอริง เนื่องจากมาตรวัด NOC พิจารณาจากจำนวนคลาสลูกภายในลำดับชั้นการรับทอด ดังนั้นค่ามาตรวัด NOC จะเพิ่มขึ้นหรือลดลงจึงต้องมีการเปลี่ยนแปลงภายในลำดับชั้นของคลาส ได้แก่ การเพิ่มหรือลดคลาสลูกภายในลำดับชั้นการรับทอด แต่เนื่องจากในงานวิจัยนี้เลือกทำแอสเป็กรีแฟคทอริงโดยสนใจจัดการเพียงเมท็อดคอลที่ซ้ำกันจึงทำให้ไม่ส่งผลต่อลำดับชั้นการรับทอด

จากตารางที่ 2.3 ความสัมพันธ์ระหว่างมาตรวัดเชิงวัตถุกับคุณภาพซอฟต์แวร์ ที่อ้างอิงจากงานวิจัยของ Kulkarni, Kalshetty และ Arde (2010) โดยมาตรวัด NOC สามารถส่งผลกระทบต่อคุณภาพซอฟต์แวร์ 2 ด้าน ได้แก่ ความสามารถการนำกลับมาใช้ใหม่และความสามารถการทดสอบ แต่จากผลการเปรียบเทียบค่ามาตรวัด NOC แสดงให้เห็นว่าการทำแอสเป็กรีแฟคทอริงไม่ทำให้ค่ามาตรวัด NOC ของคลาสเปลี่ยนแปลง ดังนั้นการทำแอสเป็กรีแฟคทอริงจึงไม่ส่งผลต่อคุณภาพซอฟต์แวร์ทั้งสองด้านด้วยเช่นกัน

(4) คัพพลิงบีทวีนอ็อบเจกต์คลาส (Coupling Between Object classes - CBO)

จากตารางที่ 4.31 เป็นการเปรียบเทียบเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นกับเจฮอตดรอว์ เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้น โดยเปรียบเทียบค่ามาตรวัด CBO รายคลาส พบว่ามีเพียงเจฮอตดรอว์ เวอร์ชั่น 8 และ 12 ที่จัดการเมท็อดคอลซ้ำกันทั้งหมด 18 และ 44 ตำแหน่งตามลำดับ มีคลาสที่มีค่ามาตรวัด CBO เปลี่ยนแปลงหลังทำแอสเป็กรีแฟคทอริง สามารถแสดงรายชื่อคลาสที่มีค่ามาตรวัด CBO เปลี่ยนแปลง และค่ามาตรวัด CBO ก่อน-หลังทำแอสเป็กรีแฟคทอริงของเจฮอตดรอว์ เวอร์ชั่นที่ 8 และ 12 ได้ดังนี้

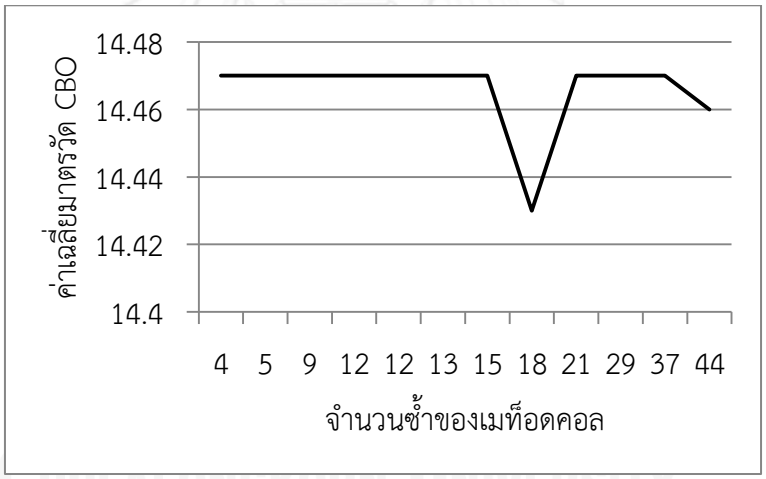
ตารางที่ 4.32 รายชื่อคลาส 9 คลาสของเจฮอตดรอว์ เวอร์ชั่น 8 ที่ค่ามาตรวัด CBO เปลี่ยนแปลง

คลาส	**มาตรวัด CBO**	
	ก่อนทำแอสเป็กรีแฟคทอริง	**หลังทำแอสเป็กรีแฟคทอริง**
org.jhotdraw.draw.EllipseFigure	12	11
org.jhotdraw.draw.ImageFigure	21	20
org.jhotdraw.draw.RectangleFigure	13	12
org.jhotdraw.draw.RoundRectangleFigure	14	13
org.jhotdraw.geom.Geom	44	36
org.jhotdraw.samples.odg.figures.ODGPathFigure	38	37
org.jhotdraw.samples.svg.figures.SVGPathFigure	42	41
org.jhotdraw.samples.svg.figures.SVGTextAreaFigure	27	26
org.jhotdraw.samples.svg.figures.SVGTextFigure	38	37

ตารางที่ 4.33 รายชื่อคลาส 3 คลาสของเจฮอตดรอว์ เวอร์ชั่น 12 ที่ค่ามาตรวัด CBO เปลี่ยนแปลง

คลาส	มาตรวัด CBO	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.draw.action.SelectSameAction	14	13
org.jhotdraw.draw.action.UngroupAction	15	14
org.jhotdraw.util.ResourceBundleUtil	174	172

ผลการเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น โดยนำค่าเฉลี่ยของมาตรวัด CBO จากข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์แต่ละเวอร์ชั่น (ตารางที่ 4.6 – 4.17) มาเปรียบเทียบแสดงด้วยกราฟดังรูปที่ 4.35 พบว่าหลังทำแอสเป็กรีแฟคทอริงการจัดการเมท็อดคอลที่ซ้ำกัน 18 และ 44 ตำแหน่งส่งผลให้ค่ามาตรวัด CBO มีค่าลดลง โดยการจัดการเมท็อดคอลที่ซ้ำกันทั้งหมด 18 ตำแหน่งให้ค่ามาตรวัด CBO ลดลงมากสุดเมื่อเทียบกับจำนวนซ้ำเมท็อดคอลอื่นๆ โดยมีค่าเฉลี่ยของมาตรวัด CBO เท่ากับ 14.43 ส่วนการจัดการเมท็อดคอลที่ซ้ำกัน 44 ตำแหน่งมีค่ามาตรวัด CBO ลดลงเพียงเล็กน้อยคือ ค่าเฉลี่ยของมาตรวัด CBO เท่ากับ 14.46



รูปที่ 4.35 ค่าเฉลี่ยของมาตรวัด CBO ของเจฮอตดรอว์แต่ละเวอร์ชั่นที่จัดการจำนวนซ้ำเมท็อดคอลแตกต่างกัน

มาตรวัด CBO พิจารณาจากจำนวนความสัมพันธ์ระหว่างคลาสอื่นๆกับคลาสที่พิจารณา โดยคลาสสองคลาสจะสัมพันธ์กันเมื่อภายในคลาสที่พิจารณาเรียกใช้งานเมท็อดของคลาสอื่นหรือประกาศตัวแปรอินสแตนซ์ของคลาสอื่น ดังนั้นค่ามาตรวัด CBO ของคลาสจะมีค่าเพิ่มขึ้นหรือลดลงจึงต้องมีการเปลี่ยนแปลงจำนวนคลาสที่สามารถเรียกใช้งานเมท็อด หรือเปลี่ยนแปลงจำนวนตัวแปรอินสแตนซ์ที่ประกาศภายในคลาส

จากผลค่ามาตรวัด CBO ของเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น มีทั้งหมด 10 เวอร์ชั่นที่ค่ามาตรวัด CBO คงเดิมหลังทำแอสเป็กรีแฟคทอริง เนื่องจากการจัดการเมท็อดคอลด้วยแอสเป็กไม่สามารถลดความสัมพันธ์ระหว่างคลาส มีทั้งหมด 3 กรณี ดังนี้

- ภายในคลาสมีเมท็อดคอลซ้ำกันหลายตำแหน่ง แต่ไม่สามารถจัดการเมท็อดคอลทั้งหมดด้วยแอสเป็กได้ เนื่องจากตำแหน่งเมท็อดคอลบางตำแหน่งไม่สามารถระบุด้วยพอยท์คัทที่มีในภาษาแอสเป็กเจ ทำให้ไม่สามารถลดความสัมพันธ์ระหว่างคลาสเพราะภายในคลาสยังคงเรียกใช้งานเมท็อดอยู่ในตำแหน่งอื่น แสดงดังตัวอย่างรูปที่ 4.36 ภายในคลาส AttributeKey เรียกใช้งานเมท็อด changed() ของแอ็บสแตรคคลาส AbstractFigure ทั้งหมด 5 ตำแหน่ง ทำให้คลาส AttributeKey มีความสัมพันธ์กับแอ็บสแตรคคลาส AbstractFigure หลังทำแอสเป็กรีแฟคทอริงสามารถจัดการเมท็อดคอลที่ซ้ำด้วยแอสเป็กได้เพียง 2 ตำแหน่ง แต่เมท็อดคอลอีก 3 ตำแหน่งไม่สามารถจัดการด้วยแอสเป็กได้ เนื่องจากตำแหน่งเมท็อดคอลอยู่ภายในเมท็อดที่ซ้อนเมท็อดของคลาสอีกทีจึงไม่สามารถกำหนดด้วยพอยท์คัทของภาษาแอสเป็กเจ ทำให้ภายในคลาสยังคงเรียกใช้งานเมท็อด changed() จากตำแหน่งอื่น ส่งผลให้คลาส AttributeKey ยังคงสัมพันธ์กับแอ็บสแตรคคลาส AbstractFigure ค่ามาตรวัด CBO ของคลาส AttributeKey จึงมีค่าไม่ลดลง

```
public class AttributeKey<T> {
    public void set(Figure f, T value) {
        f.willChange();
        basicSet(f, value);
        //f.changed();
    }
    public UndoableEdit setUndoable(final Figure figure, final T value, final
ResourceBundleUtil labels) {
        figure.willChange();
        figure.setAttribute(this, value);
        figure.changed();
        UndoableEdit edit = new AbstractUndoableEdit() {
            public void undo() {
                super.undo();
                figure.willChange();
                figure.restoreAttributesTo(restoreData);
                figure.changed();
            }
            public void redo() {
                super.redo();
                figure.willChange();
                figure.setAttribute(AttributeKey.this, value);
                figure.changed();
            }
        };
        return edit;
    }
    public void setClone(Figure f, T value) {
        f.willChange();
        basicSetClone(f, value);
        //f.changed();
    }
}
```

รูปที่ 4.36 ซอร์สโค้ดของคลาส AttributeKey ที่มีการเรียกใช้งานเมท็อด changed()

- ภายในคลาสมีเมท็อดคอลซ้ำกันหลายตำแหน่งและสามารถจัดการเมท็อดคอลที่ซ้ำกันทั้งหมดด้วยแอสเป็กได้ แต่ยังไม่สามารถลดความสัมพันธ์ระหว่างคลาสได้ หากในคลาสยังเรียกใช้งานเมท็อดอื่นที่อิมพลีเม้นต์จากในคลาสเดียวกัน ดังรูปที่ 4.37 แอ็บสแตรคคลาส AbstractFigure มีการอิมพลีเม้นต์เมท็อด willChange() และ changed() สำหรับให้คลาสอื่นเรียกใช้งาน จากตัวอย่างการจัดการเมท็อดคอลรูปที่ 4.38 ภายในคลาส ODGPathFigure เรียกใช้งานเมท็อด changed() และสามารถจัดการเมท็อดคอลทั้งหมดด้วยแอสเป็กได้ แต่เนื่องจากในคลาสยังคงเรียกใช้งานเมท็อด willChange() ทำให้ไม่สามารถลดความสัมพันธ์ระหว่างคลาส AbstractFigure และ ODGPath Figure ส่งผลให้หลังทำแอสเป็กรีแฟคทอริงค่ามาตรวัด CBO ของคลาส ODGPathFigure จึงมีค่าคงเดิมไม่ลดลง

```
public abstract class AbstractFigure extends AbstractBean implements Figure {
    protected int changingDepth = 0;
    public AbstractFigure() {}
    public void willChange() {
        if (changingDepth == 0) {
            fireAreaInvalidated();
            invalidate();
        }
        changingDepth++;
    }
    public void changed() {
        if (changingDepth == 1) {
            validate();
            fireFigureChanged(getDrawingArea());
        } else if (changingDepth < 0) {
            throw new InternalError("changed was called without a prior call to willChange.");
        }
        changingDepth--;
    }
}
```

รูปที่ 4.37 แอ็บสแตรคคลาส AbstractFigure ที่อิมพลีเม้นต์เมท็อด willChange() และ changed()

```
package org.jhotdraw.samples.odg.figures;
public class ODGPathFigure extends AbstractAttributedCompositeFigure implements
ODGFigure {
    public ODGPathFigure() {
        add(new ODGBezierFigure());
        ODGAttributeKeys.setDefaults(this);
    }
   ...
    public void flattenTransform() {
        willChange();
        AffineTransform tx = TRANSFORM.get(this);
        if (tx != null) {
            for (Figure child : getChildren()) {
                ((ODGBezierFigure) child).transform(tx);
                ((ODGBezierFigure) child).flattenTransform();
            }
        }
        TRANSFORM.basicSet(this, null);
        //changed();
    }
}
```

รูปที่ 4.38 ซอร์สโค้ดบางส่วนของคลาส ODGPathFigure

- ภายในคลาสมีเมท็อดคอลซ้ำกันหลายตำแหน่งและสามารถจัดการเมท็อดคอลที่ซ้ำกันทั้งหมดด้วยแอสเป็กได้ แต่ยังไม่สามารถลดความสัมพันธ์ระหว่างคลาสได้ หากในคลาสยังคงประกาศตัวแปรอินสแตนซ์ของคลาสเดียวกันไว้ ดังตัวอย่างรูปที่ 4.39 และ 4.40 ภายในคลาส Duplicate Action เรียกใช้งานเมท็อด configureAction() ของคลาส ResourceBundleUtil และสามารถจัดการเมท็อดคอลด้วยแอสเป็กได้ แต่ภายในคลาสยังมีการประกาศตัวแปรอินสแตนซ์ของคลาส ResourceBundleUtil ชื่อ labels ส่งผลให้ไม่สามารถลดความสัมพันธ์ระหว่างคลาส Duplicate Action และ ResourceBundleUtil ค่ามาตรวัด CBO ของคลาส DuplicateAction จึงมีค่าคงเดิมไม่ลดลง

```
package org.jhotdraw.app.action;
public class DuplicateAction extends AbstractAction {
    public final static String ID = "duplicate";
    private ResourceBundleUtil labels;
    public DuplicateAction() {
        labels = ResourceBundleUtil.getLAFBundle("org.jhotdraw.app.Labels");
        //labels.configureAction(this, ID);
    }
}
```

รูปที่ 4.39 คลาส DuplicateAction

```
public class ResourceBundleUtil {
    private ResourceBundle resource;
    private Class baseClass = getClass();
    public ResourceBundleUtil(ResourceBundle r) {
        resource = r;
    }
    public void configureAction(Action action, String argument) {
        configureAction(action, argument, getBaseClass());
    }
    public void configureAction(Action action, String argument, Class baseClass) {
        action.putValue(Action.NAME, getString(argument));
        action.putValue(Action.ACCELERATOR_KEY, getAcc(argument));
        action.putValue(Action.MNEMONIC_KEY, new Integer(getMnem(argument)));
        action.putValue(Action.SMALL_ICON, getImageIcon(argument, baseClass));
    }
}
```

รูปที่ 4.40 คลาส ResourceBundleUtil

ดังนั้นจากผลค่ามาตรวัด CBO จึงมีเพียงเจฮอตดรอว์ เวอร์ชั่น 8 และ 12 ที่ค่ามาตรวัด CBO ของบางคลาสมีค่าลดลงหลังทำแอสเป็กรีแฟคทอริง โดยเจฮอตดรอว์ เวอร์ชั่น 8 จัดการเมท็อดคอลซ้ำกัน 18 ตำแหน่ง มีค่ามาตรวัด CBO ลดลงมากกว่าเจฮอตดรอว์เวอร์ชั่นอื่นๆ เนื่องจากมีคลาสถึง 9 คลาสที่ค่ามาตรวัด CBO ลดลง จากการพิจารณาลักษณะเมท็อดคอลที่จัดการด้วยแอสเป็กของเจฮอตดรอว์ เวอร์ชั่น 8 ในตาราง 4.2 ข้างต้น พบว่าลักษณะเมท็อดคอลทั้งหมดเป็นการเรียกใช้งานเมท็อดผ่านชื่อคลาสโดยตรง คือการเรียกใช้งานเมท็อด grow() ของคลาส Geom ด้วยชื่อคลาสโดยตรง เช่น Geom.grow(b,10d,10d); เป็นต้น ส่งผลให้เมื่อจัดการเมท็อดคอลซ้ำกันทั้งหมดด้วยแอสเป็กทำให้สามารถลดความสัมพันธ์ระหว่างคลาสได้ ดังนั้นจากตารางที่ 4.32 ข้างต้น คลาส 8 คลาสที่เรียกใช้งานเมท็อด grow() ของคลาส Geom เมื่อจัดการเมท็อดคอลด้วยแอสเป็ก แต่ละคลาสจึงมีค่ามาตรวัด CBO ลดลงเท่ากับ 1 เพราะแต่ละคลาสสามารถลดความสัมพันธ์กับคลาสอื่นได้เพียง 1 คลาส คือ คลาส Geom ส่วนค่ามาตรวัดของคลาส Geom เนื่องจากสามารถลดความสัมพันธ์กับคลาสอื่นๆที่มา

เรียกใช้งานเมท็อด grow() จากในคลาสได้ถึง 8 คลาส ค่ามาตรวัด CBO ของคลาส Geom จึงมีค่าลดลงมากเท่ากับ 8

ส่วนเจฮอตดรอว์ เวอร์ชั่น 12 จัดการเมท็อดคอลซ้ำกันทั้งหมด 44 ตำแหน่ง มีค่ามาตรวัด CBO ลดลงเล็กน้อย โดยทั้ง 44 ตำแหน่งเป็นการเรียกใช้งานเมท็อด configureAction() ของคลาส ResourceBundleUtil มีเมท็อดคอลเพียง 8 ตำแหน่งที่มีลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์ของคลาสแม่ แต่มีเพียงเมท็อดคอล 2 ตำแหน่งที่จัดการด้วยแอสเป็กแล้วส่งผลให้ค่ามาตรวัดของคลาส 2 คลาสมีค่ามาตรวัดลดลง เนื่องจากตัวแปรอ็อบเจกต์สำหรับเรียกใช้งานเมท็อดเป็นตัวแปรของคลาส ResourceBundleUtil ที่ถูกประกาศไว้ในคลาสแม่ และภายในคลาสไม่มีการเข้าถึงตัวแปรดังกล่าวในตำแหน่งอื่น หลังทำแอสเป็กรีแฟคทอริงคลาสทั้ง 2 คลาสจึงสามารถลดความสัมพันธ์กับคลาส ResourceBundleUtil ขณะที่เมท็อดคอลอีก 6 ตำแหน่งที่มีลักษณะเมท็อดคอลเหมือนกันแต่ค่ามาตรวัด CBO กลับไม่ลดลง เนื่องจากภายในคลาสยังคงเข้าถึงตัวแปรอ็อบเจกต์ของคลาส ResourceBundleUtil ที่ประกาศไว้ภายในคลาสแม่ในตำแหน่งอื่นของคลาส จากตารางที่ 4.33 ข้างต้นจึงมีเพียงแค่ 2 คลาส ได้แก่ คลาส SelectSameAction และ UngroupAction ที่มีค่ามาตรวัด CBO ลดลงเท่ากับ 1 ส่วนคลาส ResourceBundleUtil สามารถลดความสัมพันธ์กับคลาสอื่นที่มาเรียกใช้งานเมท็อดได้ทั้งหมด 2 คลาส ค่ามาตรวัด CBO ของคลาส ResourceBundleUtil จึงมีลดลงเท่ากับ 2

จากตารางที่ 2.3 ความสัมพันธ์ระหว่างมาตรวัดเชิงวัตถุกับคุณภาพซอฟต์แวร์ที่อ้างอิงจากงานวิจัยของ Kulkarni, Kalshetty และ Arde (2010) โดยมาตรวัด CBO สามารถส่งผลกระทบต่อคุณภาพซอฟต์แวร์ 4 ด้าน ได้แก่ ความสามารถในการทำความเข้าใจ ความสามารถการบำรุงรักษา ความสามารถการนำกลับมาใช้ใหม่ และความสามารถการทดสอบ จากผลการเปรียบเทียบค่ามาตรวัด CBO แสดงให้เห็นว่าการทำแอสเป็กรีแฟคทอริงส่งผลให้ค่ามาตรวัด CBO ของคลาสลดลง หากจัดการเมท็อดคอลที่มีลักษณะการเรียกใช้งานเมท็อดผ่านชื่อคลาสโดยตรง หรือการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์ของคลาสแม่ การทำแอสเป็กรีแฟคทอริงจึงสามารถช่วยลดคัพพลิง โดยการขึ้นต่อกันระหว่างคลาสลดลงและส่งผลให้คุณภาพซอฟต์แวร์ทั้ง 4 ด้านปรับปรุงดีขึ้น

(5) เรสปอนซ์ฟอร์อะคลาส (Response For a Class - RFC)

จากตารางที่ 4.31 เปรียบเทียบเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นกับเจฮอตดรอว์ เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้น จากการเทียบค่ามาตรวัด RFC รายคลาส พบว่ามีเพียงเจฮอตดรอว์ เวอร์ชั่น 9 ที่ค่ามาตรวัด RFC ไม่เปลี่ยนแปลงหลังทำแอสเป็กรีแฟคทอริง ส่วนเจฮอตดรอว์เวอร์ชั่นอื่น ได้แก่ เวอร์ชั่น 1-8 และ 10-12 ค่ามาตรวัด RFC ของบางคลาสมีค่าเปลี่ยนแปลง สามารถแสดงรายชื่อคลาสที่มีค่ามาตรวัด RFC เปลี่ยนแปลง และค่ามาตรวัด RFC ก่อน-หลังทำแอสเป็กรีแฟคทอริงได้ดังนี้

- เจฮอตดรอว์ เวอร์ชั่น 1 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 4 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 2 คลาสที่มีค่ามาตรวัด RFC เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFC แสดงดังตารางที่ 4.34

ตารางที่ 4.34 รายชื่อคลาส 2 คลาสของเจฮอตดรอว์ เวอร์ชั่น 1 ที่ค่ามาตรวัด RFC เปลี่ยนแปลง

คลาส	มาตรวัด RFC	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.draw.DefaultDrawing	66	65
org.jhotdraw.draw.QuadTreeDrawing	77	76

- เจฮอตดรอว์ เวอร์ชั่น 2 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 5 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 5 คลาสที่มีค่ามาตรวัด RFC เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFC แสดงดังตารางที่ 4.35

ตารางที่ 4.35 รายชื่อคลาส 5 คลาสของเจฮอตดรอว์ เวอร์ชั่น 2 ที่ค่ามาตรวัด RFC เปลี่ยนแปลง

คลาส	มาตรวัด RFC	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.samples.draw.DrawingPanel	89	88
org.jhotdraw.samples.net.NetPanel	71	70
org.jhotdraw.samples.odg.ODGDrawingPanel	85	84
org.jhotdraw.samples.pert.PertPanel	73	72
org.jhotdraw.samples.svg.SVGDrawingPanel	87	86

- เจฮอตดรอว์ เวอร์ชั่น 3 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 9 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 9 คลาสที่มีค่ามาตรวัด RFC เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFC แสดงดังตารางที่ 4.36

ตารางที่ 4.36 รายชื่อคลาส 9 คลาสของเจฮอตดรอว์ เวอร์ชั่น 3 ที่ค่ามาตรวัด RFC เปลี่ยนแปลง

คลาส	มาตรวัด RFC	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.draw.action.GroupAction	45	44
org.jhotdraw.draw.BezierTool	79	78
org.jhotdraw.draw.CreationTool	67	66
org.jhotdraw.samples.odg.action.CombineAction	46	45
org.jhotdraw.samples.odg.action.SplitAction	46	45
org.jhotdraw.samples.odg.PathTool	23	22
org.jhotdraw.samples.svg.action.CombineAction	49	48
org.jhotdraw.samples.svg.action.SplitAction	46	45
org.jhotdraw.samples.svg.PathTool	23	22

- เจฮอตดรอว์ เวอร์ชั่น 4 ที่จัดการเมท็อดคอลซ้ำกันทั้งหมด 12 ตำแหน่งด้วยแอสเป็ก กรณีนี้จัดการเมท็อดคอลที่เรียกใช้งานเมท็อด ClearSelection() พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 4 คลาสที่ค่ามาตรวัด RFC เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFC แสดงดังตารางที่ 4.37

ตารางที่ 4.37 รายชื่อคลาส 4 คลาสของเจฮอตดรอว์ เวอร์ชั่น 4 ที่ค่ามาตรวัด RFC เปลี่ยนแปลง

คลาส	มาตรวัด RFC	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.draw.BezierTool	79	78
org.jhotdraw.draw.BidirectionalConnectionTool	82	81
org.jhotdraw.draw.ConnectionTool	95	94
org.jhotdraw.draw.CreationTool	67	66

- เจฮอตดรอว์ เวอร์ชั่น 5 ที่จัดการเมท็อดคอลซ้ำกันทั้งหมด 12 ตำแหน่งด้วยแอสเป็ก กรณีนี้จัดการเมท็อดคอลที่เรียกใช้งานเมท็อด RemoveAllChildren() พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 8 คลาสที่มีค่ามาตรวัด RFC เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFC แสดงดังตารางที่ 4.38

ตารางที่ 4.38 รายชื่อคลาส 8 คลาสของเจฮอตดรอว์ เวอร์ชั่น 5 ที่ค่ามาตรวัด RFC เปลี่ยนแปลง

คลาส	มาตรวัด RFC	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.samples.draw.DrawApplet	71	70
org.jhotdraw.samples.draw.DrawLiveConnectApplet	76	75
org.jhotdraw.samples.net.NetApplet	74	73
org.jhotdraw.samples.odg.PathTool	23	22
org.jhotdraw.samples.pert.PertApplet	68	67
org.jhotdraw.samples.svg.io.DefaultSVGFigureFactory	54	53
org.jhotdraw.samples.svg.PathTool	23	22
org.jhotdraw.samples.svg.SVGApplet	75	74

- เจฮอตดรอว์ เวอร์ชั่น 6 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 13 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 5 คลาสที่มีค่ามาตรวัด RFC เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFC แสดงดังตารางที่ 4.39

ตารางที่ 4.39 รายชื่อคลาส 5 คลาสของเจฮอตดรอว์ เวอร์ชั่น 6 ที่ค่ามาตรวัด RFC เปลี่ยนแปลง

คลาส	มาตรวัด RFC	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.app.DefaultApplicationModel	42	41
org.jhotdraw.draw.DefaultDrawing	66	65
org.jhotdraw.draw.GridConstrainer	52	50
org.jhotdraw.draw.QuadTreeDrawing	77	76
org.jhotdraw.samples.odg.ODGDrawing	35	34

- เจฮอตดรอว์ เวอร์ชั่น 7 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 15 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 10 คลาสที่มีค่ามาตรวัด RFC เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFC แสดงดังตารางที่ 4.40

ตารางที่ 4.40 รายชื่อคลาส 10 คลาสของเจฮอตดรอว์ เวอร์ชั่น 7 ที่ค่ามาตรวัด RFC เปลี่ยนแปลง

คลาส	มาตรวัด RFC	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.samples.draw.DrawingPanel	89	87
org.jhotdraw.samples.draw.DrawView	99	98
org.jhotdraw.samples.net.NetPanel	71	69
org.jhotdraw.samples.net.NetView	105	104
org.jhotdraw.samples.odg.ODGDrawingPanel	85	83
org.jhotdraw.samples.odg.ODGView	157	156
org.jhotdraw.samples.pert.PertPanel	73	71
org.jhotdraw.samples.pert.PertView	102	101
org.jhotdraw.samples.svg.SVGDrawingPanel	87	85
org.jhotdraw.samples.svg.SVGView	161	160

- เจฮอตดรอว์ เวอร์ชั่น 8 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 18 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 12 คลาสที่มีค่ามาตรวัด RFC เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFC แสดงดังตารางที่ 4.41

ตารางที่ 4.41 รายชื่อคลาส 12 คลาสของเจฮอตดรอว์ เวอร์ชั่น 8 ที่ค่ามาตรวัด RFC เปลี่ยนแปลง

คลาส	มาตรวัด RFC	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.draw.ChopDiamondConnector	19	18
org.jhotdraw.draw.ChopRoundRectangleConnector	18	17
org.jhotdraw.draw.EllipseFigure	34	33
org.jhotdraw.draw.ImageFigure	86	85
org.jhotdraw.draw.RectangleFigure	29	28
org.jhotdraw.draw.RoundRectangleFigure	48	47
org.jhotdraw.draw.TriangleFigure	54	53
org.jhotdraw.samples.net.figures.NodeFigure	49	48
org.jhotdraw.samples.odg.figures.ODGPathFigure	149	148
org.jhotdraw.samples.svg.figures.SVGPathFigure	150	149
org.jhotdraw.samples.svg.figures.SVGTextAreaFigure	130	129
org.jhotdraw.samples.svg.figures.SVGTextFigure	112	111

- เจฮอตดรอว์ เวอร์ชั่น 10 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 29 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 14 คลาสที่มีค่ามาตรวัด RFC เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFC แสดงดังตารางที่ 4.42

ตารางที่ 4.42 รายชื่อคลาส 14 คลาสของเจฮอตดรอว์ เวอร์ชั่น 10 ที่ค่ามาตรวัด RFC เปลี่ยนแปลง

คลาส	**มาตรวัด RFC**	
	ก่อนทำแอสเป็กรีแฟคทอริง	**หลังทำแอสเป็กรีแฟคทอริง**
org.jhotdraw.draw.BezierScaleHandle	37	36
org.jhotdraw.draw.ConnectionEndHandle	19	18
org.jhotdraw.draw.ConnectionStartHandle	17	16
org.jhotdraw.draw.FontSizeHandle	33	32
org.jhotdraw.draw.ImageFigure	86	85
org.jhotdraw.draw.MoveHandle	39	38
org.jhotdraw.draw.RoundRectangleRadiusHandle	28	27
org.jhotdraw.draw.TextAreaTool	47	46
org.jhotdraw.samples.odg.figures.ODGPathFigure	149	148
org.jhotdraw.samples.odg.figures.ODGRectRadiusHandle	32	31
org.jhotdraw.samples.pert.figures.TaskFigure	102	101
org.jhotdraw.samples.svg.figures.SVGImageFigure	100	99
org.jhotdraw.samples.svg.figures.SVGPathFigure	150	149
org.jhotdraw.samples.svg.figures.SVGRectRadiusHandle	32	31

- เจฮอตดรอว์ เวอร์ชั่น 11 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 37 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 21 คลาสที่มีค่ามาตรวัด RFC เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFC แสดงดังตารางที่ 4.43

ตารางที่ 4.43 รายชื่อคลาส 21 คลาสของเจฮอตดรอว์ เวอร์ชั่น 11 ที่ค่ามาตรวัด RFC เปลี่ยนแปลง

คลาส	มาตรวัด RFC	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.draw.BezierNodeHandle	69	68
org.jhotdraw.draw.BezierScaleHandle	37	36
org.jhotdraw.draw.BezierTool	79	78
org.jhotdraw.draw.ConnectionEndHandle	19	18
org.jhotdraw.draw.ConnectionStartHandle	17	16
org.jhotdraw.draw.ConnectorHandle	72	71
org.jhotdraw.draw.FontSizeHandle	33	32
org.jhotdraw.draw.ImageFigure	86	85
org.jhotdraw.draw.LineConnectionFigure	87	86
org.jhotdraw.draw.LineFigure	23	22
org.jhotdraw.draw.RoundRectangleRadiusHandle	28	27
org.jhotdraw.draw.TextAreaTool	47	46
org.jhotdraw.draw.TextTool	47	46
org.jhotdraw.samples.odg.figures.ODGBezierFigure	37	36
org.jhotdraw.samples.odg.figures.ODGPathFigure	149	148
org.jhotdraw.samples.odg.figures.ODGRectRadiusHandle	32	31
org.jhotdraw.samples.pert.figures.TaskFigure	102	101
org.jhotdraw.samples.svg.figures.SVGBezierFigure	41	40
org.jhotdraw.samples.svg.figures.SVGImageFigure	100	99
org.jhotdraw.samples.svg.figures.SVGPathFigure	150	149
org.jhotdraw.samples.svg.figures.SVGRectRadiusHandle	32	31

- เจฮอตดรอว์ เวอร์ชั่น 12 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 44 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 43 คลาสที่มีค่ามาตรวัด RFC เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFC แสดงดังตารางที่ 4.44

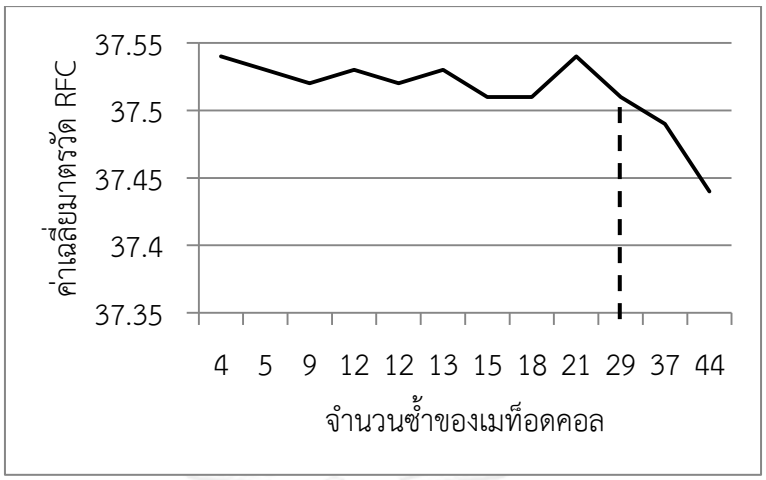
ตารางที่ 4.44 รายชื่อคลาส 43 คลาสของเจฮอตดรอว์ เวอร์ชั่น 12 ที่ค่ามาตรวัด RFC เปลี่ยนแปลง

คลาส	มาตรวัด RFC	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.app.action.AboutAction	15	14
org.jhotdraw.app.action.ClearAction	8	7
org.jhotdraw.app.action.ClearRecentFilesAction	20	19
org.jhotdraw.app.action.CloseAction	7	6
org.jhotdraw.app.action.CopyAction	11	10
org.jhotdraw.app.action.CutAction	11	10
org.jhotdraw.app.action.DeleteAction	21	20
org.jhotdraw.app.action.DuplicateAction	10	9
org.jhotdraw.app.action.ExitAction	64	63
org.jhotdraw.app.action.ExportAction	30	29
org.jhotdraw.app.action.FindAction	8	7
org.jhotdraw.app.action.FocusAction	35	34
org.jhotdraw.app.action.LoadAction	24	23
org.jhotdraw.app.action.MaximizeAction	13	12
org.jhotdraw.app.action.MinimizeAction	13	12
org.jhotdraw.app.action.NewAction	19	18
org.jhotdraw.app.action.OpenAction	49	48
org.jhotdraw.app.action.PasteAction	12	11
org.jhotdraw.app.action.PrintAction	46	45
org.jhotdraw.app.action.RedoAction	23	22
org.jhotdraw.app.action.SaveAction	46	45
org.jhotdraw.app.action.SaveAsAction	4	3
org.jhotdraw.app.action.SelectAllAction	11	10
org.jhotdraw.app.action.UndoAction	23	22
org.jhotdraw.draw.action.ApplyAttributesAction	29	28
org.jhotdraw.draw.action.EditDrawingAction	23	22
org.jhotdraw.draw.action.EditGridAction	23	22

ตารางที่ 4.44 รายชื่อคลาส 43 คลาสของเจฮอตดรอว์ เวอร์ชั่น 12
ที่ค่ามาตรวัด RFC เปลี่ยนแปลง (ต่อ)

คลาส	มาตรวัด RFC	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.draw.action.GroupAction	45	44
org.jhotdraw.draw.action.MoveToBackAction	19	18
org.jhotdraw.draw.action.MoveToFrontAction	20	19
org.jhotdraw.draw.action.PickAttributesAction	27	26
org.jhotdraw.draw.action.SelectSameAction	18	17
org.jhotdraw.draw.action.ToggleGridAction	11	10
org.jhotdraw.draw.action.UngroupAction	5	4
org.jhotdraw.samples.odg.action.CombineAction	46	45
org.jhotdraw.samples.odg.action.SplitAction	46	45
org.jhotdraw.samples.svg.action.CombineAction	49	48
org.jhotdraw.samples.svg.action.SplitAction	46	45
org.jhotdraw.samples.svg.action.ViewSourceAction	36	35
org.jhotdraw.samples.teddy.action.FindAction	12	11
org.jhotdraw.samples.teddy.action.ToggleLineNumbersAction	13	12
org.jhotdraw.samples.teddy.action.ToggleLineWrapAction	13	12
org.jhotdraw.samples.teddy.action.ToggleStatusBarAction	13	12

ผลการเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น โดยนำค่าเฉลี่ยของมาตรวัด RFC จากข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์แต่ละเวอร์ชั่น (ตารางที่ 4.6 – 4.17) มาเปรียบเทียบแสดงด้วยกราฟดังรูปที่ 4.41 พบว่าค่าเฉลี่ยของมาตรวัด RFC มีแนวโน้มลดลงเมื่อจัดการเมท็อดคอลซ้ำกันมากกว่า 29 ตำแหน่ง ส่วนเจฮอตดรอว์ เวอร์ชั่น 9 ที่จัดการเมท็อดคอลซ้ำกันทั้งหมด 21 ตำแหน่ง ค่าเฉลี่ยของมาตรวัด RFC มีค่าสูงสุดเมื่อเทียบกับเจฮอตดรอว์เวอร์ชั่นอื่นๆ เนื่องจากเจฮอตดรอว์ เวอร์ชั่น 9 เป็นเวอร์ชั่นเดียวที่คลาสทั้งหมดมีค่ามาตรวัด RFC ไม่เปลี่ยนแปลงหลังทำแอสเป็กรีแฟคทอริง



รูปที่ 4.41 ค่าเฉลี่ยของมาตรวัด RFC ของเจฮอตดรอว์ 12 เวอร์ชั่น ที่จัดการจำนวนซ้ำเมท็อดคอลแตกต่างกัน

มาตรวัด RFC พิจารณาจากจำนวนเมท็อดทั้งหมดที่ทำงานเพื่อตอบสนองต่ออ็อบเจกต์คลาสหรือเมท็อดภายในคลาส ดังนั้นค่ามาตรวัด RFC จะมีค่าเพิ่มขึ้นหรือลดลงจึงต้องมีการเปลี่ยนแปลงจำนวนเมท็อดที่ทำงานตอบสนองต่อคลาสที่พิจารณา และเนื่องจากงานวิจัยนี้เลือกทำแอสเป็กรีแฟคทอริงโดยสนใจจัดการเมท็อดคอลซ้ำกันภายในคลาสต่างๆด้วยแอสเป็ก จากการทำแอสเป็กรีแฟคทอริงจึงส่งผลให้จำนวนเมท็อดที่ทำงานตอบสนองต่อคลาสมีจำนวนลดลง

เจฮอตดรอว์ เวอร์ชั่น 1-8 และ 10-12 ที่จัดการเมท็อดคอลซ้ำกันทั้งหมด 4, 5, 9, 12, 12, 13, 15, 18, 29, 37 และ 44 ตำแหน่งตามลำดับ ผลแสดงว่าหลังทำแอสเป็กรีแฟคทอริงคลาสส่วนใหญ่ที่มีค่ามาตรวัด RFC เปลี่ยนแปลงจะมีค่ามาตรวัด RFC ลดลงเท่ากับ 1 เนื่องจากสามารถลดจำนวนเมท็อดที่ทำงานตอบสนองต่อคลาสได้เพียง 1 เมท็อดจากการจัดการเมท็อดคอลทั้งหมดที่เรียกใช้งานเมท็อดเดียวกันด้วยแอสเป็ก ตัวอย่างคลาสที่จัดการเมท็อดคอลด้วยแอสเป็กและส่งผลให้ค่ามาตรวัด RFC ของคลาสมีค่าลดลง แสดงดังรูปที่ 4.42 คลาส ImageFigure มีเมท็อดคอลซ้ำกันทั้งหมด 3 ตำแหน่งที่เรียกใช้งานเมท็อด changed() หลังการจัดการเมท็อดคอลทั้ง 3 ตำแหน่งด้วยแอสเป็กส่งผลให้ค่ามาตรวัด RFC ของคลาส ImageFigure มีค่าลดลงเท่ากับ 1 เนื่องจากสามารถลดจำนวนเมท็อดที่ทำงานตอบสนองต่อคลาส ImageFigure ได้ 1 เมท็อด คือเมท็อด changed() ดังนั้นถึงแม้มีจำนวนซ้ำของเมท็อดคอลถึง 3 ตำแหน่ง การทำแอสเป็กรีแฟคทอริงก็จะส่งผลให้ค่ามาตรวัด RFC ของคลาสลดลงเพียง 1 ค่าเท่านั้น

```
package org.jhotdraw.draw;
public class ImageFigure extends AbstractAttributedDecoratedFigure
        implements ImageHolderFigure {
    private Rectangle2D.Double rectangle;
    private byte[] imageData;
    private BufferedImage bufferedImage;
    ...
    public void setImage(byte[] imageData, BufferedImage bufferedImage) {
        willChange();
        this.imageData = imageData;
        this.bufferedImage = bufferedImage;
        //changed();
    }
    public void setImageData(byte[] imageData) {
        willChange();
        this.imageData = imageData;
        this.bufferedImage = null;
        //changed();
    }
    public void setBufferedImage(BufferedImage image) {
        willChange();
        this.imageData = null;
        this.bufferedImage = image;
        //changed();
    }
    ...
}
```

รูปที่ 4.42 คลาส ImageFigure

ตัวอย่างการจัดการเมท็อดคอลดังรูปที่ 4.43 แสดงคลาส ODGPathFigure ที่มีเมท็อดคอล 1 ตำแหน่งที่เรียกใช้งานเมท็อด changed() หลังการจัดการเมท็อดคอลด้วยแอสเป็ก ส่งผลให้ค่ามาตรวัด RFC ของคลาส ODGPathFigure ลดลงเพียงแค่ 1 ค่าเช่นกัน

```
package org.jhotdraw.samples.odg.figures;
public class ODGPathFigure extends AbstractAttributedCompositeFigure
implements ODGFigure {
    public ODGPathFigure() {
        add(new ODGBezierFigure());
        ODGAttributeKeys.setDefaults(this);
    }
   ...
    public void flattenTransform() {
        willChange();
        AffineTransform tx = TRANSFORM.get(this);
        if (tx != null) {
            for (Figure child : getChildren()) {
                ((ODGBezierFigure) child).transform(tx);
                ((ODGBezierFigure) child).flattenTransform();
            }
        }
        TRANSFORM.basicSet(this, null);
        //changed();
    }
}
```

รูปที่ 4.43 คลาส ODGPathFigure

ดังนั้นกรณีที่ค่ามาตรวัด RFC ของบางคลาสมีค่าคงเดิมไม่ลดลง เนื่องจากไม่สามารถจัดการเมท็อดคอลที่ซ้ำกันทั้งหมดภายในคลาสได้ จำนวนเมท็อดที่ทำงานตอบสนองต่อคลาสจึงมีจำนวนเท่าเดิม โดยแสดงให้เห็นชัดเจนในเจฮอตดรอว์ เวอร์ชั่น 9 ที่จัดการเมท็อดคอลซ้ำกันทั้งหมด 21 ตำแหน่ง เนื่องจากจำนวนซ้ำ 21 ตำแหน่งเป็นการคัดกรองจากข้อจำกัดการระบุพอยท์คัทของภาษาแอสเป็กเจ จึงทำให้ยังมีการเรียกใช้งานเมท็อดเดียวกันในตำแหน่งอื่นของคลาส ค่ามาตรวัด RFC ของคลาสจึงมีค่าคงเดิม

ตัวอย่างคลาสดังรูปที่ 4.44 คลาส AttributeKey มีเมท็อดคอลที่เรียกใช้งานเมท็อด changed() โดยสามารถจัดการเมท็อดคอลด้วยแอสเป็กได้เพียง 2 ตำแหน่ง เมท็อดคอลอีก 3 ตำแหน่งไม่สามารถจัดการด้วยแอสเป็กได้ เนื่องจากตำแหน่งของเมท็อดคอลอยู่ภายในเมท็อดที่ซ้อนเมท็อดของคลาสอีกที ทำให้ไม่สามารถระบุตำแหน่งด้วยพอยท์คัทของภาษาแอสเป็กเจ ส่งผลให้ภายในคลาสยังคงมีการเรียกใช้งานเมท็อด changed() ค่ามาตรวัด RFC ของคลาส AttributeKey จึงมีค่าคงเดิมเพราะไม่สามารถลดจำนวนเมท็อดที่ทำงานเพื่อตอบสนองแก่คลาสได้

```
public class AttributeKey<T> {
    public void set(Figure f, T value) {
        f.willChange();
        basicSet(f, value);
        //f.changed();
    }
    public UndoableEdit setUndoable(final Figure figure, final T value, final
ResourceBundleUtil labels) {
        figure.willChange();
        figure.setAttribute(this, value);
        figure.changed();
        UndoableEdit edit = new AbstractUndoableEdit() {
            public void undo() {
                super.undo();
                figure.willChange();
                figure.restoreAttributesTo(restoreData);
                figure.changed();
            }
            public void redo() {
                super.redo();
                figure.willChange();
                figure.setAttribute(AttributeKey.this, value);
                figure.changed();
            }
        };
        return edit;
    }
    public void setClone(Figure f, T value) {
        f.willChange();
        basicSetClone(f, value);
        //f.changed();
    }
}
```

รูปที่ 4.44 คลาส AttributeKey ที่มีการเรียกใช้งานเมท็อด changed()

อีกตัวอย่างของการจัดการเมท็อดคอลซ้ำกันด้วยแอสเป็ก แต่ค่ามาตรวัด RFC ของคลาสไม่ลดลงแสดงดังรูปที่ 4.45 คลาส DragHandle มีเมท็อดคอลที่เรียกใช้งานเมท็อด changed() แต่สามารถจัดการด้วยแอสเป็กได้เพียงแค่ 1 ตำแหน่ง ส่วนอีกตำแหน่งที่อยู่ภายในเมท็อด trackEnd() ไม่สามารถจัดการด้วยแอสเป็กได้ เนื่องจากตำแหน่งของเมท็อดคอลอยู่ในตอนท้ายของลูป for ซึ่งพอยท์คัทของภาษาแอสเป็กเจไม่สามารถระบุตำแหน่งดังกล่าวได้ ดังนั้นค่ามาตรวัด RFC ของคลาส DragHandle จึงมีค่าไม่ลดลงเช่นกัน

```
public class DragHandle extends AbstractHandle {
    private Point2D.Double oldPoint;
    private Figure f;
    public void trackStep(Point anchor, Point lead, int modifiersEx) {
        f = getOwner();
       Point2D.Double newPoint=view.getConstrainer().constrainPoint(view.viewToDrawing(lead));
        AffineTransform tx = new AffineTransform();
        tx.translate(newPoint.x - oldPoint.x, newPoint.y - oldPoint.y);
        f.willChange();
        f.transform(tx);
        //f.changed();
        oldPoint = newPoint;
    }
    public void trackEnd(Point anchor, Point lead, int modifiersEx) {
        if (dropTarget != null) {
            boolean snapBack = dropTarget.handleDrop(dropPoint, draggedFigures, getView());
            if (snapBack) {
                tx = new AffineTransform();
                tx.translate(anchor.x - lead.x, anchor.y - lead.y);
                for (Figure f : draggedFigures) {
                    f.willChange();
                    f.transform(tx);
                    f.changed();
                }
            } else {
                fireUndoableEditHappened(new TransformEdit(getOwner(),tx));
            }
        } else {
            fireUndoableEditHappened(new TransformEdit(getOwner(),tx));
        }
    }
}
```

รูปที่ 4.45 คลาส DragHandle ที่มีการเรียกใช้งานเมท็อด changed()

การทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลที่ซ้ำกันด้วยแอสเป็ก ภายในคลาสหนึ่งคลาสถึงแม้จะมีตำแหน่งของเมท็อดคอลที่ซ้ำกันหลายตำแหน่ง เมื่อจัดการเมท็อดคอลซ้ำกันทั้งหมดด้วยแอสเป็กจะส่งผลให้ค่ามาตรวัด RFC ของคลาสมีค่าลดลงจากเดิมเพียง 1 ค่า ดังนั้นหลังทำแอสเป็กรีแฟคทอริง ค่าเฉลี่ยของมาตรวัด RFC ที่คำนวณจากคลาส 442 คลาสของเจฮอตดรอว์จะมีค่าลดลงมากหรือน้อยจึงขึ้นอยู่กับจำนวนคลาสที่สามารถจัดการเมท็อดคอลซ้ำกันทั้งหมดด้วยแอสเป็ก โดยหากมีจำนวนคลาสที่สามารถจัดการเมท็อดคอลซ้ำกันทั้งหมดด้วยแอสเป็กในจำนวนที่มาก จะส่งผลให้ค่าเฉลี่ยของมาตรวัด RFC ของซอฟต์แวร์ลดลงมากด้วยเช่นกัน

จากผลการทำแอสเป็กรีแฟคทอริงของเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น สามารถแสดงจำนวนคลาสที่สามารถจัดการเมท็อดคอลซ้ำกันทั้งหมดด้วยแอสเป็กได้ดังตารางที่ 4.45 แสดงให้เห็นว่าเจฮอตดรอว์ เวอร์ชั่น 10, 11 และ 12 มีจำนวนคลาสที่สามารถจัดการได้มากกว่าเจฮอตดรอว์เวอร์ชั่นอื่นๆ

ตารางที่ 4.45 จำนวนคลาสที่สามารถจัดการเมท็อดคอลซ้ำกันทั้งหมดด้วยแอสเป็กของเจฮอตดรอว์แต่ละเวอร์ชั่น

เจฮอตดรอว์	**จำนวนซ้ำของเมท็อดคอล**	**จำนวนคลาส**
เวอร์ชั่น 1	4	2
เวอร์ชั่น 2	5	5
เวอร์ชั่น 3	9	9
เวอร์ชั่น 4	12	4
เวอร์ชั่น 5	12	8
เวอร์ชั่น 6	13	5
เวอร์ชั่น 7	15	10
เวอร์ชั่น 8	18	12
เวอร์ชั่น 9	21	0
เวอร์ชั่น 10	29	14
เวอร์ชั่น 11	37	21
เวอร์ชั่น 12	44	43

ดังนั้นปัจจัยที่ส่งผลต่อค่ามาตรวัด RFC จึงได้แก่ จำนวนซ้ำของเมท็อดคอล และจำนวนคลาสที่สามารถจัดการเมท็อดคอลซ้ำกันทั้งหมดด้วยแอสเป็กได้ โดยเจฮอตดรอว์ เวอร์ชั่น 10, 11 และ 12 ที่จัดการเมท็อดคอลซ้ำกันทั้งหมด 29, 37 และ 44 ตำแหน่ง จากจำนวนซ้ำของเมท็อดคอลที่มากอาจส่งผลให้มีตำแหน่งของเมท็อดคอลกระจัดกระจายอยู่ภายในหลายคลาสด้วยเช่นกัน จำนวนคลาสที่สามารถจัดการเมท็อดคอลซ้ำกันทั้งหมดด้วยแอสเป็กของเจฮอตดรอว์ เวอร์ชั่น 10, 11 และ 12 จึงมีจำนวนคลาสมาก ซึ่งส่งผลให้ค่าเฉลี่ยมาตรวัด RFC ของเจฮอตดรอว์ทั้งสามเวอร์ชั่นมีค่าลดลงมาก ดังแสดงในกราฟค่าเฉลี่ยมาตรวัด RFC ที่ค่าเฉลี่ยมีแนวโน้มลดลงเมื่อจำนวนซ้ำของเมท็อดคอลที่จัดการมีจำนวนซ้ำที่มาก และสามารถจัดการคลาสที่เมท็อดคอลซ้ำกันได้ในจำนวนคลาสที่มากเช่นกัน จากความสัมพันธ์ระหว่างมาตรวัด RFC และคุณภาพซอฟต์แวร์ดังตาราง 2.3 ที่แสดงในบทที่ 2 หากค่ามาตรวัด RFC ของซอฟต์แวร์มีค่าลดลง อาจกล่าวได้ว่าคุณภาพซอฟต์แวร์ด้านความสามารถในการทำความเข้าใจและความสามารถการทดสอบปรับปรุงดีขึ้น จากผลมาตรวัด RFC แสดงแนวโน้มค่าเฉลี่ยมาตรวัด RFC ที่การจัดการเมท็อดคอลซ้ำกันมากกว่า 29 ตำแหน่งมีแนวโน้มค่าเฉลี่ยมาตรวัด RFC ลดลงจึงช่วยปรับปรุงคุณภาพซอฟต์แวร์ทั้งสองด้าน อีกทั้งผลของงานวิจัยแสดงผลสอดคล้องกับงานวิจัยของ Marin, Deursen และ Moonen (2007) ที่แนะนำว่าโค้ดที่ซ้ำกันมากกว่า 10 ตำแหน่ง

เหมาะสมเป็นครอสคัทติ้งคอนเซิร์นที่จะนำมาจัดการด้วยแอสเป็ก แต่หากพิจารณาการจัดการเมท็อดคอลในจำนวนซ้ำที่มากขึ้นดังในการทดลองคือจำนวนซ้ำเท่ากับ 37 และ 44 ก็แสดงผลค่าเฉลี่ยมาตรวัด RFC ของซอฟต์แวร์ที่ลดลงมากขึ้นด้วยเช่นกันซึ่งแสดงผลในแง่บวกจากการนำแอสเป็กมาใช้ เช่นเดียวกับผลงานวิจัยของ Hanenberg, Kleinschmager และ Walter (2009) ที่นำเสนอว่าการจัดการโค้ดซ้ำกันมากกว่า 36 ตำแหน่ง การใช้แอสเป็กจะช่วยประหยัดเวลาพัฒนาได้ ดังนั้นการทำแอสเป็กรีแฟคทอริงโดยจัดการจำนวนซ้ำของเมท็อดคอลในจำนวนซ้ำที่มาก ซึ่งตำแหน่งของเมท็อดคอลกระจัดกระจายอยู่ภายในคลาสจำนวนมากด้วย สามารถส่งผลให้ค่าเฉลี่ยมาตรวัด RFC ของซอฟต์แวร์มีค่าลดลงและปรับปรุงคุณภาพซอฟต์แวร์ด้านความสามารถในการทำความเข้าใจและความสามารถการทดสอบ

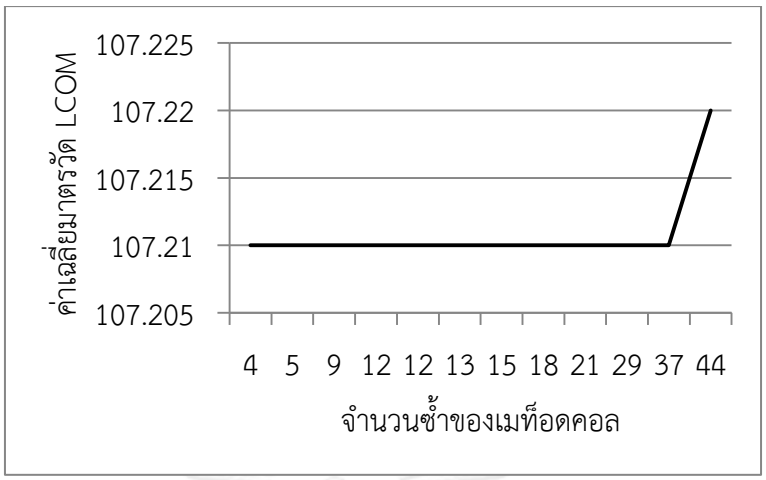
(6) แลคออฟโคฮีชันอินเมท็อด (Lack of Cohesion in Methods - LCOM)

จากตารางที่ 4.31 เป็นการเปรียบเทียบเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นกับเจฮอตดรอว์ เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้น โดยเปรียบเทียบค่ามาตรวัด LCOM รายคลาส พบว่ามีเพียงเจฮอตดรอว์ เวอร์ชั่น 12 ที่หลังจากทำแอสเป็กรีแฟคทอริค่ามาตรวัด LCOM ของบางคลาสมีค่าเปลี่ยนแปลง สามารถแสดงรายชื่อคลาสที่มีค่ามาตรวัด LCOM เปลี่ยนแปลง และค่ามาตรวัด LCOM ก่อน-หลังทำแอสเป็กรีแฟคทอริงของเจฮอตดรอว์ เวอร์ชั่น 12 ได้ดังนี้

ตารางที่ 4.46 รายชื่อคลาส 3 คลาสของเจฮอตดรอว์ เวอร์ชั่น 12 ที่ค่ามาตรวัด LCOM เปลี่ยนแปลง

คลาส	**มาตรวัด LCOM**	
	ก่อนทำแอสเป็กรีแฟคทอริง	**หลังทำแอสเป็กรีแฟคทอริง**
org.jhotdraw.draw.action.MoveToBackAction	6	8
org.jhotdraw.draw.action.MoveToFrontAction	6	8
org.jhotdraw.draw.action.UngroupAction	0	1

ผลการเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น โดยนำค่าเฉลี่ยของมาตรวัด LCOM จากข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์แต่ละเวอร์ชั่น (ตารางที่ 4.6 – 4.17) มาเปรียบเทียบแสดงด้วยกราฟดังรูปที่ 4.46 พบว่าค่าเฉลี่ยของมาตรวัด LCOM ของการจัดการเมท็อดคอลซ้ำกัน 44 ตำแหน่งมีค่าสูงกว่าเล็กน้อยเมื่อเทียบกับการจัดการด้วยจำนวนซ้ำของเมท็อดคอลอื่นๆ โดยมีค่าเฉลี่ยของมาตรวัด LCOM เท่ากับ 107.22



รูปที่ 4.46 ค่าเฉลี่ยของมาตรวัด LCOM ของเจฮอตดรอว์แต่ละเวอร์ชั่น ที่จัดการจำนวนซ้ำเมท็อดคอลแตกต่างกัน

มาตรวัด LCOM พิจารณาจากแอตทริบิวต์หรือตัวแปรอินสแตนซ์ที่ประกาศภายในคลาสและถูกเข้าถึงจากเมท็อดของคลาสเพื่อนำไปใช้งานภายในเมท็อด ค่ามาตรวัด LCOM จึงเป็นมาตรวัดสำหรับวัดระดับความเกี่ยวข้องกันระหว่างเมท็อดภายในคลาสจากการใช้งานแอตทริบิวต์ของคลาสร่วมกัน โดยค่ามาตรวัด LCOM ควรมีค่าน้อยเพื่อแสดงว่าระหว่างเมท็อดภายในคลาสมีความเกี่ยวข้องกัน โดยการใช้ชุดแอตทริบิวต์หรือตัวแปรอินสแตนซ์คล้ายกันสามารถแสดงให้เห็นว่าเมท็อดภายในคลาสทำงานตอบสนองในเรื่องเดียวกัน ดังนั้นค่ามาตรวัด LCOM จะมีค่าเพิ่มขึ้นหรือลดลงเกิดจากการใช้งานแอตทริบิวต์คลาสระหว่างเมท็อดเกิดการเปลี่ยนแปลง

เจฮอตดรอว์ เวอร์ชั่น 12 ที่จัดการเมท็อดคอลซ้ำกัน 44 ตำแหน่งด้วยแอสเป็ก เป็นเจฮอตดรอว์เวอร์ชั่นเดียวที่หลังทำแอสเป็กรีแฟคทอริงมีค่ามาตรวัด LCOM ของคลาสเปลี่ยนแปลง โดยมีคลาสทั้งหมด 3 คลาสที่มีค่ามาตรวัด LCOM เพิ่มขึ้น จากการพิจารณาคลาสทั้ง 3 คลาส ได้แก่ MoveToBackAction, MoveToFrontAction และ UngroupAction การเปลี่ยนแปลงของค่ามาตรวัด LCOM เกิดจากสาเหตุเดียวกัน โดยสามารถแสดงตัวอย่างดังรูปที่ 4.47 คลาส MoveToFront Action มีเมท็อดคอลที่เรียกใช้งานเมท็อด configureAction() อยู่ 1 ตำแหน่ง โดยเป็นเมท็อดคอลที่มีลักษณะการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์ของคลาส และเมท็อดคอลมีการส่งอาร์กิวเมนต์ซึ่งเป็นแอตทริบิวต์คลาสด้วยเช่นกัน ดังนั้นเมท็อดคอล labels.configureAction(this, ID); ซึ่งอยู่ภายในคอนสตรัคเตอร์ของคลาส จึงส่งผลให้คอนสตรัคเตอร์มีการเข้าถึงตัวแปรของคลาส ได้แก่ labels และ ID เช่นเดียวกับเมท็อด actionPerformed ที่มีการเข้าถึงตัวแปรคลาสทั้ง 2 ตัวแปรเหมือนกัน ก่อนการทำแอสเป็กรีแฟคทอริงทั้งสองเมท็อดจึงมีความเกี่ยวข้องกันจากการเข้าถึงตัวแปรที่เหมือนกัน ดังนั้นเมื่อจัดการเมท็อดคอลดังกล่าวด้วยแอสเป็กจึงทำให้คอนสตรัคเตอร์ของคลาสไม่ได้เข้าถึงตัวแปรคลาสทั้ง 2 ตัวแปรอีก หลังทำแอสเป็กรีแฟคทอริงค่ามาตรวัด LCOM ของคลาสจึงมีค่าเพิ่มขึ้นเพราะความเกี่ยวข้องกันระหว่างเมท็อดมีค่าลดลง

นอกจากนี้เมื่อพิจารณาคลาสอื่นๆภายในเจฮอตดรอว์ เวอร์ชั่น 12 ที่หลังทำแอสเป็กรีแฟคทอริงค่ามาตรวัด LCOM ของคลาสไม่เปลี่ยนแปลง เนื่องจากก่อนทำแอสเป็กรีแฟคทอริง ตัวแปรของ

คลาสที่ใช้ในเมท็อดคอลไม่มีการเข้าถึงจากเมท็อดอื่นๆของคลาส ดังนั้นเมื่อจัดการเมท็อดคอลด้วยแอสเป็กจึงไม่ส่งผลให้ความเกี่ยวข้องกันระหว่างเมท็อดของคลาสเกิดการเปลี่ยนแปลง ค่ามาตรวัด LCOM จึงมีค่าคงเดิม ดังรูปที่ 4.48

```
public class MoveToFrontAction extends AbstractSelectedAction {
    private ResourceBundleUtil labels = ResourceBundleUtil.getLAFBundle(
            "org.jhotdraw.draw.Labels", Locale.getDefault());
    public static String ID = "moveToFront";
    public MoveToFrontAction(DrawingEditor editor) {
        super(editor);
        //labels.configureAction(this, ID);
    }
    public void actionPerformed(java.awt.event.ActionEvent e) {
        final DrawingView view = getView();
        final LinkedList<Figure> figures = new LinkedList<Figure>(view.getSelectedFigures());
        bringToFront(view, figures);
        fireUndoableEditHappened(new AbstractUndoableEdit() {
            public String getPresentationName() {
                return labels.getString(ID);
            }
            public void redo() throws CannotRedoException {
                super.redo();
                MoveToFrontAction.bringToFront(view, figures);
            }
            public void undo() throws CannotUndoException {
                super.undo();
                MoveToBackAction.sendToBack(view, figures);
            }
        }
        );
    }
}
```

รูปที่ 4.47 คลาส MoveToFrontAction

```
public class CopyAction extends AbstractAction {
    public final static String ID = "copy";
    private ResourceBundleUtil labels;
    public CopyAction() {
        labels = ResourceBundleUtil.getLAFBundle("org.jhotdraw.app.Labels");
        //labels.configureAction(this, ID);
    }
   public void actionPerformed(ActionEvent evt) {
        Component focusOwner = KeyboardFocusManager.
                getCurrentKeyboardFocusManager().
                getPermanentFocusOwner();
        if (focusOwner != null && focusOwner instanceof JComponent) {
            JComponent component = (JComponent) focusOwner;
            component.getTransferHandler().exportToClipboard(
                    component,
                    component.getToolkit().getSystemClipboard(),
                    TransferHandler.COPY
                    );
        }
    }
}
```

รูปที่ 4.48 คลาส CopyAction

4.5 การเปรียบเทียบค่ามาตรวัดเชิงแอสเป็ก

จากผลการทดลองได้เจฮอตดรอว์ทั้งหมด 12 เวอร์ชั่นที่จัดการจำนวนซ้ำของเมท็อดคอล ได้แก่ 4, 5, 9, 12, 13, 15, 18, 21, 29, 37 และ 44 ตำแหน่งตามลำดับ การวัดค่ามาตรวัดเชิงแอสเป็กของเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นพิจารณาจากทั้งไฟล์คลาสและไฟล์แอสเป็ก ดังนั้นค่ามาตรวัดเชิงแอสเป็ก 10 มาตรวัดของเจฮอตดรอว์แต่ละเวอร์ชั่นจึงวัดคลาส 442 คลาสและแอสเป็ก 1 แอสเป็ก เบื้องต้นผู้วิจัยเปรียบเทียบค่ามาตรวัดเชิงแอสเป็กของคลาสทั้งหมดในเจฮอตดรอว์แต่ละเวอร์ชั่นกับเจฮอตดรอว์ เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้นที่ไม่ได้ปรับปรุงด้วยการทำแอสเป็กรีแฟคทอริง เพื่อพิจารณามาตรวัดที่มีค่ามาตรวัดของคลาสเปลี่ยนแปลงหลังทำแอสเป็กรีแฟคทอริง จากการพิจารณาเฉพาะคลาสของเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นสามารถแสดงดังตารางที่ 4.47

ตารางที่ 4.47 มาตรวัดเชิงแอสเป็กซึ่งมีคลาสที่ค่ามาตรวัดเปลี่ยนแปลงหลังทำแอสเป็กรีแฟคทอริง

เจฮอตดรอว์	**จำนวนซ้ำเมท็อดคอล**	**มาตรวัดเชิงแอสเป็ก**									
		WOM	DIT	NOC	CFA	CMC	CBM	CDA	CAE	RFM	LCO
เวอร์ชั่น 1	4	-	-	-	-	-	-	-	✓	✓	-
เวอร์ชั่น 2	5	-	-	-	-	-	-	-	✓	-	-
เวอร์ชั่น 3	9	-	-	-	-	✓	✓	-	✓	✓	-
เวอร์ชั่น 4	12	-	-	-	-	✓	✓	-	✓	✓	-
เวอร์ชั่น 5	12	-	-	-	-	-	-	-	✓	✓	-
เวอร์ชั่น 6	13	-	-	-	-	-	-	-	✓	✓	-
เวอร์ชั่น 7	15	-	-	-	-	-	-	-	✓	✓	-
เวอร์ชั่น 8	18	-	-	-	-	✓	✓	-	✓	✓	-
เวอร์ชั่น 9	21	-	-	-	-	-	-	-	✓	✓	-
เวอร์ชั่น 10	29	-	-	-	-	-	-	-	✓	✓	-
เวอร์ชั่น 11	37	-	-	-	-	-	-	-	✓	✓	-
เวอร์ชั่น 12	44	-	-	-	-	✓	✓	-	✓	-	-

<u>หมายเหตุ</u>

เครื่องหมาย - แสดงถึง ไม่มีคลาสที่ค่ามาตรวัดเปลี่ยนแปลงหลังทำแอสเป็กรีแฟคทอริง

เครื่องหมาย ✓ แสดงถึง มีคลาสที่ค่ามาตรวัดเปลี่ยนแปลงหลังทำแอสเป็กรีแฟคทอริง

เนื่องจากแอสเป็กเป็นส่วนที่สร้างขึ้นมาใหม่หลังทำแอสเป็กรีแฟคทอริง เจฮอตดรอว์ เวอร์ชั่น 7.1 จึงไม่มีไฟล์แอสเป็กที่จะนำมาเปรียบเทียบได้ ดังนั้นจึงพิจารณาค่ามาตรวัดของแอสเป็กในเจฮอตดรอว์แต่ละเวอร์ชั่นแยกออกมาดังตารางที่ 4.48 โดยเทียบค่ามาตรวัดของแอสเป็กกับค่าเฉลี่ยของมาตรวัดที่คำนวณจากคลาสทั้งหมด 442 คลาส และพิจารณาค่ามาตรวัดของแอสเป็กออกเป็นทั้งหมดสามกรณี (1) ค่ามาตรวัดของแอสเป็กมีค่าสูงกว่าค่าเฉลี่ยของคลาส (+) (2) ค่ามาตรวัดของแอสเป็กมีค่าต่ำกว่าค่าเฉลี่ยของคลาส (-) และ (3) ค่ามาตรวัดของแอสเป็กมีค่าเท่ากับค่าเฉลี่ยของคลาส (0)

ตารางที่ 4.48 มาตรวัดเชิงแอสเป็กของแอสเป็กหลังทำแอสเป็กรีแฟคทอริง
(พิจารณาเฉพาะไฟล์แอสเป็ก)

เจฮอตดรอว์	**จำนวนซ้ำเมท็อดคอล**	**มาตรวัดเชิงแอสเป็ก**									
		WOM	DIT	NOC	CFA	CMC	CBM	CDA	CAE	RFM	LCO
เวอร์ชั่น 1	4	-	-	-	-	-	-	+	0	-	-
เวอร์ชั่น 2	5	-	-	-	-	-	-	+	-	-	-
เวอร์ชั่น 3	9	-	-	-	-	-	-	+	-	-	-
เวอร์ชั่น 4	12	-	-	-	-	-	-	+	-	-	-
เวอร์ชั่น 5	12	-	-	-	-	-	-	+	-	-	-
เวอร์ชั่น 6	13	-	-	-	-	-	-	+	-	-	-
เวอร์ชั่น 7	15	-	-	-	-	-	-	+	-	-	-
เวอร์ชั่น 8	18	-	-	-	-	-	-	+	-	-	-
เวอร์ชั่น 9	21	-	-	-	+	-	+	+	0	-	-
เวอร์ชั่น 10	29	-	-	-	-	-	-	+	-	-	-
เวอร์ชั่น 11	37	-	-	-	-	-	-	+	-	-	-
เวอร์ชั่น 12	44	-	-	-	-	-	-	+	-	-	-

หมายเหตุ
เครื่องหมาย + แสดงถึงค่ามาตรวัดของแอสเป็กมีค่าสูงกว่าค่าเฉลี่ยของคลาส
เครื่องหมาย - แสดงถึงค่ามาตรวัดของแอสเป็กมีค่าต่ำกว่าค่าเฉลี่ยของคลาส
เครื่องหมาย 0 แสดงถึงค่ามาตรวัดของแอสเป็กมีค่าเท่ากับค่าเฉลี่ยของคลาส

การพิจารณาข้อมูลค่ามาตรวัดของแต่ละมาตรวัดเชิงแอสเป็กสามารถนำมาพิจารณาตามกรอบการวิเคราะห์ข้อมูลทั้งสามส่วน ได้แก่ (1) การเปรียบเทียบค่ามาตรวัดก่อนและหลังทำแอสเป็กรีแฟคทอริง (2) การเปรียบเทียบค่าเฉลี่ยมาตรวัดระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น และ (3) การวิเคราะห์ผลมาตรวัดต่อคุณภาพซอฟต์แวร์ รายละเอียดการวิเคราะห์ข้อมูลของแต่ละมาตรวัดมีดังนี้

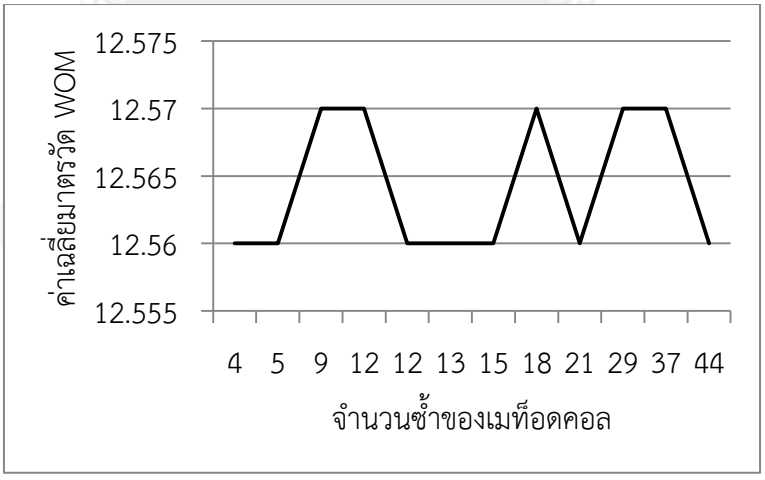
(1) เวทเทดโอเปอร์เรชั่นอินมอดูล (Weighted Operations in Module - WOM)

จากตารางที่ 4.47 เป็นการเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นกับเจฮอตดรอว์เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้น โดยพิจารณาเฉพาะคลาส พบว่าหลังทำแอสเป็กรีแฟคทอริงค่ามาตรวัด WOM ของทุกคลาสในเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นมีค่าไม่เปลี่ยนแปลง แต่จากการพิจารณาเฉพาะแอสเป็กในเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นดังตารางที่ 4.48 พบว่าค่ามาตรวัด WOM ของแอสเป็กของเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นมีค่าต่ำกว่าค่าเฉลี่ยมาตรวัด WOM ที่คำนวณจากคลาสทั้ง 442 คลาส ค่ามาตรวัด WOM ของแอสเป็กของเจฮอตดรอว์แต่ละเวอร์ชั่นแสดงดังตารางที่ 4.49

ตารางที่ 4.49 ค่ามาตรวัด WOM ของแอสเป็กในเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น หลังทำแอสเป็กรีแฟคทอริง

เจฮอตดรอว์ เวอร์ชั่น / ค่ามาตรวัด WOM	1	2	3	4	5	6	7	8	9	10	11	12
แอสเป็ก	1	2	5	5	1	3	5	5	1	4	5	2

การเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น โดยนำค่าเฉลี่ยของมาตรวัด WOM จากข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์แต่ละเวอร์ชั่น (ตารางที่ 4.19 - 4.30) มาเปรียบเทียบกันแสดงด้วยกราฟดังรูปที่ 4.49 พบว่าการทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลในจำนวนซ้ำของเมท็อดคอลที่แตกต่างกันไม่แสดงแนวโน้มของค่าเฉลี่ยมาตรวัด WOM โดยค่าเฉลี่ยของมาตรวัด WOM ที่แสดงในกราฟมีค่าสูงหรือต่ำสลับกันไป



รูปที่ 4.49 ค่าเฉลี่ยของมาตรวัด WOM ของเจฮอตดรอว์แต่ละเวอร์ชั่นที่จัดการจำนวนซ้ำเมท็อดคอลแตกต่างกัน

นิยามมาตรวัด WOM มีความคล้ายกันกับนิยามของมาตรวัด WMC ของมาตรวัดเชิงวัตถุ สำหรับการพิจารณาค่ามาตรวัดของคลาสจากจำนวนเมท็อดที่อิมพลีเม้นต์ภายในคลาส แต่นิยามของมาตรวัด WOM ได้กำหนดนิยามสำหรับการพิจารณาแอสเป็กเพิ่มเติม ดังนั้นการพิจารณาค่ามาตรวัด WOM ของคลาสและแอสเป็กจะพิจารณาจากจำนวนโอเปอเรชั่นภายในโมดูล ได้แก่ จำนวนเมท็อดภายในคลาส และจำนวนแอดไวซ์ จำนวนอินโทรดักชันภายในแอสเป็ก

จากกรณีที่หลังทำแอสเป็กรีแฟคทอริง ค่ามาตรวัด WOM ของคลาสไม่เปลี่ยนแปลง เนื่องจากงานวิจัยนี้เลือกทำแอสเป็กรีแฟคทอริงโดยจัดการเพียงเมท็อดคอลซ้ำกันในหลายคลาสด้วยแอสเป็ก ทำให้มีแค่เมท็อดคอลซึ่งเป็นเพียงคำสั่งภายในคลาสถูกนำออกจากคลาส การทำแอสเป็กรีแฟคทอริงจึงไม่ส่งผลให้จำนวนเมท็อดที่อิมพลีเม้นต์ภายในคลาสเปลี่ยนแปลง แต่สำหรับแอสเป็กของเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น เนื่องจากแอสเป็กเป็นส่วนที่สร้างขึ้นมาจากการทำแอสเป็กรีแฟคทอริงด้วยนิยามมาตรวัด WOM พิจารณาแอสเป็กจากจำนวนแอดไวซ์ที่อิมพลีเม้นต์ในแอสเป็ก จึงทำให้ค่ามาตรวัด WOM ของแอสเป็กในเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นมีค่าแตกต่างกันไปตามจำนวนแอดไวซ์ โดยจำนวนแอดไวซ์จะมีจำนวนมากหรือน้อยขึ้นอยู่กับความเหมือนของตำแหน่งเมท็อดคอลที่จัดการทั้งหมด คือหากสามารถใช้พอยท์คัทและแอดไวซ์เดียวกันสำหรับจัดการเมท็อดคอลทั้งหมดได้จะส่งผลให้ค่ามาตรวัด WOM มีค่าน้อย แต่หากตำแหน่งของเมท็อดคอลมีตำแหน่งอยู่ก่อนหรือหลังพอยท์คัทแตกต่างกันจะส่งผลให้ต้องใช้แอดไวซ์มากและค่ามาตรวัด WOM ของแอสเป็กจึงมีค่าสูงตาม นอกจากนี้หากในเมท็อดคอลที่ต้องการจัดการมีการเข้าถึงตัวแปรอ็อบเจกต์ของคลาสจะส่งผลทำให้ต้องสร้างแอดไวซ์สำหรับเก็บค่าของตัวแปรอ็อบเจกต์เพื่อนำมาใช้ในแอสเป็กด้วยเช่นกัน

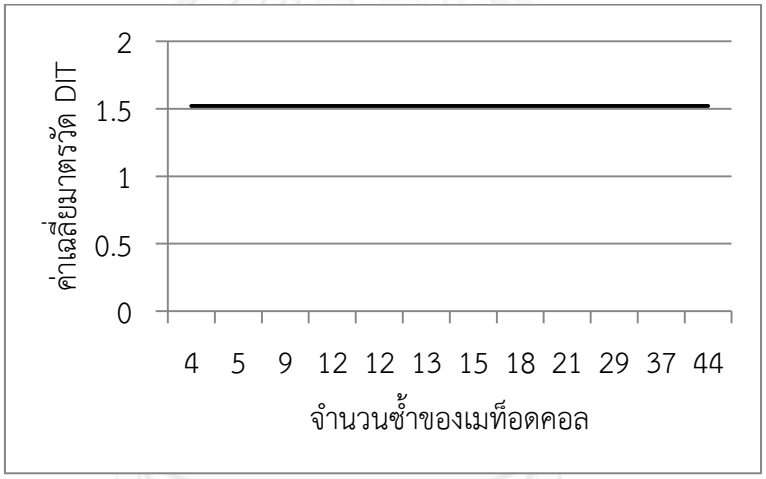
จากตารางที่ 2.4 ความสัมพันธ์ระหว่างมาตรวัดเชิงแอสเป็กกับคุณภาพซอฟต์แวร์ ที่อ้างอิงจากงานวิจัยของ Sirbi และ Kulkarni (2013) โดยมาตรวัด WOM สามารถส่งผลกระทบต่อคุณภาพซอฟต์แวร์ 2 ด้าน ได้แก่ ความซับซ้อนและความสามารถการบำรุงรักษา จากผลการเปรียบเทียบค่ามาตรวัด WOM รายคลาสพบว่าเมื่อเทียบกับเจฮอตดรอว์ เวอร์ชั่น 7.1 ค่ามาตรวัดของคลาสมีค่าไม่เปลี่ยนแปลง ดังนั้นซอฟต์แวร์จึงมีค่ามาตรวัด WOM ของแอสเป็กเพิ่มเข้ามาจากการสร้างแอดไวซ์ภายในแอสเป็ก จึงสามารถส่งผลต่อคุณภาพซอฟต์แวร์ทั้งสองด้านด้วยเช่นกัน ทำให้ความซับซ้อนของซอร์สโค้ดเพิ่มสูงขึ้นและการบำรุงรักษาซอฟต์แวร์ทำได้ยากขึ้น

(2) เดพออฟอินเฮอริแทนทรี (Depth of Inheritance Tree - DIT)

จากตารางที่ 4.47 แสดงการเปรียบเทียบเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นกับเจฮอตดรอว์ เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้น โดยพิจารณาเฉพาะคลาส พบว่าค่ามาตรวัด DIT ก่อนและหลังทำแอสเป็กรีแฟคทอริงของคลาสทั้งหมดในเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นมีค่าคงเดิมไม่เปลี่ยนแปลง และจากตารางที่ 4.48 แสดงให้เห็นว่าค่ามาตรวัด DIT ของแอสเป็กในเจฮอตดรอว์ทุกเวอร์ชั่นมีค่าเท่ากับ 0 จึงมีค่าต่ำกว่าค่าเฉลี่ยมาตรวัด DIT ของคลาสทั้ง 442 คลาส

ผลการเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นจึงให้ผลเช่นเดียวกัน โดยนำค่าเฉลี่ยของมาตรวัด DIT จากข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์แต่ละเวอร์ชั่น (ตารางที่ 4.19 - 4.30) มาเปรียบเทียบ แสดงด้วยกราฟดังรูปที่ 4.50 พบว่าการทำแอสเป็กรีแฟคทอริงเพื่อจัดการจำนวนซ้ำของเมท็อดคอลที่แตกต่างกันไม่ส่งผลให้ค่ามาตรวัด DIT เปลี่ยนแปลง โดยค่าเฉลี่ยของมาตรวัด DIT มีค่าคงเดิมเท่ากับ 1.52

นิยามมาตรวัด DIT ของมาตรวัดเชิงแอสเป็กเหมือนกันกับนิยามมาตรวัด DIT ของมาตรวัดเชิงวัตถุ โดยมาตรวัด DIT พิจารณาจากระดับความลึกของจำนวนชั้นการรับทอดจากมอดูลไปยังรูทในลำดับชั้น หลังทำแอสเป็กรีแฟคทอริงค่ามาตรวัด DIT ของคลาสไม่เปลี่ยนแปลงและค่ามาตรวัด DIT ของแอสเป็กเท่ากับ 0 เนื่องจากการทำแอสเป็กรีแฟคทอริงเลือกจัดการเพียงเมท็อดคอลที่ซ้ำกันภายในคลาส จึงไม่ส่งผลให้ลำดับชั้นคลาสเปลี่ยนแปลง ค่ามาตรวัด DIT ของคลาสจึงมีค่าคงเดิม และการสร้างแอสเป็กขึ้นมาใหม่ไม่ได้สร้างขึ้นมาภายในลำดับชั้น ค่ามาตรวัด DIT ของแอสเป็กจึงมีค่าเท่ากับ 0

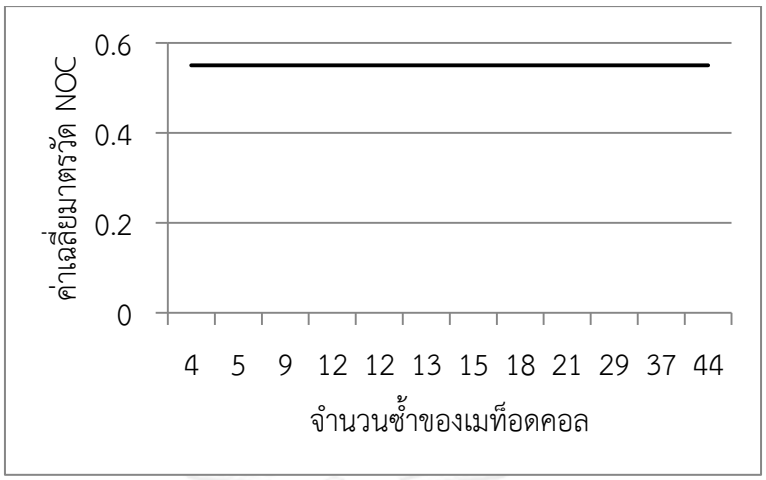


รูปที่ 4.50 ค่าเฉลี่ยของมาตรวัด DIT ของเจฮอตดรอว์แต่ละเวอร์ชั่นที่จัดการจำนวนซ้ำเมท็อดคอลแตกต่างกัน

(3) นัมเบอร์ออฟชิลเดรน (Number of Children - NOC)

จากตารางที่ 4.47 แสดงการเปรียบเทียบเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นกับเจฮอตดรอว์ เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้น โดยพิจารณาเฉพาะคลาส พบว่าค่ามาตรวัด NOC ก่อนและหลังทำแอสเป็กรีแฟคทอริงของคลาสทั้งหมดในเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นมีค่าคงเดิมไม่เปลี่ยนแปลง และจากตารางที่ 4.48 แสดงให้เห็นว่าค่ามาตรวัด NOC ของแอสเป็กในเจฮอตดรอว์ทุกเวอร์ชั่นมีค่าเท่ากับ 0 จึงมีค่าต่ำกว่าค่าเฉลี่ยมาตรวัด NOC ของคลาสทั้ง 442 คลาส

ผลการเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นจึงให้ผลเช่นเดียวกัน โดยนำค่าเฉลี่ยของมาตรวัด NOC จากข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์แต่ละเวอร์ชั่น (ตารางที่ 4.19 – 4.30) มาเปรียบเทียบ แสดงด้วยกราฟดังรูปที่ 4.51 พบว่าการทำแอสเป็กรีแฟคทอริงเพื่อจัดการจำนวนซ้ำของเมท็อดคอลที่แตกต่างกันไม่ส่งผลให้ค่ามาตรวัด NOC เปลี่ยนแปลง โดยค่าเฉลี่ยของมาตรวัด NOC มีค่าคงเดิมเท่ากับ 0.55



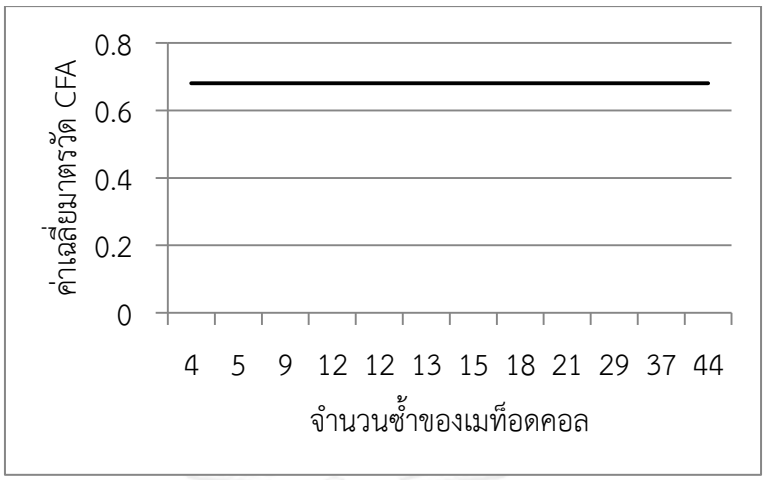
รูปที่ 4.51 ค่าเฉลี่ยของมาตรวัด NOC ของเจฮอตดรอว์แต่ละเวอร์ชั่น ที่จัดการจำนวนซ้ำเมท็อดคอลแตกต่างกัน

นิยามมาตรวัด NOC ของมาตรวัดเชิงแอสเป็กเหมือนกันกับนิยามมาตรวัด NOC ของมาตรวัดเชิงวัตถุ โดยมาตรวัด NOC พิจารณาจากจำนวนคลาสลูกหรือจำนวนซับแอสเป็กของมอดูล หลังทำแอสเป็กรีแฟคทอริงค่ามาตรวัด NOC ของคลาสไม่เปลี่ยนแปลงและค่ามาตรวัด NOC ของแอสเป็กมีค่าเท่ากับ 0 เนื่องจากการทำแอสเป็กรีแฟคทอริงเลือกจัดการเพียงเมท็อดคอลที่ซ้ำกันภายในคลาสด้วยแอสเป็ก จึงไม่ส่งผลกระทบต่อการเปลี่ยนแปลงของจำนวนคลาสลูก ค่ามาตรวัด NOC ของคลาสจึงมีค่าคงเดิม และการสร้างแอสเป็กขึ้นมาใหม่โดยไม่ได้รับทอดมาจากแอสเป็กอื่นๆ และไม่มีการสร้างซับแอสเป็ก ค่ามาตรวัด NOC ของแอสเป็กจึงมีค่าเท่ากับ 0

(4) คัพพลิงออนฟิลด์แอคเซส (Coupling on Field Access - CFA)

จากตารางที่ 4.47 แสดงการเปรียบเทียบเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นกับเจฮอตดรอว์ เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้น โดยพิจารณาเฉพาะคลาส พบว่าค่ามาตรวัด CFA ก่อนและหลังทำแอสเป็กรีแฟคทอริงของคลาสทั้งหมดในเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นมีค่าไม่เปลี่ยนแปลง และจากตารางที่ 4.48 แสดงให้เห็นว่าค่ามาตรวัด CFA ของแอสเป็กในเจฮอตดรอว์เวอร์ชั่นส่วนใหญ่มีค่าเท่ากับ 0 จึงมีค่าต่ำกว่าค่าเฉลี่ยมาตรวัด CFA ที่คำนวณจากคลาส 442 คลาส ขณะที่เจฮอตดรอว์เวอร์ชั่น 9 ที่มีค่ามาตรวัด CFA ของแอสเป็กเท่ากับ 2 มีค่าสูงกว่าค่าเฉลี่ยมาตรวัด CFA ของคลาส

ผลการเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น โดยนำค่าเฉลี่ยของมาตรวัด CFA จากข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์แต่ละเวอร์ชั่น (ตารางที่ 4.19 – 4.30) มาเปรียบเทียบแสดงด้วยกราฟดังรูปที่ 4.52 พบว่าค่าเฉลี่ยของมาตรวัด CFA ระหว่างเจฮอตดรอว์แต่ละเวอร์ชั่นมีค่าไม่แตกต่างกัน จากข้างต้นเจฮอตดรอว์ เวอร์ชั่น 9 มีค่ามาตรวัด CFA ของแอสเป็กแตกต่างคือเท่ากับ 2 แต่เมื่อคำนวณค่าเฉลี่ยจากคลาสและแอสเป็กทั้งหมด 443 คลาส/แอสเป็ก จึงทำให้ค่าเฉลี่ยมาตรวัด CFA มีค่าไม่แตกต่างกัน ดังนั้นการทำแอสเป็กรีแฟคทอริงเพื่อจัดการจำนวนซ้ำของเมท็อดคอลที่แตกต่างกันไม่ส่งผลให้ค่ามาตรวัด CFA เปลี่ยนแปลง โดยค่าเฉลี่ยของมาตรวัด CFA มีค่าคงเดิมเท่ากับ 0.68



รูปที่ 4.52 ค่าเฉลี่ยของมาตรวัด CFA ของเจฮอตดรอว์แต่ละเวอร์ชั่น ที่จัดการจำนวนซ้ำเมท็อดคอลแตกต่างกัน

มาตรวัด CFA พิจารณาจากจำนวนมอดูลทั้งหมดที่มีฟิลด์ที่สามารถเข้าถึงได้จากมอดูลที่พิจารณา การทำแอสเป็กรีแฟคทอริงไม่ส่งผลให้ค่ามาตรวัด CFA ของคลาสเปลี่ยนแปลง เนื่องจากเมท็อดคอลที่จัดการด้วยแอสเป็ก ส่วนใหญ่ภายในเมท็อดคอลไม่ได้เข้าถึงฟิลด์ของคลาสอื่นโดยตรง เมื่อจัดการเมท็อดคอลออกจากคลาส ค่ามาตรวัด CFA ของคลาสจึงมีค่าคงเดิม แต่สำหรับภายในแอสเป็กของเจฮอตดรอว์ เวอร์ชั่น 9 มีการเข้าถึงฟิลด์ของคลาสอื่นโดยตรงทั้งหมด 2 คลาส ค่ามาตรวัด CFA ของแอสเป็กจึงมีค่าเท่ากับ 2 ขณะที่ในแอสเป็กของเจฮอตดรอว์เวอร์ชั่นอื่นๆ ไม่ได้เข้าถึงฟิลด์ของคลาสอื่น ค่ามาตรวัด CFA ของแอสเป็กเลยมีค่ามาตรวัดเท่ากับ 0

(5) คัพพลิงออนเมท็อดคอล (Coupling on Method Call - CMC)

จากตารางที่ 4.47 แสดงการเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นกับเจฮอตดรอว์เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้น โดยพิจารณาเฉพาะคลาส พบว่าหลังทำแอสเป็กรีแฟคทอริงเจฮอตดรอว์ เวอร์ชั่น 3, 4, 8 และ 12 ค่ามาตรวัด CMC ของบางคลาสมีค่าเปลี่ยนแปลง และจากตารางที่ 4.48 พบว่าค่ามาตรวัด CMC ของแอสเป็กในเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นมีค่าต่ำกว่าค่าเฉลี่ยมาตรวัด CMC ที่คำนวณจากคลาส 442 คลาส จากการพิจารณาเฉพาะคลาสสามารถแสดงรายชื่อคลาสที่มีค่ามาตรวัด CMC เปลี่ยนแปลง และค่ามาตรวัด CMC ก่อน-หลังทำแอสเป็กรีแฟคทอริงของเจฮอตดรอว์เวอร์ชั่น 3, 4, 8 และ 12 ได้ดังนี้

- เจฮอตดรอว์ เวอร์ชั่น 3 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 9 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 2 คลาสที่มีค่ามาตรวัด CMC เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CMC แสดงดังตารางที่ 4.50

ตารางที่ 4.50 รายชื่อคลาส 2 คลาสของเจฮอตดรอว์ เวอร์ชั่น 3 ที่ค่ามาตรวัด CMC เปลี่ยนแปลง

คลาส	มาตรวัด CMC	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.samples.odg.PathTool	5	4
org.jhotdraw.samples.svg.PathTool	5	4

- เจฮอตดรอว์ เวอร์ชั่น 4 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 12 ตำแหน่งด้วยแอสเป็ก กรณีนี้เป็นการจัดการเมท็อดคอลที่เรียกใช้งานเมท็อด ClearSelection() พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 1 คลาสที่มีค่ามาตรวัด CMC เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CMC แสดงดังตารางที่ 4.51

ตารางที่ 4.51 รายชื่อคลาส 1 คลาสของเจฮอตดรอว์ เวอร์ชั่น 4 ที่ค่ามาตรวัด CMC เปลี่ยนแปลง

คลาส	มาตรวัด CMC	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.draw.BidirectionalConnectionTool	9	8

- เจฮอตดรอว์ เวอร์ชั่น 8 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 18 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 8 คลาสที่มีค่ามาตรวัด CMC เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CMC แสดงดังตารางที่ 4.52

ตารางที่ 4.52 รายชื่อคลาส 8 คลาสของเจฮอตดรอว์ เวอร์ชั่น 8 ที่ค่ามาตรวัด CMC เปลี่ยนแปลง

คลาส	มาตรวัด CMC	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.draw.EllipseFigure	4	3
org.jhotdraw.draw.ImageFigure	11	10
org.jhotdraw.draw.RectangleFigure	3	2
org.jhotdraw.draw.RoundRectangleFigure	7	6
org.jhotdraw.samples.odg.figures.ODGPathFigure	23	22
org.jhotdraw.samples.svg.figures.SVGPathFigure	22	21
org.jhotdraw.samples.svg.figures.SVGTextAreaFigure	16	15
org.jhotdraw.samples.svg.figures.SVGTextFigure	19	18

- เจฮอตดรอว์ เวอร์ชั่น 12 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 44 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 2 คลาสที่มีค่ามาตรวัด CMC เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CMC แสดงดังตารางที่ 4.53

ตารางที่ 4.53 รายชื่อคลาส 2 คลาสของเจฮอตดรอว์ เวอร์ชั่น 12 ที่ค่ามาตรวัด CMC เปลี่ยนแปลง

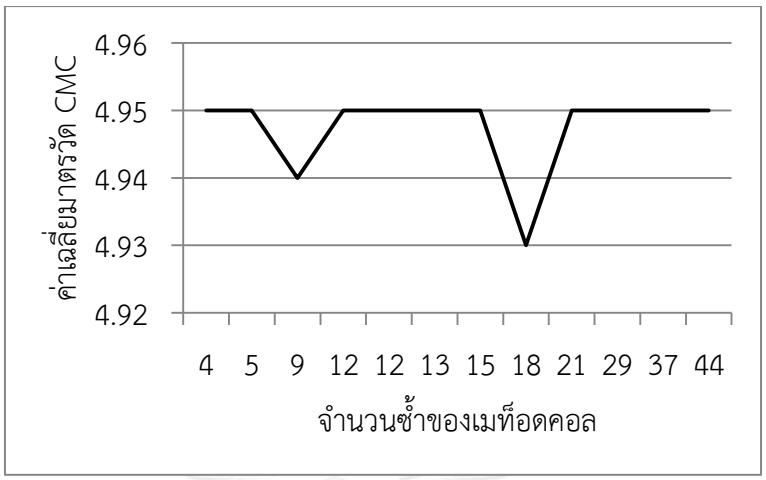
คลาส	**มาตรวัด CMC**	
	ก่อนทำแอสเป็กรีแฟคทอริง	**หลังทำแอสเป็กรีแฟคทอริง**
org.jhotdraw.draw.action.SelectSameAction	4	3
org.jhotdraw.draw.action.UngroupAction	2	1

หลังทำแอสเป็กรีแฟคทอริงจากการพิจารณาแอสเป็กของเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น พบค่ามาตรวัด CMC ของทุกแอสเป็กมีค่าดังตารางที่ 4.54

ตารางที่ 4.54 ค่ามาตรวัด CMC ของแอสเป็กในเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น หลังทำแอสเป็กรีแฟคทอริง

เจฮอตดรอว์ เวอร์ชั่น / ค่ามาตรวัด CMC	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
แอสเป็ก	2	2	2	3	2	4	2	2	4	3	3	4

การเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น โดยนำค่าเฉลี่ยของมาตรวัด CMC จากข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์แต่ละเวอร์ชั่น (ตารางที่ 4.19 - 4.30) มาเปรียบเทียบ แสดงด้วยกราฟดังรูปที่ 4.53 พบว่าหลังทำแอสเป็กรีแฟคทอริง เจฮอตดรอว์ เวอร์ชั่น 3 และ 8 ที่จัดการเมท็อดคอลซ้ำกันทั้งหมด 9 และ 18 ตำแหน่งตามลำดับ มีค่าเฉลี่ยมาตรวัด CMC ลดลงมากกว่าเจฮอตดรอว์เวอร์ชั่นอื่นๆเล็กน้อย โดยมีค่าเฉลี่ยมาตรวัด CMC เท่ากับ 4.94 และ 4.93 ตามลำดับ



รูปที่ 4.53 ค่าเฉลี่ยของมาตรวัด CMC ของเจฮอตดรอว์แต่ละเวอร์ชั่น ที่จัดการจำนวนซ้ำเมท็อดคอลแตกต่างกัน

มาตรวัด CMC พิจารณาจากจำนวนมอดูลทั้งหมดที่มีโอเปอเรชั่นที่สามารถเรียกใช้งานได้จากมอดูลที่พิจารณา ค่ามาตรวัด CMC ของคลาสจึงเป็นความสัมพันธ์ระหว่างคลาสโดยนับจำนวนคลาสทั้งหมดที่สามารถเรียกใช้งานเมท็อดได้ โดยหลังทำแอสเป็กรีแฟคทอริงมีเพียงเจฮอตดรอว์บางเวอร์ชั่นที่ค่ามาตรวัด CMC ของบางคลาสมีค่าลดลง แสดงตัวอย่างการลดลงของค่ามาตรวัด CMC จากเจฮอตดรอว์ เวอร์ชั่น 12 ที่จัดการเมท็อดคอลที่เรียกใช้งานเมท็อด configureAction() ของคลาส ResourceBundleUtil โดยหลังทำแอสเป็กรีแฟคทอริงมีเพียง 2 คลาสที่มีค่ามาตรวัด CMC ลดลงดังแสดงในตารางที่ 4.53 ข้างต้น ขอยกมาอธิบายหนึ่งคลาสดังรูปที่ 4.54 ภายในคลาส UngroupAction มีเมท็อดคอลที่เรียกใช้งานเมท็อด configureAction() ทั้งหมด 2 ตำแหน่ง หลังทำแอสเป็กรีแฟคทอริงทำให้ภายในคลาสไม่มีการเรียกใช้งานเมท็อด configureAction() ของคลาส ResourceBundleUtil และจากการตรวจสอบตำแหน่งอื่นในคลาส UngroupAction พบว่าไม่มีการเรียกใช้งานเมท็อดอื่นของคลาส ResourceBundleUtil ทำให้สามารถลดความสัมพันธ์ระหว่างคลาส UngroupAction และ ResourceBundleUtil ที่เกิดจากการเรียกใช้งานเมท็อด ค่ามาตรวัด CMC ของคลาส UngroupAction จึงมีค่าลดลง 1 ค่า

```
public class UngroupAction extends GroupAction {
    public final static String ID = "selectionUngroup";
    private CompositeFigure prototype;
    public UngroupAction(DrawingEditor editor) {
        super(editor, new GroupFigure(), false);
        //labels.configureAction(this, ID);
    }
    public UngroupAction(DrawingEditor editor, CompositeFigure prototype) {
        super(editor, prototype, false);
        //labels.configureAction(this, ID);
    }
}
```

รูปที่ 4.54 คลาส UngroupAction ที่มีค่ามาตรวัด CMC ลดลงหลังทำแอสเป็กรีแฟคทอริง

สำหรับคลาสส่วนใหญ่ภายในเจฮอตดรอว์ เวอร์ชั่น 12 ถึงแม้จัดการเมท็อดคอลที่เรียกใช้งานเมท็อด configureAction() ด้วยแอสเป็กแล้วก็ตาม แต่ค่ามาตรวัด CMC ยังมีค่าคงเดิม ดังตัวอย่างรูปที่ 4.55 ภายในคลาส CloseAction หลังทำแอสเป็กรีแฟคทอริง ภายในคลาสยังคงมีการเรียกใช้งานเมท็อด getLAFBundle() ของคลาส ResourceBundleUtil ส่งผลให้ไม่สามารถลดความสัมพันธ์ระหว่างคลาส CloseAction กับคลาส ResourceBundleUtil ที่เกิดจากการเรียกใช้งานเมท็อดได้ ดังนั้นค่ามาตรวัด CMC ของคลาส CloseAction จึงมีค่าคงเดิมไม่ลดลง

```
public class CloseAction extends AbstractSaveBeforeAction {
    public final static String ID = "close";
    private ResourceBundleUtil labels;
    public CloseAction(Application app) {
        super(app);
        labels = ResourceBundleUtil.getLAFBundle("org.jhotdraw.app.Labels");
        //labels.configureAction(this, ID);
    }
    @Override protected void doIt(View view) {
        if (view != null && view.getApplication() != null) {
            view.getApplication().
                    dispose(view);
        }
    }
}
```

รูปที่ 4.55 คลาส CloseAction ที่มีค่ามาตรวัด CMC คงเดิมหลังทำแอสเป็กรีแฟคทอริง

สำหรับค่ามาตรวัด CMC ของแอสเป็กพิจารณาความสัมพันธ์ระหว่างแอสเป็กกับคลาสโดยนับจำนวนคลาสทั้งหมดที่แอสเป็กสามารถเรียกใช้งานเมท็อดได้ เนื่องจากแอสเป็กเป็นส่วนที่สร้างขึ้นมาใหม่หลังจากทำแอสเป็กรีแฟคทอริง ดังนั้นค่ามาตรวัด CMC ของแอสเป็กจะมีค่ามากหรือน้อยขึ้นอยู่กับจำนวนคลาสที่แอสเป็กเรียกใช้งานเมท็อด โดยค่ามาตรวัด CMC ของแอสเป็กในเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นแสดงดังตารางที่ 4.54 ข้างต้น ส่วนใหญ่มีค่ามาตรวัดอยู่ในช่วง 2 ถึง 4

เนื่องจากค่าเฉลี่ยคำนวณจากค่ามาตรวัด CMC ของคลาส 442 คลาส และแอสเป็ก 1 แอสเป็ก ดังนั้นกราฟค่าเฉลี่ยมาตรวัด CMC ของเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น ดังรูปที่ 4.53 ข้างต้น จึงแสดงให้เห็นว่าเจฮอตดรอว์ เวอร์ชั่น 8 ที่จัดการเมท็อดคอลซ้ำกันทั้งหมด 18 ตำแหน่งมีค่าเฉลี่ยต่ำสุด เนื่องจากมีคลาสถึง 8 คลาสที่มีค่ามาตรวัด CMC ลดลงและค่ามาตรวัด CMC ของแอสเป็กมีค่าเพียงแค่ 2 ซึ่งเป็นค่าที่ต่ำเมื่อเทียบกับค่ามาตรวัดของแอสเป็กในเจฮอตดรอว์เวอร์ชั่นอื่นๆ และเจฮอตดรอว์ เวอร์ชั่น 3 ที่จัดการเมท็อดคอลซ้ำกันทั้งหมด 9 ตำแหน่ง มีคลาส 2 คลาสที่มีค่ามาตรวัด CMC ลดลง และค่ามาตรวัด CMC ของแอสเป็กเท่ากับ 2 เช่นกันจึงทำให้ค่าเฉลี่ยมาตรวัด CMC มีค่าต่ำรองลงมา ขณะที่เจฮอตดรอว์เวอร์ชั่น 12 ที่จัดการเมท็อดคอลซ้ำกัน 44 ตำแหน่ง ถึงแม้มีคลาส 2 คลาสที่มีค่ามาตรวัด CMC ลดลง แต่ค่ามาตรวัด CMC ของแอสเป็กเท่ากับ 4 ซึ่งเป็นค่าที่สูงจึงทำให้ค่าเฉลี่ยมาตรวัด CMC มีค่าไม่แตกต่างกับเจฮอตดรอว์ เวอร์ชั่นอื่นๆ ที่ไม่มีคลาสที่มีค่ามาตรวัด CMC ลดลงเลย

(6) คัพพลิงบีทวีนมอดูล (Coupling between Modules - CBM)

จากตารางที่ 4.47 แสดงการเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นกับเจฮอตดรอว์เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้น โดยพิจารณาเฉพาะคลาส พบว่าหลังทำแอสเป็กรีแฟคทอริง เจฮอตดรอว์ เวอร์ชั่น 3, 4, 8 และ 12 ค่ามาตรวัด CBM ของบางคลาสมีค่าเปลี่ยนแปลง และจากตารางที่ 4.48 พบว่าค่ามาตรวัด CBM ของทุกแอสเป็กในเจฮอตดรอว์ส่วนใหญ่มีค่าต่ำกว่าค่าเฉลี่ยมาตรวัด CBM ที่คำนวณจากคลาสทั้ง 442 คลาส ขณะที่เจฮอตดรอว์ เวอร์ชั่น 9 มีค่ามาตรวัด CBM ของแอสเป็กสูงกว่าค่าเฉลี่ยมาตรวัด CBM จากการพิจารณาเฉพาะคลาสสามารถแสดงรายชื่อคลาสที่มีค่ามาตรวัด CBM เปลี่ยนแปลง และค่ามาตรวัด CBM ก่อน-หลังทำแอสเป็กรีแฟคทอริงของเจฮอตดรอว์เวอร์ชั่น 3, 4, 8 และ 12 ได้ดังนี้

- เจฮอตดรอว์ เวอร์ชั่น 3 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 9 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 2 คลาสที่ค่ามาตรวัด CBM เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CBM แสดงดังตารางที่ 4.55

ตารางที่ 4.55 รายชื่อคลาส 2 คลาสของเจฮอตดรอว์ เวอร์ชั่น 3 ที่ค่ามาตรวัด CBM เปลี่ยนแปลง

คลาส	**มาตรวัด CBM**	
	ก่อนทำแอสเป็กรีแฟคทอริง	**หลังทำแอสเป็กรีแฟคทอริง**
org.jhotdraw.samples.odg.PathTool	5	4
org.jhotdraw.samples.svg.PathTool	5	4

- เจฮอตดรอว์ เวอร์ชั่น 4 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 12 ตำแหน่งด้วยแอสเป็ก กรณีนี้จัดการเมท็อดคอลที่เรียกใช้งานเมท็อด ClearSelection() พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 1 คลาสที่ค่ามาตรวัด CBM เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CBM แสดงดังตารางที่ 4.56

ตารางที่ 4.56 รายชื่อคลาส 1 คลาสของเจฮอตดรอว์ เวอร์ชั่น 4 ที่ค่ามาตรวัด CBM เปลี่ยนแปลง

คลาส	**มาตรวัด CBM**	
	ก่อนทำแอสเป็กรีแฟคทอริง	**หลังทำแอสเป็กรีแฟคทอริง**
org.jhotdraw.draw.BidirectionalConnectionTool	9	8

- เจฮอตดรอว์ เวอร์ชั่น 8 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 18 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 8 คลาสที่มีค่ามาตรวัด CBM เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CBM แสดงดังตารางที่ 4.57

ตารางที่ 4.57 รายชื่อคลาส 8 คลาสของเจฮอตดรอว์ เวอร์ชั่น 8 ที่ค่ามาตรวัด CBM เปลี่ยนแปลง

คลาส	**มาตรวัด CBM**	
	ก่อนทำแอสเป็กรีแฟคทอริง	**หลังทำแอสเป็กรีแฟคทอริง**
org.jhotdraw.draw.EllipseFigure	4	3
org.jhotdraw.draw.ImageFigure	12	11
org.jhotdraw.draw.RectangleFigure	3	2
org.jhotdraw.draw.RoundRectangleFigure	7	6
org.jhotdraw.samples.odg.figures.ODGPathFigure	26	25
org.jhotdraw.samples.svg.figures.SVGPathFigure	23	22
org.jhotdraw.samples.svg.figures.SVGTextAreaFigure	16	15
org.jhotdraw.samples.svg.figures.SVGTextFigure	19	18

- เจฮอตดรอว์ เวอร์ชั่น 12 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 44 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 2 คลาสที่มีค่ามาตรวัด CBM เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CBM แสดงดังตารางที่ 4.58

ตารางที่ 4.58 รายชื่อคลาส 2 คลาสของเจฮอตดรอว์ เวอร์ชั่น 12 ที่ค่ามาตรวัด CBM เปลี่ยนแปลง

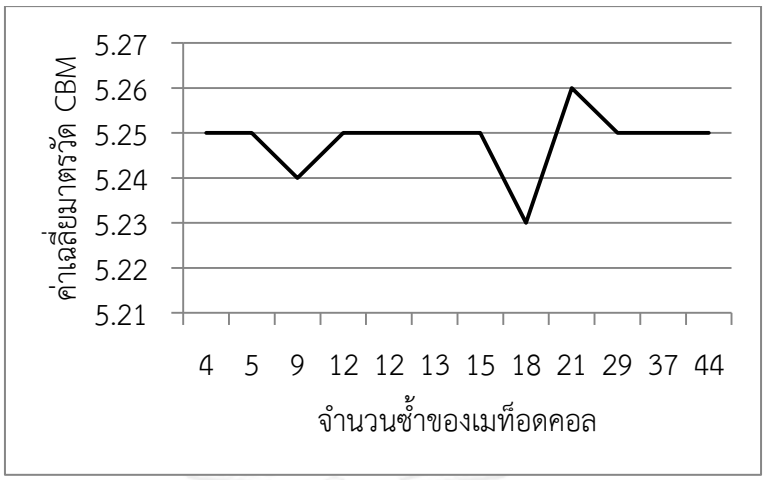
คลาส	**มาตรวัด CBM**	
	ก่อนทำแอสเป็กรีแฟคทอริง	**หลังทำแอสเป็กรีแฟคทอริง**
org.jhotdraw.draw.action.SelectSameAction	4	3
org.jhotdraw.draw.action.UngroupAction	2	1

หลังทำแอสเป็กรีแฟคทอริงจากการพิจารณาแอสเป็กของเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น สามารถแสดงค่ามาตรวัด CBM ของทุกแอสเป็กดังตารางที่ 4.59

ตารางที่ 4.59 ค่ามาตรวัด CBM ของแอสเป็กในเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น หลังทำแอสเป็กรีแฟคทอริง

เจฮอตดรอว์ เวอร์ชั่น / ค่ามาตรวัด CBM	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
แอสเป็ก	2	2	2	3	2	4	2	2	6	3	3	4

การเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น โดยนำค่าเฉลี่ยของมาตรวัด CBM จากข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์แต่ละเวอร์ชั่น (ตารางที่ 4.19 – 4.30) มาเปรียบเทียบ แสดงด้วยกราฟได้ดังรูปที่ 4.56 พบว่าหลังทำแอสเป็กรีแฟคทอริงเจฮอตดรอว์ เวอร์ชั่น 3 และ 8 ที่จัดการเมท็อดคอลซ้ำกันทั้งหมด 9 และ 18 ตำแหน่งตามลำดับมีค่าเฉลี่ยมาตรวัด CBM ต่ำกว่าเจฮอตดรอว์เวอร์ชั่นอื่นๆเล็กน้อย โดยค่าเฉลี่ยมาตรวัด CBM เท่ากับ 5.24 และ 5.23 ตามลำดับ ส่วนเจฮอตดรอว์เวอร์ชั่น 9 ที่จัดการเมท็อดคอลซ้ำกันทั้งหมด 21 ตำแหน่ง มีค่าเฉลี่ยมาตรวัด CBM สูงกว่าเวอร์ชั่นอื่นเล็กน้อย โดยค่าเฉลี่ยมาตรวัด CBM เท่ากับ 5.26



รูปที่ 4.56 ค่าเฉลี่ยของมาตรวัด CBM ของเจฮอตดรอว์แต่ละเวอร์ชั่น ที่จัดการจำนวนซ้ำเมท็อดคอลแตกต่างกัน

นิยามมาตรวัด CBM คล้ายกันกับนิยามมาตรวัด CBO ของมาตรวัดเชิงวัตถุตรงที่พิจารณาความสัมพันธ์ระหว่างคลาสที่เกิดจากการเรียกใช้งานเมท็อดเหมือนกัน แต่แตกต่างกันที่นิยามมาตรวัด CBO พิจารณาความสัมพันธ์ระหว่างคลาสที่เกิดจากการประกาศตัวแปรอินสแตนซ์ของคลาสอื่นด้วย ขณะที่มาตรวัด CBM พิจารณาความสัมพันธ์ระหว่างคลาสที่เกิดจากการเข้าถึงฟิลด์ของคลาสอื่นแทน ดังนั้นมาตรวัด CBM จึงพิจารณาจำนวนความสัมพันธ์ระหว่างมอดูลที่เกิดจากการเรียกใช้งานโอเปอเรชั่นและการเข้าถึงฟิลด์ของมอดูลอื่น จากนิยามของมาตรวัด CBM อาจกล่าวได้ว่าเป็นการพิจารณาร่วมกันระหว่างมาตรวัด CMC และมาตรวัด CFA เพราะมาตรวัด CMC พิจารณาความสัมพันธ์ระหว่างคลาสหรือแอสเป็กที่เกิดจากการเรียกใช้งานเมท็อด ขณะที่มาตรวัด CFA พิจารณาความสัมพันธ์ระหว่างคลาสหรือแอสเป็กที่เกิดจากการเข้าถึงฟิลด์ของคลาส กล่าวคือค่ามาตรวัด CBM ของคลาสหรือแอสเป็กมีค่าเท่ากับค่ามาตรวัด CMC และค่ามาตรวัด CFA รวมกัน แต่ไม่นับความสัมพันธ์ที่ซ้ำกันในกรณีที่คลาสมีทั้งเมท็อดที่สามารถเรียกใช้งานและฟิลด์ที่สามารถเข้าถึงได้

ตัวอย่างการพิจารณาค่ามาตรวัด CBM ของคลาส ViewPropertyAction ที่มีค่ามาตรวัดเท่ากับ 3 โดยคลาส ViewPropertyAction มีความสัมพันธ์กับคลาส 3 คลาส เป็นความสัมพันธ์ที่เกิดจากการเรียกใช้งานเมท็อดทั้งหมด 2 คลาส (มาตรวัด CMC) ได้แก่ คลาส View และคลาส AbstractViewAction และความสัมพันธ์ที่เกิดจากการเข้าถึงฟิลด์ทั้งหมด 1 คลาส (มาตรวัด CFA) ได้แก่ คลาส Actions เนื่องจากคลาสทั้งหมดจากมาตรวัด CMC และมาตรวัด CFA ไม่ซ้ำกัน ค่ามาตรวัด CBM จึงมีเท่ากับค่ามาตรวัด CMC และ CFA รวมกัน

อีกตัวอย่างการพิจารณาค่ามาตรวัด CBM ของคลาส FocusAction ที่มีค่ามาตรวัดเท่ากับ 3 เช่นกัน โดยเป็นความสัมพันธ์ที่เกิดจากการเรียกใช้งานเมท็อดทั้งหมด 3 คลาส (มาตรวัด CMC) ได้แก่ คลาส AbstractAction คลาส ResourceBundleUtil และคลาส View และความสัมพันธ์ที่เกิดจากการเข้าถึงฟิลด์ทั้งหมด 1 คลาส (มาตรวัด CFA) ได้แก่ คลาส View เนื่องจากคลาส View มีทั้งเมท็อดและฟิลด์ที่คลาส FocusAction สามารถเรียกใช้งานและเข้าถึงได้ ดังนั้นค่ามาตรวัด CBM ของคลาส

FocusAction จึงมีค่าเท่ากับ 3 เป็นการรวมกันระหว่างมาตรวัด CMC และมาตรวัด CFA โดยไม่นับความสัมพันธ์ของคลาสที่ซ้ำ

กราฟค่าเฉลี่ยมาตรวัด CBM ของเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นดังรูปที่ 4.56 ข้างต้น จึงมีลักษณะคล้ายกราฟค่าเฉลี่ยมาตรวัด CMC (รูปที่ 4.53) และกราฟค่าเฉลี่ยมาตรวัด CFA (รูปที่ 4.52) รวมกัน โดยกราฟของมาตรวัด CFA หลังทำแอสเป็กรีแฟคทอริงมีเพียงเจฮอตดรอว์ เวอร์ชั่น 9 ที่ค่ามาตรวัด CFA ของแอสเป็กมีค่ามากกว่า 0 แต่ค่าไม่สูงพอที่จะแสดงให้เห็นความแตกต่างในกราฟค่าเฉลี่ยมาตรวัด CFA แต่เมื่อนำค่ามาตรวัด CFA ของแอสเป็กมาคำนวณรวมกับค่ามาตรวัด CMC ของแอสเป็ก ทำให้ค่ามาตรวัด CBM ของแอสเป็กในเจฮอตดรอว์ เวอร์ชั่น 9 มีค่าสูงเท่ากับ 6 และส่งผลให้ค่าเฉลี่ยมาตรวัด CBM ของเจฮอตดรอว์ เวอร์ชั่น 9 มีค่าสูงกว่าเจฮอตดรอว์เวอร์ชั่นอื่นๆ ขณะที่เจฮอตดรอว์ เวอร์ชั่น 8 ที่จัดการเมท็อดคอลซ้ำกัน 18 ตำแหน่ง แสดงค่าเฉลี่ยของมาตรวัด CBM ลดลงมากกว่าเจฮอตดรอว์ เวอร์ชั่นอื่นๆ โดยผลแสดงไปในทิศทางเดียวกันกับการพิจารณาด้วยมาตรวัด CBO ของมาตรวัดเชิงวัตถุ ดังนั้นผลของค่าเฉลี่ยมาตรวัด CBM ที่ลดลงมากจึงเกิดจากการจัดการลักษณะเมท็อดคอลที่ส่วนใหญ่เรียกใช้งานเมท็อดผ่านชื่อคลาสโดยตรง เมื่อจัดการเมท็อดคอลด้วยแอสเป็กจึงสามารถลดความสัมพันธ์ระหว่างคลาสได้มาก

จากตารางที่ 2.4 ความสัมพันธ์ระหว่างมาตรวัดเชิงแอสเป็กกับคุณภาพซอฟต์แวร์ ที่อ้างอิงจากงานวิจัยของ Sirbi และ Kulkarni (2013) โดยมาตรวัด CBM สามารถส่งผลกระทบต่อคุณภาพซอฟต์แวร์ 2 ด้าน ได้แก่ ความสามารถการบำรุงรักษาและความสามารถการนำกลับมาใช้ใหม่ จากผลการเปรียบเทียบมาตรวัด CBM แสดงให้เห็นว่าการทำแอสเป็กรีแฟคทอริงโดยจัดการเมท็อดคอลที่มีลักษณะการเรียกใช้งานเมท็อดผ่านชื่อคลาสโดยตรงสามารถช่วยลดความสัมพันธ์หรือการขึ้นต่อกันระหว่างคลาสได้ และส่งผลให้คุณภาพซอฟต์แวร์ทั้งสองด้านปรับปรุงดีขึ้น

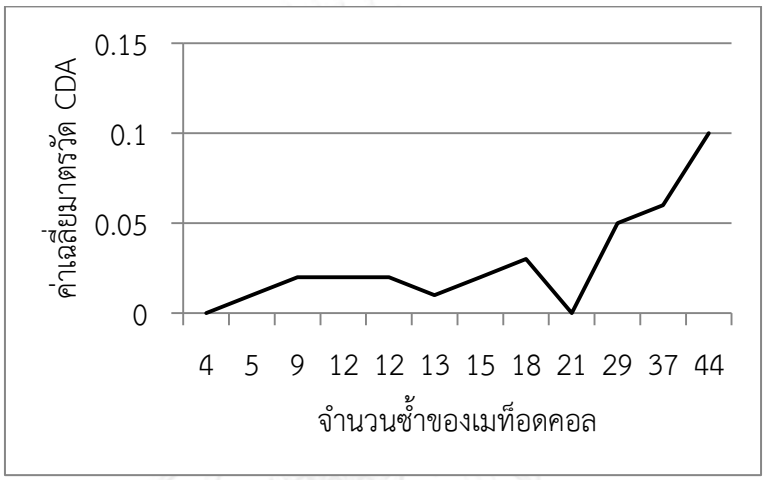
(7) ครอสคัทติ้งดีกรีออฟแอนแอสเป็ก (Crosscutting Degree of an Aspect - CDA)

จากตารางที่ 4.47 แสดงการเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นกับเจฮอตดรอว์เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้น โดยพิจารณาเฉพาะคลาส พบว่าหลังทำแอสเป็กรีแฟคทอริงค่ามาตรวัด CDA ของคลาสทั้งหมดในเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นมีค่าไม่เปลี่ยนแปลง แต่จากตารางที่ 4.48 พบว่าค่ามาตรวัด CDA ของแอสเป็กในเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นมีค่าสูงกว่าค่าเฉลี่ยมาตรวัด CDA ที่คำนวณจากคลาสทั้ง 442 คลาส สามารถแสดงค่ามาตรวัด CDA ของแอสเป็กในเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นได้ดังตารางที่ 4.60

ตารางที่ 4.60 ค่ามาตรวัด CDA ของแอสเป็กในเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น
หลังทำแอสเป็กรีแฟคทอริง

เจฮอตดรอว์ เวอร์ชั่น / **ค่ามาตรวัด CDA**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
แอสเป็ก	2	5	9	11	9	6	10	12	2	23	26	44

การเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น โดยนำค่าเฉลี่ยของมาตรวัด CDA จากข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์แต่ละเวอร์ชั่น (ตารางที่ 4.19 – 4.30) มาเปรียบเทียบ แสดงด้วยกราฟดังรูปที่ 4.57 แต่เนื่องจากหลังทำแอสเป็กรีแฟคทอริงค่ามาตรวัด CDA ของคลาสมีค่าคงเดิม ดังนั้นค่าเฉลี่ยมาตรวัด CDA จึงมีค่าขึ้นอยู่กับค่ามาตรวัด CDA ของแอสเป็กในเจฮอตดรอว์แต่ละเวอร์ชั่น พบว่าเจฮอตดรอว์ เวอร์ชั่น 12 ที่จัดการเมท็อดคอลซ้ำกันทั้งหมด 44 ตำแหน่งมีค่าเฉลี่ยมาตรวัด CDA สูงสุดเท่ากับ 0.09 ส่วนเจฮอตดรอว์ เวอร์ชั่น 1 และ 9 ที่จัดการเมท็อดคอลซ้ำกัน 4 และ 21 ตำแหน่งตามลำดับมีค่าเฉลี่ยมาตรวัด CDA ต่ำสุดเท่ากับ 0.00



รูปที่ 4.57 ค่าเฉลี่ยของมาตรวัด CDA ของเจฮอตดรอว์แต่ละเวอร์ชั่นที่จัดการจำนวนซ้ำเมท็อดคอลแตกต่างกัน

มาตรวัด CDA พิจารณาจากจำนวนคลาสทั้งหมดที่ได้รับผลกระทบจากพอยท์คัทและอินโทรดักชันที่กำหนดภายในแอสเป็ก ค่ามาตรวัด CDA จึงใช้สำหรับพิจารณาแอสเป็กโดยเฉพาะ ค่าเฉลี่ยมาตรวัด CDA ของเจฮอตดรอว์แต่ละเวอร์ชั่นจึงมีค่าขึ้นอยู่กับจำนวนคลาสที่แอสเป็กสามารถเข้าไปขัดขวางการทำงานเพื่อให้เมท็อดคอลทำงานในตำแหน่งเดิมของคลาส โดยหลังทำแอสเป็กรีแฟคทอริงพบว่าแอสเป็กของเจฮอตดรอว์ เวอร์ชั่น 12 สามารถเข้าไปขัดขวางการทำงานของคลาสทั้งหมด 44 คลาส ดังแสดงในตารางที่ 4.60 ข้างต้น จึงส่งผลให้ค่าเฉลี่ยมาตรวัด CDA มีค่าสูงสุดเมื่อเทียบกับเจฮอตดรอว์เวอร์ชั่นอื่นๆ ขณะที่แอสเป็กของเจฮอตดรอว์ เวอร์ชั่น 1 และ 9 สามารถเข้าไปขัดขวางการทำงานของคลาสได้เพียง 2 คลาส จึงส่งผลให้ค่าเฉลี่ยมาตรวัด CDA มีค่าต่ำสุด

(8) คัพพลิงออนแอดไวซ์เอ็กซิคิวชั่น (Coupling on Advice Execution - CAE)

จากตารางที่ 4.47 แสดงการเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นกับเจฮอตดรอว์เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้น โดยพิจารณาเฉพาะคลาส พบว่าหลังทำแอสเป็กรีแฟคทอริงเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นค่ามาตรวัด CAE ของคลาสมีค่าเปลี่ยนแปลง แต่จากตารางที่ 4.48 พบว่าค่ามาตรวัด CAE ของแอสเป็กในเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นมีค่ามาตรวัดเท่ากับ 0 จึงมีค่าต่ำกว่าค่าเฉลี่ยมาตรวัด CAE ที่คำนวณจากคลาสทั้ง 442 คลาส สามารถแสดงรายชื่อคลาสที่มีค่ามาตรวัด

CAE เปลี่ยนแปลง และค่ามาตรวัด CAE ก่อน-หลังทำแอสเป็กรีแฟคทอริงของเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นได้ดังนี้

- เจฮอตดรอว์ เวอร์ชั่น 1 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 4 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 2 คลาสที่มีค่ามาตรวัด CAE เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CAE แสดงดังตารางที่ 4.61

ตารางที่ 4.61 รายชื่อคลาส 2 คลาสของเจฮอตดรอว์ เวอร์ชั่น 1 ที่ค่ามาตรวัด CAE เปลี่ยนแปลง

คลาส	**มาตรวัด CAE**	
	ก่อนทำแอสเป็กรีแฟคทอริง	**หลังทำแอสเป็กรีแฟคทอริง**
org.jhotdraw.draw.DefaultDrawing	0	1
org.jhotdraw.draw.QuadTreeDrawing	0	1

- เจฮอตดรอว์ เวอร์ชั่น 2 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 5 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 5 คลาสที่มีค่ามาตรวัด CAE เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CAE แสดงดังตารางที่ 4.62

ตารางที่ 4.62 รายชื่อคลาส 5 คลาสของเจฮอตดรอว์ เวอร์ชั่น 2 ที่ค่ามาตรวัด CAE เปลี่ยนแปลง

คลาส	**มาตรวัด CAE**	
	ก่อนทำแอสเป็กรีแฟคทอริง	**หลังทำแอสเป็กรีแฟคทอริง**
org.jhotdraw.samples.draw.DrawingPanel	0	1
org.jhotdraw.samples.net.NetPanel	0	1
org.jhotdraw.samples.odg.ODGDrawingPanel	0	1
org.jhotdraw.samples.pert.PertPanel	0	1
org.jhotdraw.samples.svg.SVGDrawingPanel	0	1

- เจฮอตดรอว์ เวอร์ชั่น 3 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 9 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 9 คลาสที่มีค่ามาตรวัด CAE เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CAE แสดงดังตารางที่ 4.63

ตารางที่ 4.63 รายชื่อคลาส 9 คลาสของเจฮอตดรอว์ เวอร์ชั่น 3 ที่ค่ามาตรวัด CAE เปลี่ยนแปลง

คลาส	มาตรวัด CAE	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.draw.action.GroupAction	0	1
org.jhotdraw.draw.BezierTool	0	1
org.jhotdraw.draw.CreationTool	0	1
org.jhotdraw.samples.odg.action.CombineAction	0	1
org.jhotdraw.samples.odg.action.SplitAction	0	1
org.jhotdraw.samples.odg.PathTool	0	1
org.jhotdraw.samples.svg.action.CombineAction	0	1
org.jhotdraw.samples.svg.action.SplitAction	0	1
org.jhotdraw.samples.svg.PathTool	0	1

- เจฮอตดรอว์ เวอร์ชั่น 4 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 12 ตำแหน่งด้วยแอสเป็ก กรณีนี้จัดการเมท็อดคอลที่เรียกใช้งานเมท็อด ClearSelection() พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 11 คลาสที่มีค่ามาตรวัด CAE เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CAE แสดงดังตารางที่ 4.64

ตารางที่ 4.64 รายชื่อคลาส 11 คลาสของเจฮอตดรอว์ เวอร์ชั่น 4 ที่ค่ามาตรวัด CAE เปลี่ยนแปลง

คลาส	มาตรวัด CAE	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.draw.BezierTool	0	1
org.jhotdraw.draw.BidirectionalConnectionTool	0	1
org.jhotdraw.draw.ConnectionTool	0	1
org.jhotdraw.draw.CreationTool	0	1
org.jhotdraw.draw.DefaultDrawingView	0	1
org.jhotdraw.draw.DefaultDrawingViewTransferHandler	0	1
org.jhotdraw.draw.action.GroupAction	0	1
org.jhotdraw.samples.svg.action.CombineAction	0	1
org.jhotdraw.samples.svg.action.SplitAction	0	1
org.jhotdraw.samples.odg.action.CombineAction	0	1
org.jhotdraw.samples.odg.action.SplitAction	0	1

- เจฮอตดรอว์ เวอร์ชั่น 5 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 12 ตำแหน่งด้วยแอสเป็ก กรณีนี้จัดการเมท็อดคอลที่เรียกใช้งานเมท็อด RemoveAllChildren() พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 9 คลาสที่มีค่ามาตรวัด CAE เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CAE แสดงดังตารางที่ 4.65

ตารางที่ 4.65 รายชื่อคลาส 9 คลาสของเจฮอตดรอว์ เวอร์ชั่น 5 ที่ค่ามาตรวัด CAE เปลี่ยนแปลง

คลาส	**มาตรวัด CAE**	
	ก่อนทำแอสเป็กรีแฟคทอริง	**หลังทำแอสเป็กรีแฟคทอริง**
org.jhotdraw.samples.draw.DrawApplet	0	1
org.jhotdraw.samples.draw.DrawLiveConnectApplet	0	1
org.jhotdraw.samples.net.NetApplet	0	1
org.jhotdraw.samples.odg.PathTool	0	1
org.jhotdraw.samples.pert.PertApplet	0	1
org.jhotdraw.samples.svg.io.DefaultSVGFigureFactory	0	1
org.jhotdraw.samples.svg.PathTool	0	1
org.jhotdraw.samples.svg.SVGApplet	0	1
org.jhotdraw.samples.odg.io.ODGInputFormat	0	1

- เจฮอตดรอว์ เวอร์ชั่น 6 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 13 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 6 คลาสที่มีค่ามาตรวัด CAE เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CAE แสดงดังตารางที่ 4.66

ตารางที่ 4.66 รายชื่อคลาส 6 คลาสของเจฮอตดรอว์ เวอร์ชั่น 6 ที่ค่ามาตรวัด CAE เปลี่ยนแปลง

คลาส	**มาตรวัด CAE**	
	ก่อนทำแอสเป็กรีแฟคทอริง	**หลังทำแอสเป็กรีแฟคทอริง**
org.jhotdraw.app.DefaultApplicationModel	0	1
org.jhotdraw.draw.DefaultDrawing	0	1
org.jhotdraw.draw.DefaultDrawingEditor	0	1
org.jhotdraw.draw.GridConstrainer	0	1
org.jhotdraw.draw.QuadTreeDrawing	0	1
org.jhotdraw.samples.odg.ODGDrawing	0	1

- เจฮอตดรอว์ เวอร์ชั่น 7 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 15 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 10 คลาสที่มีค่ามาตรวัด CAE เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CAE แสดงดังตารางที่ 4.67

ตารางที่ 4.67 รายชื่อคลาส 10 คลาสของเจฮอตดรอว์ เวอร์ชั่น 7 ที่ค่ามาตรวัด CAE เปลี่ยนแปลง

คลาส	**มาตรวัด CAE**	
	ก่อนทำแอสเป็กรีแฟคทอริง	**หลังทำแอสเป็กรีแฟคทอริง**
org.jhotdraw.samples.draw.DrawingPanel	0	1
org.jhotdraw.samples.draw.DrawView	0	1
org.jhotdraw.samples.net.NetPanel	0	1
org.jhotdraw.samples.net.NetView	0	1
org.jhotdraw.samples.odg.ODGDrawingPanel	0	1
org.jhotdraw.samples.odg.ODGView	0	1
org.jhotdraw.samples.pert.PertPanel	0	1
org.jhotdraw.samples.pert.PertView	0	1
org.jhotdraw.samples.svg.SVGDrawingPanel	0	1
org.jhotdraw.samples.svg.SVGView	0	1

- เจฮอตดรอว์ เวอร์ชั่น 8 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 18 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 12 คลาสที่มีค่ามาตรวัด CAE เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CAE แสดงดังตารางที่ 4.68

ตารางที่ 4.68 รายชื่อคลาส 12 คลาสของเจฮอตดรอว์ เวอร์ชั่น 8 ที่ค่ามาตรวัด CAE เปลี่ยนแปลง

คลาส	**มาตรวัด CAE**	
	ก่อนทำแอสเป็กรีแฟคทอริง	**หลังทำแอสเป็กรีแฟคทอริง**
org.jhotdraw.draw.ChopDiamondConnector	0	1
org.jhotdraw.draw.ChopRoundRectangleConnector	0	1
org.jhotdraw.draw.EllipseFigure	0	1
org.jhotdraw.draw.ImageFigure	0	1
org.jhotdraw.draw.RectangleFigure	0	1
org.jhotdraw.draw.RoundRectangleFigure	0	1
org.jhotdraw.draw.TriangleFigure	0	1

ตารางที่ 4.68 รายชื่อคลาส 12 คลาสของเจฮอตดรอว์ เวอร์ชั่น 8
ที่ค่ามาตรวัด CAE เปลี่ยนแปลง (ต่อ)

คลาส	มาตรวัด CAE	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.samples.net.figures.NodeFigure	0	1
org.jhotdraw.samples.odg.figures.ODGPathFigure	0	1
org.jhotdraw.samples.svg.figures.SVGPathFigure	0	1
org.jhotdraw.samples.svg.figures.SVGTextAreaFigure	0	1
org.jhotdraw.samples.svg.figures.SVGTextFigure	0	1

- เจฮอตดรอว์ เวอร์ชั่น 9 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 21 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 2 คลาสที่มีค่ามาตรวัด CAE เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CAE แสดงดังตารางที่ 4.69

ตารางที่ 4.69 รายชื่อคลาส 2 คลาสของเจฮอตดรอว์ เวอร์ชั่น 9 ที่ค่ามาตรวัด CAE เปลี่ยนแปลง

คลาส	มาตรวัด CAE	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.samples.svg.io.SVGOutputFormat	0	1
org.jhotdraw.samples.svg.io.ImageMapOutputFormat	0	1

- เจฮอตดรอว์ เวอร์ชั่น 10 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 29 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 23 คลาสที่มีค่ามาตรวัด CAE เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CAE แสดงดังตารางที่ 4.70

ตารางที่ 4.70 รายชื่อคลาส 23 คลาสของเจฮอตดรอว์ เวอร์ชั่น 10 ที่ค่ามาตรวัด CAE เปลี่ยนแปลง

คลาส	มาตรวัด CAE	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.draw.ImageFigure	0	1
org.jhotdraw.draw.LabeledLineConnectionFigure	0	1
org.jhotdraw.draw.LineConnectionFigure	0	1
org.jhotdraw.samples.odg.figures.ODGPathFigure	0	1
org.jhotdraw.samples.pert.figures.TaskFigure	0	1
org.jhotdraw.samples.svg.figures.SVGImageFigure	0	1
org.jhotdraw.samples.svg.figures.SVGPathFigure	0	1
org.jhotdraw.draw.AttributeKey	0	1
org.jhotdraw.draw.BezierNodeHandle	0	1
org.jhotdraw.draw.BezierScaleHandle	0	1
org.jhotdraw.draw.BezierTool	0	1
org.jhotdraw.draw.ConnectionEndHandle	0	1
org.jhotdraw.draw.ConnectionStartHandle	0	1
org.jhotdraw.draw.ConnectionTool	0	1
org.jhotdraw.draw.CreationTool	0	1
org.jhotdraw.draw.FontSizeHandle	0	1
org.jhotdraw.draw.RoundRectangleRadiusHandle	0	1
org.jhotdraw.samples.odg.figures.ODGRectRadiusHandle	0	1
org.jhotdraw.samples.svg.figures.SVGRectRadiusHandle	0	1
org.jhotdraw.draw.DragHandle	0	1
org.jhotdraw.draw.MoveHandle	0	1
org.jhotdraw.draw.TextAreaTool	0	1
org.jhotdraw.draw.TextTool	0	1

- เจฮอตดรอว์ เวอร์ชั่น 11 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 37 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 26 คลาสที่มีค่ามาตรวัด CAE เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CAE แสดงดังตารางที่ 4.71

ตารางที่ 4.71 รายชื่อคลาส 26 คลาสของเจฮอตดรอว์ เวอร์ชั่น 11 ที่ค่ามาตรวัด CAE เปลี่ยนแปลง

คลาส	มาตรวัด CAE	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.draw.ImageFigure	0	1
org.jhotdraw.draw.LabeledLineConnectionFigure	0	1
org.jhotdraw.draw.LineConnectionFigure	0	1
org.jhotdraw.draw.LineFigure	0	1
org.jhotdraw.samples.odg.figures.ODGBezierFigure	0	1
org.jhotdraw.samples.odg.figures.ODGPathFigure	0	1
org.jhotdraw.samples.pert.figures.TaskFigure	0	1
org.jhotdraw.samples.svg.figures.SVGBezierFigure	0	1
org.jhotdraw.samples.svg.figures.SVGImageFigure	0	1
org.jhotdraw.samples.svg.figures.SVGPathFigure	0	1
org.jhotdraw.draw.AttributeKey	0	1
org.jhotdraw.draw.BezierNodeHandle	0	1
org.jhotdraw.draw.BezierTool	0	1
org.jhotdraw.draw.ConnectionTool	0	1
org.jhotdraw.draw.TextAreaTool	0	1
org.jhotdraw.draw.TextTool	0	1
org.jhotdraw.draw.BezierScaleHandle	0	1
org.jhotdraw.draw.ConnectionEndHandle	0	1
org.jhotdraw.draw.ConnectionStartHandle	0	1
org.jhotdraw.draw.ConnectorHandle	0	1
org.jhotdraw.draw.CreationTool	0	1
org.jhotdraw.draw.FontSizeHandle	0	1
org.jhotdraw.draw.RoundRectangleRadiusHandle	0	1
org.jhotdraw.draw.TextInputFormat	0	1
org.jhotdraw.samples.odg.figures.ODGRectRadiusHandle	0	1
org.jhotdraw.samples.svg.figures.SVGRectRadiusHandle	0	1

- เจฮอตดรอว์ เวอร์ชั่น 12 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 44 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 44 คลาสที่มีค่ามาตรวัด CAE เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด CAE แสดงดังตารางที่ 4.72

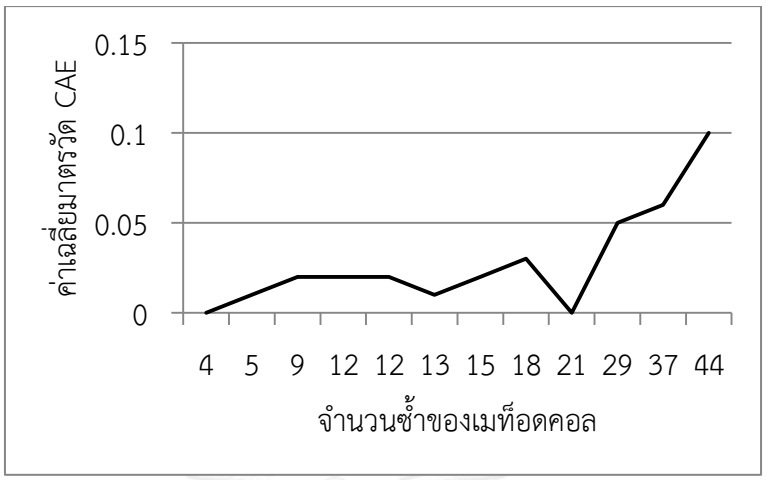
ตารางที่ 4.72 รายชื่อคลาส 44 คลาสของเจฮอตดรอว์ เวอร์ชั่น 12 ที่ค่ามาตรวัด CAE เปลี่ยนแปลง

คลาส	**มาตรวัด CAE**	
	ก่อนทำแอสเป็กรีแฟคทอริง	**หลังทำแอสเป็กรีแฟคทอริง**
org.jhotdraw.app.action.AboutAction	0	1
org.jhotdraw.app.action.ClearAction	0	1
org.jhotdraw.app.action.ClearRecentFilesAction	0	1
org.jhotdraw.app.action.CloseAction	0	1
org.jhotdraw.app.action.CopyAction	0	1
org.jhotdraw.app.action.CutAction	0	1
org.jhotdraw.app.action.DeleteAction	0	1
org.jhotdraw.app.action.DuplicateAction	0	1
org.jhotdraw.app.action.ExitAction	0	1
org.jhotdraw.app.action.ExportAction	0	1
org.jhotdraw.app.action.FindAction	0	1
org.jhotdraw.app.action.FocusAction	0	1
org.jhotdraw.app.action.LoadAction	0	1
org.jhotdraw.app.action.MaximizeAction	0	1
org.jhotdraw.app.action.MinimizeAction	0	1
org.jhotdraw.app.action.NewAction	0	1
org.jhotdraw.app.action.OpenAction	0	1
org.jhotdraw.app.action.PasteAction	0	1
org.jhotdraw.app.action.PrintAction	0	1
org.jhotdraw.app.action.RedoAction	0	1
org.jhotdraw.app.action.SaveAction	0	1
org.jhotdraw.app.action.SaveAsAction	0	1
org.jhotdraw.app.action.SelectAllAction	0	1
org.jhotdraw.app.action.UndoAction	0	1

ตารางที่ 4.72 รายชื่อคลาส 44 คลาสของเจฮอตดรอว์ เวอร์ชั่น 12
ที่ค่ามาตรวัด CAE เปลี่ยนแปลง (ต่อ)

คลาส	มาตรวัด CAE	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.draw.action.ApplyAttributesAction	0	1
org.jhotdraw.draw.action.EditDrawingAction	0	1
org.jhotdraw.draw.action.EditGridAction	0	1
org.jhotdraw.draw.action.MoveToBackAction	0	1
org.jhotdraw.draw.action.MoveToFrontAction	0	1
org.jhotdraw.draw.action.PickAttributesAction	0	1
org.jhotdraw.draw.action.ToggleGridAction	0	1
org.jhotdraw.samples.odg.action.CombineAction	0	1
org.jhotdraw.samples.odg.action.SplitAction	0	1
org.jhotdraw.samples.svg.action.CombineAction	0	1
org.jhotdraw.samples.svg.action.SplitAction	0	1
org.jhotdraw.samples.svg.action.ViewSourceAction	0	1
org.jhotdraw.samples.teddy.action.FindAction	0	1
org.jhotdraw.samples.teddy.action.ToggleLineNumbersAction	0	1
org.jhotdraw.samples.teddy.action.ToggleLineWrapAction	0	1
org.jhotdraw.samples.teddy.action.ToggleStatusBarAction	0	1
org.jhotdraw.draw.action.AbstractSelectedAction	0	1
org.jhotdraw.draw.action.GroupAction	0	1
org.jhotdraw.draw.action.UngroupAction	0	1
org.jhotdraw.draw.action.SelectSameAction	0	1

การเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น โดยนำค่าเฉลี่ยของมาตรวัด CAE จากข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์แต่ละเวอร์ชั่น (ตารางที่ 4.19 - 4.30) มาเปรียบเทียบ แสดงด้วยกราฟได้ดังรูปที่ 4.58 พบว่าลักษณะกราฟค่าเฉลี่ยมาตรวัด CAE มีลักษณะเหมือนกับกราฟค่าเฉลี่ยมาตรวัด CDA ของเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นดังรูปที่ 4.57 ข้างต้น โดยเจฮอตดรอว์ เวอร์ชั่น 12 ที่จัดการเมท็อดคอลซ้ำกันทั้งหมด 44 ตำแหน่งมีค่าเฉลี่ยมาตรวัด CAE สูงสุดเท่ากับ 0.09 และเจฮอตดรอว์ เวอร์ชั่น 1 และ 9 ที่จัดการเมท็อดคอลซ้ำกันทั้งหมด 4 และ 21 ตำแหน่งตามลำดับ มีค่าเฉลี่ยมาตรวัด CAE ต่ำสุดเท่ากับ 0.00 เช่นกัน



รูปที่ 4.58 ค่าเฉลี่ยของมาตรวัด CAE ของเจฮอตดรอว์แต่ละเวอร์ชั่น ที่จัดการจำนวนซ้ำเมท็อดคอลแตกต่างกัน

มาตรวัด CAE ใช้สำหรับพิจารณาคลาสโดยเฉพาะ โดยนับจำนวนแอสเป็กทั้งหมดที่เข้ามาขัดขวางการทำงานของคลาสที่พิจารณา และเนื่องจากการทำแอสเป็กรีแฟคทอริงของเจฮอตดรอว์แต่ละเวอร์ชั่นได้สร้างแอสเป็กเพียง 1 แอสเป็ก ดังนั้นจำนวนแอสเป็กที่สามารถขัดขวางการทำงานของคลาสจึงมีเพียง 1 แอสเป็ก ทำให้ค่ามาตรวัด CAE ของคลาสที่แอสเป็กเข้ามาขัดขวางการทำงานมีค่ามาตรวัดเท่ากับ 1 ดังตารางที่ 4.61 – 4.72 ข้างต้น จากนิยามมาตรวัด CAE จึงแสดงให้เห็นว่าในเจฮอตดรอว์เวอร์ชั่นหนึ่งๆ จะมีผลรวมค่ามาตรวัด CAE ของคลาสทั้งหมดเท่ากับค่ามาตรวัด CDA ของแอสเป็ก เนื่องจากมาตรวัด CDA วัดแอสเป็กโดยนับจากจำนวนคลาสทั้งหมดที่แอสเป็กสามารถเข้าไปขัดขวางการทำงาน จากนิยามจึงส่งผลให้กราฟค่าเฉลี่ยของเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นระหว่างมาตรวัด CDA และ CAE มีลักษณะของกราฟเหมือนกัน ค่าเฉลี่ยของมาตรวัด CAE จึงขึ้นอยู่กับจำนวนคลาสที่แอสเป็กสามารถเข้าไปขัดขวางการทำงานด้วยเช่นกัน จากผลค่าเฉลี่ยมาตรวัด CAE ของเจฮอตดรอว์ เวอร์ชั่น 12 มีคลาสทั้งหมด 44 คลาสที่แอสเป็กสามารถเข้าไปขัดขวางการทำงาน ทำให้ค่าเฉลี่ยของมาตรวัด CAE มีค่าสูงสุดเท่ากับ 0.09 ขณะที่เจฮอตดรอว์ เวอร์ชั่น 1 และ 9 มีคลาสเพียง 2 คลาสที่แอสเป็กสามารถเข้าไปขัดขวางการทำงาน ค่าเฉลี่ยของมาตรวัด CAE จึงมีค่าต่ำสุดเท่ากับ 0.00

(9) เรสปอนซ์ฟอร์อะมอดูล (Response For a Module - RFM)

จากตารางที่ 4.47 แสดงการเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นกับเจฮอตดรอว์เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้น โดยพิจารณาเฉพาะคลาส พบว่าหลังทำแอสเป็กรีแฟคทอริงมีเจฮอตดรอว์ทั้งหมด 10 เวอร์ชั่นที่ค่ามาตรวัด RFM ของคลาสเปลี่ยนแปลง ยกเว้นเจฮอตดรอว์เวอร์ชั่น 2 และ 12 ที่ค่ามาตรวัด RFM ของคลาสมีค่าไม่เปลี่ยนแปลง และจากตารางที่ 4.48 พบว่าค่ามาตรวัด RFM ของแอสเป็กในเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นมีค่าเท่ากับ 0 จึงต่ำกว่าค่าเฉลี่ยมาตรวัด RFM ที่คำนวณจากคลาสทั้ง 442 คลาส ดังนั้นจึงสามารถแสดงรายชื่อคลาสที่มีค่ามาตรวัด RFM เปลี่ยนแปลง และค่ามาตรวัด RFM ก่อน-หลังทำแอสเป็กรีแฟคทอริงของเจฮอตดรอว์ เวอร์ชั่น 1 และ 3-11ได้ดังนี้

- เจฮอตดรอว์ เวอร์ชั่น 1 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 4 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 1 คลาสที่มีค่ามาตรวัด RFM เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFM แสดงดังตารางที่ 4.73

ตารางที่ 4.73 รายชื่อคลาส 1 คลาสของเจฮอตดรอว์ เวอร์ชั่น 1 ที่ค่ามาตรวัด RFM เปลี่ยนแปลง

คลาส	**มาตรวัด RFM**	
	ก่อนทำแอสเป็กรีแฟคทอริง	**หลังทำแอสเป็กรีแฟคทอริง**
org.jhotdraw.draw.QuadTreeDrawing	51	52

- เจฮอตดรอว์ เวอร์ชั่น 3 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 9 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 9 คลาสที่มีค่ามาตรวัด RFM เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFM แสดงดังตารางที่ 4.74

ตารางที่ 4.74 รายชื่อคลาส 9 คลาสของเจฮอตดรอว์ เวอร์ชั่น 3 ที่ค่ามาตรวัด RFM เปลี่ยนแปลง

คลาส	**มาตรวัด RFM**	
	ก่อนทำแอสเป็กรีแฟคทอริง	**หลังทำแอสเป็กรีแฟคทอริง**
org.jhotdraw.draw.action.GroupAction	33	34
org.jhotdraw.draw.BezierTool	52	55
org.jhotdraw.draw.CreationTool	39	41
org.jhotdraw.samples.odg.action.CombineAction	27	28
org.jhotdraw.samples.odg.action.SplitAction	27	28
org.jhotdraw.samples.odg.PathTool	14	16
org.jhotdraw.samples.svg.action.CombineAction	29	30
org.jhotdraw.samples.svg.action.SplitAction	27	28
org.jhotdraw.samples.svg.PathTool	14	16

- เจฮอตดรอว์ เวอร์ชั่น 4 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 12 ตำแหน่งด้วยแอสเป็ก กรณีนี้เป็นการจัดการเมท็อดคอลที่เรียกใช้งานเมท็อด ClearSelection() พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 11 คลาสที่มีค่ามาตรวัด RFM เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFM แสดงดังตารางที่ 4.75

ตารางที่ 4.75 รายชื่อคลาส 11 คลาสของเจฮอตดรอว์ เวอร์ชั่น 4 ที่ค่ามาตรวัด RFM เปลี่ยนแปลง

คลาส	มาตรวัด RFM	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.draw.BezierTool	52	54
org.jhotdraw.draw.BidirectionalConnectionTool	68	69
org.jhotdraw.draw.ConnectionTool	58	59
org.jhotdraw.draw.CreationTool	39	40
org.jhotdraw.draw.DefaultDrawingView	145	146
org.jhotdraw.draw.DefaultDrawingViewTransferHandler	49	51
org.jhotdraw.draw.action.GroupAction	33	35
org.jhotdraw.samples.svg.action.CombineAction	29	31
org.jhotdraw.samples.svg.action.SplitAction	27	29
org.jhotdraw.samples.odg.action.CombineAction	27	29
org.jhotdraw.samples.odg.action.SplitAction	27	29

- เจฮอตดรอว์ เวอร์ชั่น 5 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 12 ตำแหน่งด้วยแอสเป็ก กรณีนี้เป็นการจัดการเมท็อดคอลที่เรียกใช้งานเมท็อด RemoveAllChildren() พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 1 คลาสที่มีค่ามาตรวัด RFM เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFM แสดงดังตารางที่ 4.76

ตารางที่ 4.76 รายชื่อคลาส 1 คลาสของเจฮอตดรอว์ เวอร์ชั่น 5 ที่ค่ามาตรวัด RFM เปลี่ยนแปลง

คลาส	มาตรวัด RFM	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.samples.odg.io.ODGInputFormat	96	97

- เจฮอตดรอว์ เวอร์ชั่น 6 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 13 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 6 คลาสที่มีค่ามาตรวัด RFM เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFM แสดงดังตารางที่ 4.77

ตารางที่ 4.77 รายชื่อคลาส 6 คลาสของเจฮอตดรอว์ เวอร์ชั่น 6 ที่ค่ามาตรวัด RFM เปลี่ยนแปลง

คลาส	มาตรวัด RFM	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.app.DefaultApplicationModel	21	23
org.jhotdraw.draw.DefaultDrawing	38	40
org.jhotdraw.draw.DefaultDrawingEditor	40	43
org.jhotdraw.draw.GridConstrainer	27	29
org.jhotdraw.draw.QuadTreeDrawing	51	53
org.jhotdraw.samples.odg.ODGDrawing	23	25

- เจฮอตดรอว์ เวอร์ชั่น 7 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 15 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 10 คลาสที่มีค่ามาตรวัด RFM เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFM แสดงดังตารางที่ 4.78

ตารางที่ 4.78 รายชื่อคลาส 10 คลาสของเจฮอตดรอว์ เวอร์ชั่น 7 ที่ค่ามาตรวัด RFM เปลี่ยนแปลง

คลาส	มาตรวัด RFM	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.samples.draw.DrawingPanel	21	22
org.jhotdraw.samples.draw.DrawView	49	52
org.jhotdraw.samples.net.NetPanel	14	15
org.jhotdraw.samples.net.NetView	55	58
org.jhotdraw.samples.odg.ODGDrawingPanel	17	18
org.jhotdraw.samples.odg.ODGView	71	74
org.jhotdraw.samples.pert.PertPanel	14	15
org.jhotdraw.samples.pert.PertView	54	57
org.jhotdraw.samples.svg.SVGDrawingPanel	17	18
org.jhotdraw.samples.svg.SVGView	72	75

- เจฮอตดรอว์ เวอร์ชั่น 8 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 18 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 12 คลาสที่มีค่ามาตรวัด RFM เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFM แสดงดังตารางที่ 4.79

ตารางที่ 4.79 รายชื่อคลาส 12 คลาสของเจฮอตดรอว์ เวอร์ชั่น 8 ที่ค่ามาตรวัด RFM เปลี่ยนแปลง

คลาส	มาตรวัด RFM	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.draw.ChopDiamondConnector	10	13
org.jhotdraw.draw.ChopRoundRectangleConnector	10	13
org.jhotdraw.draw.EllipseFigure	19	22
org.jhotdraw.draw.ImageFigure	51	54
org.jhotdraw.draw.RectangleFigure	17	20
org.jhotdraw.draw.RoundRectangleFigure	30	33
org.jhotdraw.draw.TriangleFigure	34	37
org.jhotdraw.samples.net.figures.NodeFigure	28	29
org.jhotdraw.samples.odg.figures.ODGPathFigure	92	95
org.jhotdraw.samples.svg.figures.SVGPathFigure	92	95
org.jhotdraw.samples.svg.figures.SVGTextAreaFigure	63	66
org.jhotdraw.samples.svg.figures.SVGTextFigure	69	72

- เจฮอตดรอว์ เวอร์ชั่น 9 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 21 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 2 คลาสที่มีค่ามาตรวัด RFM เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFM แสดงดังตารางที่ 4.80

ตารางที่ 4.80 รายชื่อคลาส 2 คลาสของเจฮอตดรอว์ เวอร์ชั่น 9 ที่ค่ามาตรวัด RFM เปลี่ยนแปลง

คลาส	มาตรวัด RFM	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.samples.svg.io.SVGOutputFormat	120	121
org.jhotdraw.samples.svg.io.ImageMapOutputFormat	61	62

- เจฮอตดรอว์ เวอร์ชั่น 10 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 29 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 14 คลาสที่มีค่ามาตรวัด RFM เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFM แสดงดังตารางที่ 4.81

ตารางที่ 4.81 รายชื่อคลาส 14 คลาสของเจฮอตดรอว์ เวอร์ชั่น 10 ที่ค่ามาตรวัด RFM เปลี่ยนแปลง

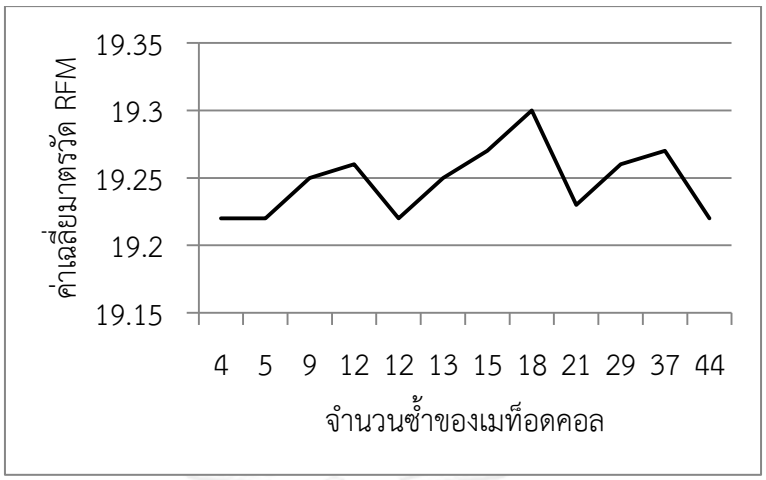
คลาส	มาตรวัด RFM	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.draw.LabeledLineConnectionFigure	79	80
org.jhotdraw.draw.LineConnectionFigure	81	82
org.jhotdraw.samples.odg.figures.ODGPathFigure	92	93
org.jhotdraw.samples.svg.figures.SVGPathFigure	92	93
org.jhotdraw.draw.AttributeKey	26	27
org.jhotdraw.draw.BezierNodeHandle	55	56
org.jhotdraw.draw.BezierTool	52	54
org.jhotdraw.draw.ConnectionTool	58	60
org.jhotdraw.draw.CreationTool	39	41
org.jhotdraw.draw.FontSizeHandle	26	27
org.jhotdraw.samples.odg.figures.ODGRectRadiusHandle	28	29
org.jhotdraw.samples.svg.figures.SVGRectRadiusHandle	28	29
org.jhotdraw.draw.DragHandle	22	23
org.jhotdraw.draw.TextTool	37	38

- เจฮอตดรอว์ เวอร์ชั่น 11 ปรับปรุงซอร์สโค้ดโดยจัดการเมท็อดคอลซ้ำกันทั้งหมด 37 ตำแหน่งด้วยแอสเป็ก พบว่าหลังทำแอสเป็กรีแฟคทอริงมีคลาส 16 คลาสที่มีค่ามาตรวัด RFM เปลี่ยนแปลง รายชื่อคลาสและค่ามาตรวัด RFM แสดงดังตารางที่ 4.82

ตารางที่ 4.82 รายชื่อคลาส 16 คลาสของเจฮอตดรอว์ เวอร์ชั่น 11 ที่ค่ามาตรวัด RFM เปลี่ยนแปลง

คลาส	มาตรวัด RFM	
	ก่อนทำแอสเป็กรีแฟคทอริง	หลังทำแอสเป็กรีแฟคทอริง
org.jhotdraw.draw.LabeledLineConnectionFigure	79	80
org.jhotdraw.draw.LineConnectionFigure	81	82
org.jhotdraw.draw.LineFigure	14	15
org.jhotdraw.samples.odg.figures.ODGBezierFigure	26	27
org.jhotdraw.samples.odg.figures.ODGPathFigure	92	93
org.jhotdraw.samples.svg.figures.SVGBezierFigure	30	31
org.jhotdraw.samples.svg.figures.SVGPathFigure	92	93
org.jhotdraw.draw.AttributeKey	26	28
org.jhotdraw.draw.BezierNodeHandle	55	57
org.jhotdraw.draw.BezierTool	52	54
org.jhotdraw.draw.ConnectionTool	58	60
org.jhotdraw.draw.CreationTool	39	41
org.jhotdraw.draw.FontSizeHandle	26	27
org.jhotdraw.draw.TextInputFormat	16	17
org.jhotdraw.samples.odg.figures.ODGRectRadiusHandle	28	29
org.jhotdraw.samples.svg.figures.SVGRectRadiusHandle	28	29

การเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น โดยนำค่าเฉลี่ยของมาตรวัด RFM จากข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์แต่ละเวอร์ชั่น (ตารางที่ 4.19 – 4.30) มาเปรียบเทียบ แสดงด้วยกราฟดังรูปที่ 4.59 พบว่าการทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลในจำนวนซ้ำที่แตกต่างกันไม่แสดงแนวโน้มของค่าเฉลี่ยมาตรวัด RFM โดยเจฮอตดรอว์ เวอร์ชั่น 1, 2, 5 และ 12 ที่จัดการเมท็อดคอลซ้ำกัน 4, 5, 12 และ 44 ตำแหน่งตามลำดับ มีค่าเฉลี่ยมาตรวัด RFM ต่ำสุดเท่ากับ 19.22 ส่วนเจฮอตดรอว์ เวอร์ชั่น 8 ที่จัดการเมท็อดคอลซ้ำกัน 18 ตำแหน่งมีค่าเฉลี่ยมาตรวัด RFM สูงสุดเท่ากับ 19.30



รูปที่ 4.59 ค่าเฉลี่ยของมาตรวัด RFM ของเจฮอตดรอว์แต่ละเวอร์ชั่น ที่จัดการจำนวนซ้ำเมท็อดคอลแตกต่างกัน

นิยามมาตรวัด RFM คล้ายกับนิยามมาตรวัด RFC ของมาตรวัดเชิงวัตถุตรงที่พิจารณาคลาสจากจำนวนเมท็อดทั้งหมดที่ทำงานเพื่อตอบสนองต่อคลาสเหมือนกัน แต่แตกต่างกันที่มาตรวัด RFM มีนิยามเกี่ยวกับแอสเป็กเพิ่มเติม โดยเพิ่มการนับจำนวนแอดไวซ์ที่ทำงานเพิ่มเติมหรือแทนที่การทำงานของคลาส ดังนั้นมาตรวัด RFM จึงพิจารณาคลาสจากการนับทั้งจำนวนเมท็อดและแอดไวซ์ทั้งหมดที่ทำงานเพื่อตอบสนองต่อคลาส โดยหลังทำแอสเป็กรีแฟคทอริงพบว่าค่ามาตรวัด RFM ของคลาสส่วนใหญ่มีค่ามาตรวัดเพิ่มขึ้น เนื่องจากก่อนทำแอสเป็กรีแฟคทอริงภายในเจฮอตดรอว์ทุกเวอร์ชั่นยังไม่มีแอสเป็ก ค่ามาตรวัด RFM ของคลาสจึงพิจารณาแค่จำนวนเมท็อดที่ถูกเรียกใช้งานจากภายในคลาส แต่หลังทำแอสเป็กรีแฟคทอริงภายในเจฮอตดรอว์มีการสร้างแอสเป็กจึงทำให้ค่ามาตรวัด RFM ของคลาสเพิ่มขึ้นจากจำนวนแอดไวซ์ภายในแอสเป็กที่เข้ามาขัดขวางการทำงานของคลาส ดังนั้นค่ามาตรวัด RFM ของคลาสจะมีค่าเพิ่มขึ้นมากหรือน้อยขึ้นอยู่กับจำนวนแอดไวซ์ที่เข้ามาขัดขวางการทำงานของคลาส

ตัวอย่างการพิจารณาค่ามาตรวัด RFM ของคลาส ChopRoundRectangleConnector ในเจฮอตดรอว์ เวอร์ชั่น 8 เป็นการจัดการเมท็อดคอลที่เรียกใช้งานเมท็อด grow() ของคลาส Geom ด้วยแอสเป็ก จากรูปที่ 4.60 ในคลาสมีการเรียกใช้งานเมท็อด grow() ทั้งหมด 1 ตำแหน่ง โดยก่อนทำแอสเป็กรีแฟคทอริง ค่ามาตรวัด RFM ของคลาส ChopRoundRectangleConnector พิจารณาเพียงจำนวนเมท็อดที่ทำงานเพื่อตอบสนองต่อคลาส ซึ่งมีเมท็อดทั้งหมด 10 เมท็อด ค่ามาตรวัด RFM ของคลาสจึงมีค่าเท่ากับ 10 สามารถแสดงรายชื่อเมท็อดทั้ง 10 เมท็อดได้ดังตารางที่ 4.83 และแสดงตำแหน่งของทั้ง 10 เมท็อดคอลภายในคลาส ChopRoundRectangleConnector ได้ดังรูปที่ 4.60 โดยมาตรวัด RFM ไม่พิจารณาการเรียกใช้งานเมท็อดของคลาสซึ่งเป็นคลาสที่อยู่ภายในไลบรารี่ของจาวาดังเช่นเมท็อด min() และ max() ของคลาส Math เป็นต้น

```
public class ChopRoundRectangleConnector extends ChopRectangleConnector {
    public Rectangle2D.Double outer;
    public double grow;
    public Rectangle2D.Double inner;
    public ChopRoundRectangleConnector(Figure owner) { super(owner); }
    protected Point2D.Double chop(Figure target, Point2D.Double from) {
        target =  getConnectorTarget(target);
        RoundRectangleFigure rrf = (RoundRectangleFigure) target;
        outer = rrf.getBounds();
        switch (STROKE_PLACEMENT.get(target)) {
            case CENTER :
            default :
                grow = AttributeKeys.getStrokeTotalWidth(target) / 2d;
                break;
            case OUTSIDE :
                grow = AttributeKeys.getStrokeTotalWidth(target);
                break;
            case INSIDE :
                grow = 0;
                break;
        }
        Geom.grow(outer, grow, grow);
        inner = (Rectangle2D.Double) outer.clone();
        double gw = -(rrf.getArcWidth() + grow * 2) / 2;
        double gh = -(rrf.getArcHeight() + grow *2) / 2;
        inner.x -= gw;
        inner.y -= gh;
        inner.width += gw * 2;
        inner.height += gh * 2;
        double angle = Geom.pointToAngle(outer, from);
        Point2D.Double p = Geom.angleToPoint(outer, Geom.pointToAngle(outer, from));
        ...
    }
}
```

รูปที่ 4.60 คลาส ChopRoundRectangleConnector และตำแหน่งเมท็อดคอลทั้งหมด

ตารางที่ 4.83 รายชื่อเมท็อดที่ทำงานตอบสนองต่อคลาส ChopRoundRectangleConnector

ลำดับ	คลาส	เมท็อด
1	AbstractConnector	getConnectorTarget(..)
2	RoundRectangleFigure	getBounds()
3	AttributeKey	get(Figure f)
4	AttributeKeys	getStrokeTotalWidth(..)
5	Geom	grow(..)
6	RectangularShape	clone()
7	RoundRectangleFigure	getArcWidth()
8	RoundRectangleFigure	getArcHeight()
9	Geom	pointToAngle(..)
10	Geom	angleToPoint(..)

หลังทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลที่เรียกใช้งานเมท็อด grow() ด้วยแอสเป็ก แสดงแอสเป็กดังรูปที่ 4.61 การพิจารณาค่ามาตรวัด RFM ของคลาส ChopRoundRectangle Connector หลังทำแอสเป็กรีแฟคทอริงจึงต้องพิจารณาทั้งจำนวนเมท็อดและจำนวนแอดไวซ์ที่ทำงานเพื่อตอบสนองคลาส เนื่องจากเดิมมีเมท็อดทั้งหมด 10 เมท็อดที่ทำงานตอบสนองต่อคลาส แต่หลังจัดการเมท็อดคอลที่เรียกใช้งานเมท็อด grow() ออกจากคลาสและนำไปไว้ในแอสเป็กแทน จึงทำให้จำนวนเมท็อดที่ทำงานตอบสนองต่อคลาสเหลือ 9 เมท็อด และเมื่อพิจารณาจำนวนแอดไวซ์ภายในแอสเป็กที่เข้ามาขัดขวางการทำงานของคลาส ChopRoundRectangleConnector โดยมีแอดไวซ์ทั้งหมด 4 แอดไวซ์ ทำให้ผลรวมค่ามาตรวัด RFM ของคลาสมีค่าเท่ากับ 13 ส่งผลให้หลังทำแอสเป็กรีแฟคทอริงค่ามาตรวัด RFM มีค่าเพิ่มขึ้น

```
public aspect  AspectGrow {
    private Rectangle2D.Double tmpR;
    private double tmpH;
    private double tmpV;
    pointcut collectR() : set(public * ChopRoundRectangleConnector.outer);
    after() : collectR(){
        Object tmp[] = thisJoinPoint.getArgs();
        tmpR = (Rectangle2D.Double)tmp[0];
    }
    pointcut collectH() : set(public * ChopRoundRectangleConnector.grow);
    after() : collectH(){
        Object tmp[] = thisJoinPoint.getArgs();
        tmpH = (Double)tmp[0];
    }
    pointcut tmpp3() : set(public * ChopRoundRectangleConnector.grow);
    after() : collectV(){
        Object tmp[] = thisJoinPoint.getArgs();
        tmpV = (Double)tmp[0];
    }
    pointcut growAction() : set(public * ChopRoundRectangleConnector.inner);
    before() : growAction(){
        Geom.grow(tmpR,tmpH,tmpV);
    }
}
```

รูปที่ 4.61 แอสเป็กสำหรับจัดการเมท็อดคอลภายในคลาส ChopRoundRectangleConnector

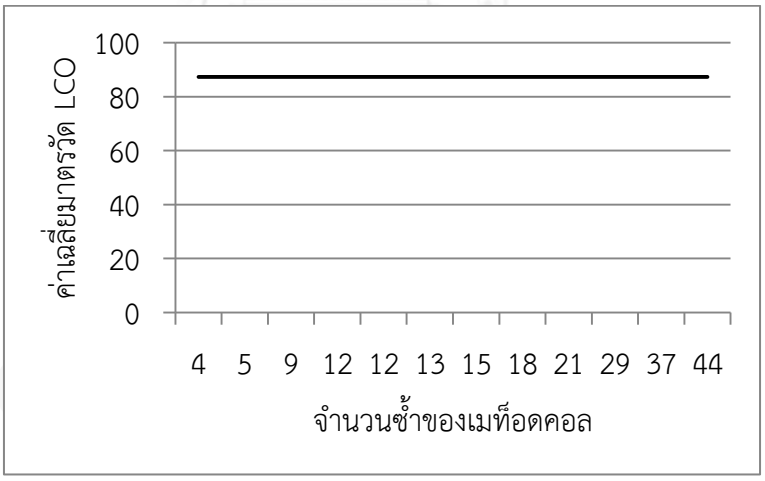
จากการเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น พบว่าหลังทำแอสเป็กรีแฟคทอริงเจฮอตดรอว์ เวอร์ชั่น 1, 2, 5 และ 12 มีค่าเฉลี่ยมาตรวัด RFM ต่ำสุด เนื่องจากเจฮอตดรอว์ เวอร์ชั่น 2 และ 12 ค่ามาตรวัด RFM ของคลาสมีค่าไม่เปลี่ยนแปลง ส่วนเจฮอตดรอว์ เวอร์ชั่น 1 และ 5 มีเพียงหนึ่งคลาสที่มีค่ามาตรวัด RFM เปลี่ยนแปลง เมื่อคำนวณค่าเฉลี่ยมาตรวัดจากจำนวนคลาสและแอสเป็กทั้งหมดจึงทำให้ค่าเฉลี่ยมาตรวัด RFM มีค่าต่ำสุด แต่สำหรับเจฮอตดรอว์ เวอร์ชั่น 8 ที่มีค่าเฉลี่ยมาตรวัด RFM สูงสุด เนื่องจากภายในแอสเป็กมีการใช้แอดไวซ์ถึงสามแอดไวซ์เพื่อเก็บค่าตัวแปรอ็อบเจกต์ที่ต้องใช้ในเมท็อดคอล และหนึ่งแอดไวซ์สำหรับกำหนดให้เมท็อดคอลทำงานในตำแหน่งเดิมของคลาส ทำให้มีจำนวนแอดไวซ์ที่เข้าไปขัดขวางการทำงานของคลาสจำนวนมากเมื่อเทียบกับเจฮอตดรอว์เวอร์ชั่นอื่นๆ ส่งผลให้ค่าเฉลี่ยมาตรวัด RFM มีค่าสูงตามเช่นกัน

จากตารางที่ 2.4 ความสัมพันธ์ระหว่างมาตรวัดเชิงแอสเป็กกับคุณภาพซอฟต์แวร์ ที่อ้างอิงจากงานวิจัยของ Sirbi และ Kulkarni (2013) โดยมาตรวัด RFM สามารถส่งผลกระทบต่อคุณภาพซอฟต์แวร์ 2 ด้าน ได้แก่ ความสามารถในการทำความเข้าใจและความสามารถการทดสอบ เนื่องจากผลการทำแอสเป็กรีแฟคทอริงส่วนใหญ่แสดงค่ามาตรวัด RFM ของคลาสมีค่าเพิ่มขึ้นจากจำนวนแอดไวซ์ของแอสเป็ก จึงส่งผลต่อคุณภาพซอฟต์แวร์ทั้งสองด้านทำให้การทำความเข้าใจและการทดสอบซอฟต์แวร์ทำได้ยากขึ้น

(10) แลคออฟโคฮีชันอินโอเปอร์เรชั่น (Lack of Cohesion in Operations - LCO)

จากตารางที่ 4.47 แสดงการเปรียบเทียบเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นกับเจฮอตดรอว์ เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ตั้งต้น โดยพิจารณาเฉพาะคลาส พบว่าหลังทำแอสเป็กรีแฟคทอริงค่ามาตรวัด LCO ของคลาสทั้งหมดในเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นมีค่าไม่เปลี่ยนแปลง และจากตารางที่ 4.48 พบว่าค่ามาตรวัด LCO ของแอสเป็กในเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นมีค่าเท่ากับ 0 จึงมีค่าต่ำกว่าค่าเฉลี่ยมาตรวัด LCO ที่คำนวณจากคลาสทั้ง 442 คลาส

ผลการเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นจึงให้ผลเช่นเดียวกัน โดยนำค่าเฉลี่ยของมาตรวัด LCO จากข้อมูลสถิติเชิงพรรณนาของเจฮอตดรอว์แต่ละเวอร์ชั่น (ตารางที่ 4.19 – 4.30) มาเปรียบเทียบ แสดงด้วยกราฟดังรูปที่ 4.62 พบว่าการทำแอสเป็กรีแฟคทอริงจัดการเมท็อดคอลในจำนวนซ้ำของเมท็อดคอลที่แตกต่างกันไม่ส่งผลให้ค่ามาตรวัด LCO เกิดการเปลี่ยนแปลง โดยค่าเฉลี่ยของมาตรวัด LCO มีค่าคงเดิมเท่ากับ 87.26



รูปที่ 4.62 ค่าเฉลี่ยของมาตรวัด LCO ของเจฮอตดรอว์แต่ละเวอร์ชั่นที่จัดการจำนวนซ้ำเมท็อดคอลแตกต่างกัน

มาตรวัด LCO พิจารณาจำนวนคู่ของโอเปอเรชั่นที่เข้าถึงฟิลด์ของคลาสและแอสเป็กแตกต่างกันหักลบกับจำนวนคู่ของโอเปอเรชั่นที่เข้าถึงฟิลด์เดียวกัน หากโอเปอเรชั่นส่วนใหญ่เข้าถึงฟิลด์เดียวกันแสดงว่าโอเปอเรชั่นทั้งหมดในคลาสมีความเกี่ยวข้องกันมาก โดยนิยามของมาตรวัด LCO เหมือนกันกับนิยามของมาตรวัด LCOM ของมาตรวัดเชิงวัตถุ ดังนั้นแนวโน้มของกราฟค่าเฉลี่ยของมาตรวัด LCO และมาตรวัด LCOM จากการเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นจึงคล้ายกัน คือค่าเฉลี่ยของเวอร์ชั่นเจฮอตดรอว์ส่วนใหญ่มีค่าเฉลี่ยของมาตรวัดคงที่ เพราะหลังทำแอสเป็กรีแฟคทอริงไม่มีคลาสที่มีค่ามาตรวัดเปลี่ยนแปลง เนื่องจากการจัดการเมท็อดคอลด้วยแอสเป็ก แม้เมท็อดคอลที่จัดการมีการเข้าถึงฟิลด์คลาส แต่หลังจัดการเมท็อดคอลออกจากคลาสและตรวจสอบภายในคลาสพบว่ายังคงมีตำแหน่งอื่นของเมท็อดที่เข้าถึงฟิลด์เดียวกัน การจัดการเมท็อดคอลด้วยแอสเป็กจึงไม่ส่งผลให้การเข้าถึงฟิลด์คลาสระหว่างเมท็อดเปลี่ยนแปลง ค่ามาตรวัด LCO ของคลาสจึงมีค่าคงเดิม ดังรูปที่ 4.63 แสดงตัวอย่างคลาส DefaultDrawing ที่ภายในเมท็อด bringToFront มีการเรียกใช้งานเมท็อด fireAreaInvalidated() ที่ต้องการจัดการด้วยแอสเป็ก โดยเมท็อดคอลมีการเข้าถึงฟิลด์ของคลาสคือฟิลด์ drawArea แต่หลังจากการทำแอสเป็กรีแฟคทอริง ตำแหน่งอื่นภายในเมท็อด bringToFront ยังคงมีการเข้าถึงฟิลด์ drawArea ส่งผลให้ค่ามาตรวัด LCO ของคลาสหลังทำแอสเป็กรีแฟคทอริงยังมีค่าคงเดิม เพราะการเข้าถึงฟิลด์ของคลาสระหว่างเมท็อดไม่เปลี่ยนแปลง

```
public class DefaultDrawing
                    extends AbstractDrawing {
    private Rectangle2D.Double drawArea;
    public DefaultDrawing() {    }
    public void bringToFront(Figure figure) {
        if (basicRemove(figure) != -1) {
            basicAdd(figure);
            invalidateSortOrder();
            drawArea = figure.getDrawingArea();
            fireAreaInvalidated(drawArea);
        }
    }
}
```

→

```
public class DefaultDrawing
                    extends AbstractDrawing {
    private Rectangle2D.Double drawArea;
    public DefaultDrawing() {    }
    public void bringToFront(Figure figure) {
        if (basicRemove(figure) != -1) {
            basicAdd(figure);
            invalidateSortOrder();
            drawArea = figure.getDrawingArea();
        }
    }
}
```

รูปที่ 4.63 คลาส DefaultDrawing ก่อนและหลังการทำแอสเป็กรีแฟคทอริง

แต่สำหรับเจฮอตดรอว์ เวอร์ชั่น 12 เป็นเวอร์ชั่นเดียวที่ค่าเฉลี่ยของมาตรวัด LCO ให้ผลไม่เหมือนกับมาตรวัด LCOM ของมาตรวัดเชิงวัตถุ คือเมื่อพิจารณาด้วยมาตรวัด LCO แสดงให้เห็นว่าหลังทำแอสเป็กรีแฟคทอริงไม่มีคลาสที่มีค่ามาตรวัด LCO เปลี่ยนแปลง ขณะที่พิจารณาด้วยมาตรวัด LCOM มีคลาส 3 คลาสที่มีค่ามาตรวัด LCOM เปลี่ยนแปลง ข้อแตกต่างดังกล่าวเกิดจากหลักการเก็บค่ามาตรวัดที่แตกต่างกันของเครื่องมือ เนื่องจากเจฮอตดรอว์ เวอร์ชั่น 12 ภายในซอร์สโค้ดมีตำแหน่งของเมท็อดคอลที่ต้องการจัดการด้วยแอสเป็ก โดยตำแหน่งของเมท็อดคอลอยู่ภายในคอนสตรัคเตอร์ของคลาส โดยการเก็บค่ามาตรวัด LCO ของคลาสด้วยเครื่องมือเอโอพีเมตริก (AOPMetrics) จะพิจารณาการเข้าถึงฟิลด์คลาสระหว่างเมท็อดเท่านั้น โดยไม่พิจารณาการเข้าถึงฟิลด์คลาสที่มาจาก

คอนสตรัคเตอร์ ดังนั้นหลังจัดการเมท็อดคอลดังกล่าวด้วยแอสเป็ก จึงไม่ส่งผลให้การเข้าถึงฟิลด์คลาสระหว่างเมท็อดเปลี่ยนแปลง ค่ามาตรวัด LCO ของคลาสหลังทำแอสเป็กรีแฟคทอริงจึงมีค่าคงเดิม ขณะที่การเก็บค่ามาตรวัด LCOM ของคลาสด้วยเครื่องมือซีเคเจเอ็ม (CKJM) จะพิจารณาการเข้าถึงฟิลด์คลาสที่มาจากทั้งในคอนสตรัคเตอร์และเมท็อดของคลาส ดังนั้นหลังจัดการเมท็อดคอลดังกล่าวด้วยแอสเป็ก จึงส่งผลให้การเข้าถึงฟิลด์คลาสระหว่างเมท็อดและคอนสตรัคเตอร์เปลี่ยนแปลง หลังทำแอสเป็กรีแฟคทอริงค่ามาตรวัด LCOM ของคลาสจึงมีค่าเพิ่มขึ้น

4.6 การอภิปรายผลและการศึกษาเพิ่มเติม

จากการทำแอสเป็กรีแฟคทอริงปรับปรุงซอร์สโค้ดของซอฟต์แวร์เจฮอตดรอว์ เวอร์ชั่น 7.1 เพื่อศึกษาผลคุณภาพซอฟต์แวร์เมื่อจัดการจำนวนซ้ำของเมท็อดคอลด้วยแอสเป็กในจำนวนซ้ำที่แตกต่างกัน โดยประเมินด้วยสองชุดมาตรวัด ได้แก่ มาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็ก สามารถอภิปรายผลได้ดังนี้

การเปรียบเทียบผลมาตรวัดเชิงวัตถุทั้ง 6 มาตรวัดของเจฮอตดรอว์ที่ปรับปรุงซอร์สโค้ดโดยการทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลในจำนวนซ้ำที่แตกต่างกันด้วยแอสเป็ก มาตรวัดที่ได้รับผลกระทบโดดเด่นจากการทำแอสเป็กรีแฟคทอริงมีเพียงสองมาตรวัด ได้แก่

- คัพพลิงบีทวีนอ็อบเจกต์คลาส (Coupling Between Object classes - CBO)

จากเจฮอตดรอว์ทั้งหมดมีเพียง 2 เวอร์ชั่นที่ค่ามาตรวัด CBO ของคลาสมีค่ามาตรวัดลดลง โดยจากการวิเคราะห์พบว่าปัจจัยที่ส่งผลให้ค่ามาตรวัดลดลงไม่ได้มีสาเหตุมาจากการจัดการเมท็อดคอลในจำนวนซ้ำที่น้อยหรือมากด้วยแอสเป็ก แต่ปัจจัยที่ส่งผลให้ค่ามาตรวัด CBO ลดลงคือลักษณะเมท็อดคอล โดยจากการทดลองในงานวิจัยนี้พบว่าลักษณะเมท็อดคอลที่เรียกใช้งานเมท็อดผ่านชื่อคลาสโดยตรงหรือเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์ของคลาสแม่ ทั้งสองลักษณะเมท็อดคอลเมื่อนำแอสเป็กมาใช้จัดการเมท็อดคอลจะส่งผลให้ค่ามาตรวัด CBO ของคลาสมีค่าลดลง และจากความสัมพันธ์ระหว่างมาตรวัดและคุณภาพซอฟต์แวร์ พบว่าหากซอฟต์แวร์มีการขึ้นต่อกันระหว่างคลาสที่น้อยจะช่วยปรับปรุงคุณภาพด้านความสามารถในการทำความเข้าใจ ความสามารถการบำรุงรักษา ความสามารถการนำกลับมาใช้ใหม่ และความสามารถการทดสอบ ดังนั้นเมื่อพิจารณาด้วยมาตรวัด CBO การทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลซ้ำกันด้วยแอสเป็ก การพิจารณาลักษณะเมท็อดคอลที่เหมาะสมก่อนการทำแอสเป็กรีแฟคทอริงจะสามารถช่วยปรับปรุงคุณภาพของซอฟต์แวร์ได้ในหลายด้าน

- เรสปอนซ์ฟอร์อะคลาส (Response For a Class - RFC)

จากผลข้อมูลมาตรวัด RFC แสดงให้เห็นว่าการทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลซ้ำกันด้วยแอสเป็กส่งผลโดยตรงต่อค่ามาตรวัด RFC เนื่องจากเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น มีคลาสที่มีค่ามาตรวัด RFC ลดลง โดยปัจจัยที่สามารถส่งผลให้ค่ามาตรวัด RFC มีค่าลดลงคือจำนวนซ้ำของเมท็อดคอลและจำนวนคลาสที่สามารถจัดการเมท็อดคอลซ้ำกันทั้งหมดด้วยแอสเป็ก โดยจากการทดลองในงานวิจัยนี้พบว่าค่าเฉลี่ยมาตรวัด RFC ของซอฟต์แวร์จะมีค่าลดลงมากหลังทำแอสเป็กรีแฟคทอริง หากมีจำนวนซ้ำของเมท็อดที่ต้องการจัดการด้วยแอสเป็กในจำนวนซ้ำที่มากและเมท็อดคอลที่ซ้ำกันมีตำแหน่งกระจายอยู่ภายในคลาสจำนวนที่มากด้วยเช่นกัน เมื่อจัดการเมท็อดคอล

ทั้งหมดด้วยแอสเป็กจึงส่งผลให้มีคลาสจำนวนมากที่มีค่ามาตรวัด RFC ลดลง และทำให้ค่าเฉลี่ยมาตรวัด RFC ของซอฟต์แวร์ลดลงด้วย จากความสัมพันธ์ระหว่างมาตรวัดและคุณภาพซอฟต์แวร์ พบว่าหากสามารถลดการเรียกใช้งานเมท็อดอื่นๆจากภายในคลาสได้มากจะช่วยปรับปรุงคุณภาพด้านความสามารถในการทำความเข้าใจและความสามารถการทดสอบของซอฟต์แวร์ ดังนั้นการทำแอสเป็กรีแฟคทอริงโดยจัดการเมท็อดคอลที่มีจำนวนซ้ำของเมท็อดคอลมากและมีตำแหน่งเมท็อดคอลกระจัดกระจายอยู่ในหลายคลาส การนำแอสเป็กมาใช้พัฒนาจะสามารถช่วยปรับปรุงคุณภาพซอฟต์แวร์ทั้งสองด้านได้

การเปรียบเทียบค่ามาตรวัดเชิงแอสเป็กทั้ง 10 มาตรวัดของเจฮอตดรอว์ที่ปรับปรุงซอร์สโค้ดโดยการทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลในจำนวนซ้ำที่แตกต่างกันด้วยแอสเป็ก มาตรวัดที่ได้รับผลกระทบโดดเด่นจากการทำแอสเป็กรีแฟคทอริงมีดังนี้

- เวทเทดโอเปอร์เรชั่นอินมอดูล (Weighted Operations in Module - WOM)

ผลจากการทำแอสเป็กรีแฟคทอริงไม่ส่งผลกระทบต่อค่ามาตรวัด WOM ของคลาส แต่จากการทำแอสเป็กรีแฟคทอริงทำให้มีแอสเป็กเป็นส่วนที่สร้างขึ้นมาใหม่ ค่าเฉลี่ยมาตรวัด WOM ของซอฟต์แวร์จึงมีค่าเพิ่มขึ้นจากจำนวนแอดไวซ์ที่อิมพลีเม้นต์ภายในแอสเป็ก ปัจจัยที่ส่งผลต่อค่ามาตรวัด WOM จึงไม่ใช่จำนวนซ้ำของเมท็อดคอลแต่เป็นจำนวนแอดไวซ์ที่อิมพลีเม้นต์ภายในแอสเป็ก จากผลค่ามาตรวัด WOM ของแอสเป็กอาจส่งผลต่อคุณภาพซอฟต์แวร์ทำให้มีความซับซ้อนภายในซอฟต์แวร์มากขึ้นและการบำรุงรักษาซอฟต์แวร์ทำได้ยากขึ้น

- คัพพลิงออนเมท็อดคอล (Coupling on Method Call - CMC) และคัพพลิงบีทวีนมอดูล (Coupling between Modules - CBM)

เนื่องจากทั้งสองมาตรวัดเป็นมาตรวัดที่เกี่ยวข้องกับคัพพลิง โดยมีส่วนของการพิจารณาความสัมพันธ์ระหว่างคลาสที่เหมือนกันคือ พิจารณาความสัมพันธ์ระหว่างคลาสที่เกิดจากการเรียกใช้งานเมท็อด ดังนั้นผลจากการทำแอสเป็กรีแฟคทอริงจึงกระทบต่อค่ามาตรวัดบางคลาสของเจฮอตดรอว์ โดยมีเพียงเจฮอตดรอว์ 4 เวอร์ชั่นเท่านั้นที่ค่ามาตรวัด CMC และ CBM ของบางคลาสมีค่าลดลง ปัจจัยที่ส่งผลให้มาตรวัดมีค่าลดลงไม่ได้เกิดจากจำนวนซ้ำของเมท็อดคอลที่แตกต่างกัน แต่ปัจจัยที่ส่งผลคือลักษณะเมท็อดคอล โดยจากการทดลองของงานวิจัยนี้พบว่าลักษณะเมท็อดคอลที่เรียกใช้งานเมท็อดผ่านชื่อคลาสโดยตรง เมื่อจัดการเมท็อดคอลด้วยแอสเป็กสามารถลดความสัมพันธ์ระหว่างคลาสได้ และส่งผลให้คุณภาพซอฟต์แวร์ด้านความสามารถการบำรุงรักษาและความสามารถการนำกลับมาใช้ใหม่ปรับปรุงดีขึ้น

- ครอสคัทติ้งดีกรีออฟแอนแอสเป็ก (Crosscutting Degree of an Aspect - CDA) และคัพพลิงออนแอดไวซ์เอ็กซิคิวชั่น (Coupling on Advice Execution - CAE)

เนื่องจากการทำแอสเป็กรีแฟคทอริงของเจฮอตดรอว์แต่ละเวอร์ชั่นจะใช้แอสเป็กเพียง 1 แอสเป็กจึงทำให้ค่าเฉลี่ยระหว่างมาตรวัด CDA และมาตรวัด CAE มีค่าเท่ากันเมื่อเปรียบเทียบระหว่างเจฮอตดรอว์ทั้ง 12 เวอร์ชั่น มาตรวัด CDA ใช้วัดแอสเป็กโดยพิจารณาจำนวนคลาสที่แอสเป็กสามารถเข้าไปขัดขวางการทำงาน ขณะที่มาตรวัด CAE ใช้วัดคลาสจากจำนวนแอสเป็กที่เข้ามาขัดขวางการทำงาน และเนื่องจากเจฮอตดรอว์แต่ละเวอร์ชั่นจะสร้างแอสเป็กเพียง 1 แอสเป็ก ผลรวมค่ามาตรวัด CAE ของคลาสทั้งหมดในเจฮอตดรอว์จึงมีค่าเท่ากับค่ามาตรวัด CDA ของแอสเป็ก ทำให้

ปัจจัยที่ส่งผลต่อมาตรวัดทั้งสองมาตรวัดไม่ใช่จำนวนซ้ำของเมท็อดคอล แต่เป็นจำนวนคลาสที่แอสเป็กสามารถเข้าไปขัดขวางการทำงาน จากการพิจารณาผลของการทำแอสเป็กรีแฟคทอริงด้วยมาตรวัด CDA และ CAE ต่อคุณภาพซอฟต์แวร์ เนื่องจากยังไม่พบงานวิจัยที่แสดงผลทั้งสองมาตรวัดต่อคุณภาพจึงไม่สามารถสรุปได้

- เรสปอนซ์ฟอร์อะมอดูล (Response For a Module - RFM)

ผลจากการทำแอสเป็กรีแฟคทอริงส่วนใหญ่พบว่าค่ามาตรวัด RFM ของคลาสที่จัดการเมท็อดคอลด้วยแอสเป็กส่วนใหญ่มีค่ามาตรวัดเพิ่มขึ้น โดยปัจจัยที่ส่งผลต่อค่ามาตรวัด RFM ไม่ใช่จำนวนซ้ำของเมท็อดคอล แต่เป็นจำนวนแอดไวซ์ที่สามารถเข้าไปขัดขวางการทำงานของคลาส ส่งผลให้หลังทำแอสเป็กรีแฟคทอริงค่าเฉลี่ยมาตรวัด RFM ของเจฮอตดรอว์มีค่าเพิ่มขึ้น จากผลค่ามาตรวัด RFM อาจส่งผลลบต่อคุณภาพซอฟต์แวร์ด้านความสามารถในการทำความเข้าใจและความสามารถการทดสอบ

ดังนั้นจากผลการวิเคราะห์มาตรวัดเชิงวัตถุพบว่าการจัดการเมท็อดคอลในจำนวนซ้ำที่แตกต่างกันด้วยแอสเป็ก มีปัจจัยอื่นๆเพิ่มเติมที่ส่งผลต่อค่ามาตรวัด โดยมาตรวัดคัพพลิงบีทวีนอ็อบเจกต์คลาส (Coupling Between Object classes - CBO) พบว่าลักษณะเมท็อดคอลเป็นปัจจัยที่ส่งผลต่อค่ามาตรวัด และมาตรวัดเรสปอนซ์ฟอร์อะคลาส (Response For a Class - RFC) พบว่าจำนวนคลาสที่สามารถจัดการเมท็อดคอลทั้งหมดด้วยแอสเป็กเป็นปัจจัยนอกเหนือจากจำนวนซ้ำของเมท็อดคอลที่ส่งผลต่อค่ามาตรวัด โดยทั้งสองมาตรวัดแสดงผลค่ามาตรวัดของคลาสที่มีค่าลดลงและปรับปรุงคุณภาพซอฟต์แวร์ ผู้วิจัยจึงนำมาศึกษาและอธิบายรายละเอียดเพิ่มเติม ขณะที่ผลการวิเคราะห์มาตรวัดเชิงแอสเป็กในการจัดการเมท็อดคอลที่มีจำนวนซ้ำแตกต่างกันด้วยแอสเป็ก พบว่าปัจจัยที่ส่งผลต่อค่ามาตรวัดไม่ได้เกิดจากจำนวนซ้ำของเมท็อดคอล และเนื่องจากมาตรวัดเชิงแอสเป็กเป็นชุดมาตรวัดที่ปรับปรุงมาจากมาตรวัดเชิงวัตถุ โดยในการทำแอสเป็กรีแฟคทอริง บางมาตรวัดของมาตรวัดเชิงแอสเป็กแสดงผลค่ามาตรวัดและคุณภาพซอฟต์แวร์สอดคล้องกับมาตรวัดเชิงวัตถุ และบางมาตรวัดของมาตรวัดเชิงแอสเป็กแสดงผลค่ามาตรวัดและคุณภาพซอฟต์แวร์ขัดแย้งกับมาตรวัดเชิงวัตถุ ดังนั้นผู้วิจัยจึงศึกษาและอธิบายรายละเอียดเพิ่มเติมเพื่อเปรียบเทียบผลค่ามาตรวัดและคุณภาพซอฟต์แวร์ระหว่างมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็ก การศึกษาเพิ่มเติมมีรายละเอียดดังนี้

4.6.1 ปัจจัยที่ส่งผลต่อค่ามาตรวัดเรสปอนซ์ฟอร์อะคลาสจากการทำแอสเป็กรีแฟคทอริง

จากผลการวิเคราะห์มาตรวัดเรสปอนซ์ฟอร์อะคลาส (Response For a Class - RFC) พบว่าการทำแอสเป็กรีแฟคทอริง โดยจัดการเมท็อดคอลที่ซ้ำกันมากกว่า 29 ตำแหน่ง ค่าเฉลี่ยมาตรวัด RFC มีแนวโน้มลดลง นอกจากนี้ยังพบปัจจัยที่ส่งผลให้ค่าเฉลี่ยมาตรวัด RFC มีค่าลดลง โดยหากเมท็อดคอลที่ซ้ำกันมีตำแหน่งของเมท็อดคอลกระจัดกระจายอยู่ภายในคลาสจำนวนมาก และสามารถจัดการเมท็อดคอลทั้งหมดออกจากคลาสได้ในจำนวนคลาสที่มากเช่นกัน จะทำให้ค่าเฉลี่ยมาตรวัด RFC มีค่าลดลงมาก ดังนั้นอาจกล่าวได้ว่าค่าเฉลี่ยมาตรวัด RFC จะมีค่าลดลงมากหรือน้อยขึ้นอยู่กับจำนวนคลาสที่สามารถจัดการเมท็อดคอลทั้งหมดด้วยแอสเป็กได้ ดังนั้นผู้วิจัยจึงตั้งสมมติฐานเพื่อพิสูจน์ข้อค้นพบดังกล่าวคือ การเปรียบเทียบค่ามาตรวัด RFC ของเจฮอตดรอว์ที่ปรับปรุงด้วยการทำแอสเป็กรีแฟคทอริงโดยมีจำนวนคลาสที่สามารถจัดการเมท็อดคอลทั้งหมดด้วยแอสเป็กแตกต่างกัน

ผู้วิจัยจึงนำเวอร์ชั่นของเจฮอตดรอว์มาจัดเรียงตามจำนวนคลาสที่สามารถจัดการเมท็อดคอลทั้งหมดด้วยแอสเป็ก โดยเรียงจากจำนวนคลาสจำนวนน้อยไปยังจำนวนมากดังตารางที่ 4.84

ตารางที่ 4.84 เวอร์ชั่นของเจฮอตดรอว์จัดเรียงตามจำนวนคลาสที่สามารถจัดการเมท็อดคอลซ้ำกันทั้งหมดด้วยแอสเป็ก

เจฮอตดรอว์	**จำนวนซ้ำของเมท็อดคอล**	**จำนวนคลาสที่สามารถจัดการเมท็อดคอลทั้งหมด**
เวอร์ชั่น 9	21	0
เวอร์ชั่น 1	4	2
เวอร์ชั่น 4	12	4
เวอร์ชั่น 2	5	5
เวอร์ชั่น 6	13	5
เวอร์ชั่น 5	12	8
เวอร์ชั่น 3	9	9
เวอร์ชั่น 7	15	10
เวอร์ชั่น 8	18	12
เวอร์ชั่น 10	29	14
เวอร์ชั่น 11	37	21
เวอร์ชั่น 12	44	43

การเปรียบเทียบค่ามาตรวัด RFC หลังทำแอสเป็กรีแฟคทอริงของเจฮอตดรอว์ที่มีจำนวนคลาสที่สามารถจัดการเมท็อดคอลทั้งหมดด้วยแอสเป็กแตกต่างกัน สามารถตั้งสมมติฐานได้ดังนี้

กำหนดให้

I และ J คือ จำนวนคลาสที่สามารถจัดการเมท็อดคอลด้วยแอสเป็กแสดงดังตารางที่ 4.84 ข้างต้น โดย $I<J$ และ $I \neq J$

μ_1 คือค่ามาตรวัด RFC รายคลาสหลังทำแอสเป็กรีแฟคทอริงของเจฮอตดรอว์ที่สามารถจัดการคลาสทั้งหมด I คลาส

μ_2 คือค่ามาตรวัด RFC รายคลาสหลังทำแอสเป็กรีแฟคทอริงของเจฮอตดรอว์ที่สามารถจัดการคลาสทั้งหมด J คลาส

$$H_0 : \mu_i \leq \mu_j$$

$$H_1 : \mu_i > \mu_j$$

ตารางที่ 4.85 สมมติฐานของการเปรียบเทียบค่ามาตรวัด RFC หลังทำแอสเป็กรีแฟคทอริง ที่มีจำนวนคลาสที่สามารถจัดการด้วยแอสเป็กแตกต่างกัน

		จำนวนคลาสที่สามารถจัดการด้วยแอสเป็ก						
จำนวนคลาสที่สามารถจัดการด้วยแอสเป็ก	j / i	0	2	4	5	5	8	9
	0		$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$
	2			$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$
	4				$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$
	5					$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$
	5						$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$
	8							$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$
	9							
	10							
	12							
	14							
	21							
	43							

ตารางที่ 4.85 สมมติฐานของการเปรียบเทียบค่ามาตรวัด RFC หลังทำแอสเป็กรีแฟคทอริง ที่มีจำนวนคลาสที่สามารถจัดการด้วยแอสเป็กแตกต่างกัน (ต่อ)

		จำนวนคลาสที่สามารถจัดการด้วยแอสเป็ก				
จำนวนคลาสที่สามารถจัดการด้วยแอสเป็ก	i \ j	10	12	14	21	43
	0	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$
	2	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$
	4	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$
	5	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$
	5	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$
	8	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$
	9	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$
	10		$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$
	12			$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$
	14				$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$	$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$
	21					$H_0 : \mu_i \le \mu_j$ $H_1 : \mu_i > \mu_j$
	43					

การทดสอบการแจกแจงข้อมูลเพื่อเลือกวิธีการทดสอบสมมติฐาน โดยทดสอบการแจกแจงข้อมูลค่ามาตรวัด RFC ทั้ง 442 คลาสของเจฮอตดรอว์แต่ละเวอร์ชั่น สามารถเลือกสถิติทดสอบการแจกแจงข้อมูลได้จากขนาดตัวอย่าง หากขนาดตัวอย่างมากกว่า 50 หน่วย การทดสอบการแจกแจงของข้อมูลจะเลือกใช้สถิติทดสอบโคลโมโกรอฟ-สเมอร์นอฟ (Kolmogorov-Smirnov) แต่หากขนาดตัวอย่างน้อยกว่า 50 หน่วย การทดสอบการแจกแจงของข้อมูลจะเลือกใช้สถิติทดสอบชาพิโร-วิลค์ (Shapiro-Wilk) (กัลยา วานิชย์บัญชา, 2551a) เนื่องจากเจฮอตดรอว์แต่ละเวอร์ชั่นมีจำนวนคลาสมากกว่า 50 คลาส จึงใช้สถิติทดสอบโคลโมโกรอฟ-สเมอร์นอฟสำหรับทดสอบการแจกแจงของข้อมูล และยอมรับสมมติฐาน H_0 เมื่อค่า Sig. มีค่ามากกว่าค่า α ซึ่งกำหนดให้เท่ากับ 0.05 สามารถตั้งสมมติฐานสำหรับทดสอบการแจกแจงของข้อมูลดังนี้

กำหนดให้ M คือ เจฮอตดรอว์ เวอร์ชั่น 1-12

H_0 : ข้อมูลค่ามาตรวัด RFC ทั้ง 442 คลาสของเจฮอตดรอว์ เวอร์ชั่น M มีการแจกแจงแบบปกติ

H_1 : ข้อมูลค่ามาตรวัด RFC ทั้ง 442 คลาสของเจฮอตดรอว์ เวอร์ชั่น M ไม่ได้แจกแจงแบบปกติ

ตารางที่ 4.86 การทดสอบการแจกแจงของข้อมูลค่ามาตรวัด RFC ของคลาส 442 คลาส จากเจฮอตดรอว์แต่ละเวอร์ชั่น

เจฮอตดรอว์	จำนวนซ้ำเมท็อดคอล	โคลโมโกรอฟ-สเมอร์นอฟ		การแจกแจงของข้อมูล
		ค่าสถิติ	Sig	
เวอร์ชั่น 1	4	0.164	0.000	ไม่ปกติ
เวอร์ชั่น 2	5	0.164	0.000	ไม่ปกติ
เวอร์ชั่น 3	9	0.164	0.000	ไม่ปกติ
เวอร์ชั่น 4	12	0.164	0.000	ไม่ปกติ
เวอร์ชั่น 5	12	0.164	0.000	ไม่ปกติ
เวอร์ชั่น 6	13	0.164	0.000	ไม่ปกติ
เวอร์ชั่น 7	15	0.164	0.000	ไม่ปกติ
เวอร์ชั่น 8	18	0.164	0.000	ไม่ปกติ
เวอร์ชั่น 9	21	0.164	0.000	ไม่ปกติ
เวอร์ชั่น 10	29	0.164	0.000	ไม่ปกติ
เวอร์ชั่น 11	37	0.164	0.000	ไม่ปกติ
เวอร์ชั่น 12	44	0.164	0.000	ไม่ปกติ

จากการทดสอบด้วยสถิติโคลโมโกรอฟ-สเมอร์นอฟ ดังตารางที่ 4.86 พบว่าค่า Sig. ของข้อมูลค่ามาตรวัด RFC ของเจฮอตดรอว์แต่ละเวอร์ชั่นมีค่าน้อยกว่า α ซึ่งกำหนดให้เท่ากับ 0.05 จึงปฏิเสธสมมติฐาน H_0 ดังนั้นข้อมูลค่ามาตรวัด RFC รายคลาสของเจฮอตดรอว์ทุกเวอร์ชั่นไม่ได้แจกแจงแบบปกติ การตอบวัตถุประสงค์ผู้วิจัยจึงเลือกใช้วิธีการทดสอบสมมติฐานแบบไม่อิงพารามิเตอร์ (Nonparametric Test) โดยเลือกใช้สถิติทดสอบวิลคอกซัน ไซน์-แรงค์ เทสต์ (Wilcoxon signed-rank test) เป็นการทดสอบที่หาค่าผลต่างระหว่างค่าของตัวแปร โดยจับคู่พิจารณาเครื่องหมายและปริมาณผลต่างว่ามากหรือน้อย

จากการทดสอบสถิติทดสอบวิลคอกซัน ไซน์-แรงค์ เทสต์ (Wilcoxon signed-rank test) ของการจับคู่เจฮอตดรอว์ระหว่างเวอร์ชั่นเพื่อเปรียบเทียบผลต่างของค่ามาตรวัด RFC รายคลาส สามารถแสดงค่า Asymp. Sig. (1-tailed) ทั้งหมดดังตารางที่ 4.87 การตอบสมมติฐานจะปฏิเสธ H_0 หากค่า Asymp. Sig. (1-tailed) มีค่าน้อยกว่าระดับนัยสำคัญที่กำหนดคือ 0.05

ตารางที่ 4.87 ค่า Asymp. Sig. (1-tailed) ที่ได้จากสถิติทดสอบ
วิลคอกซัน ไซน์-แรงค์ เทสต์ ของค่ามาตรวัด RFC

	จำนวนคลาสที่สามารถจัดการด้วยแอสเป็ก											
	0	2	4	5	5	8	9	10	12	14	21	43
0		0.079	0.023*	0.013*	0.017*	0.003*	0.002*	0.002*	0.001*	0.000*	0.000*	0.000*
2			0.207	0.129	0.051	0.029*	0.018*	0.006*	0.004*	0.002*	0.000*	0.000*
4				0.370	0.282	0.124	0.048*	0.017*	0.023*	0.009*	0.000*	0.000*
5					0.391	0.203	0.143	0.001*	0.045*	0.020*	0.001*	0.000*
5						0.309	0.234	0.054	0.090	0.044*	0.003*	0.000*
8							0.391	0.089	0.186	0.101	0.008*	0.000*
9								0.122	0.257	0.149	0.012*	0.000*
10									0.262	0.381	0.227	0.001*
12										0.328	0.042*	0.000*
14											0.010*	0.000*
21												0.003*
43												

* มีความแตกต่างอย่างมีนัยสำคัญ

จากการทดสอบสถิติพบว่าเจฮอตดรอว์ เวอร์ชั่น 9 ที่จัดการเมท็อดคอลที่ซ้ำกันทั้งหมด 21 ตำแหน่ง มีจำนวนคลาสที่สามารถจัดการเมท็อดคอลทั้งหมดเท่ากับ 0 คลาส เพราะหลังทำแอสเป็กรีแฟคทอริงไม่มีคลาสที่มีค่ามาตรวัด RFC ลดลง ดังนั้นเมื่อเปรียบเทียบกับเจฮอตดรอว์เวอร์ชั่นอื่นที่มีจำนวนคลาสที่สามารถจัดการเมท็อดคอลตั้งแต่ 4 คลาสขึ้นไปจึงแสดงให้เห็นว่าการทำแอสเป็กรีแฟคทอริงหากมีจำนวนคลาสที่สามารถจัดการเมท็อดคอลภายในคลาสด้วยแอสเป็กได้ โดยส่วนใหญ่จะส่งผลให้ค่ามาตรวัด RFC ของคลาสมีค่าลดลง ขณะที่เจฮอตดรอว์ เวอร์ชั่น 11 และ 12 ที่จัดการเมท็อดคอลซ้ำกันทั้งหมด 37 และ 44 ตำแหน่งตามลำดับ มีจำนวนคลาสจำนวนมากที่สามารถจัดการเมท็อดคอลด้วยแอสเป็ก โดยมีคลาสทั้งหมด 21 และ 43 คลาสตามลำดับ เจฮอตดรอว์ทั้งสองเวอร์ชั่นแสดงให้เห็นว่าหลังทำแอสเป็กรีแฟคทอริงค่ามาตรวัด RFC ของคลาสมีค่าลดลงอย่างมีนัยสำคัญเมื่อเทียบกับเจฮอตดรอว์เวอร์ชั่นอื่นๆ จากผลดังกล่าวอาจกล่าวได้ว่าหากมีคลาสที่สามารถจัดการเมท็อดคอลทั้งหมดด้วยแอสเป็กในจำนวนคลาสที่มากจะส่งผลให้ค่ามาตรวัด RFC ของคลาสมีค่าลดลงอย่างมีนัยสำคัญ และการทำแอสเป็กรีแฟคทอริงหากมีคลาสจำนวนมากที่มีค่ามาตรวัด RFC ลดลงจะส่งผลให้ค่าเฉลี่ยมาตรวัด RFC ของซอฟต์แวร์ลดลงมากเช่นกัน

จากข้อค้นพบปัจจัยที่ส่งผลทำให้ค่ามาตรวัด RFC มีค่าลดลงหลังทำแอสเป็กรีแฟคทอริงคือจำนวนคลาสที่สามารถจัดการเมท็อดคอลทั้งหมดด้วยแอสเป็กได้ โดยจากผลของงานวิจัยแสดงให้เห็นว่าการจัดการเมท็อดคอลที่อยู่ภายในคลาสแตกต่างกันตั้งแต่ 21 คลาสขึ้นไปจะส่งผลให้ค่ามาตรวัด RFC ของคลาสลดลงอย่างมีนัยสำคัญ จึงทำให้ค่าเฉลี่ยมาตรวัด RFC ลดลงและช่วยปรับปรุงคุณภาพซอฟต์แวร์เช่นกัน ดังนั้นจากผลค่าเฉลี่ยมาตรวัด RFC และผลการทดสอบสถิติจึงอาจกล่าวได้ว่าการทำแอสเป็กรีแฟคทอริงโดยจัดการจำนวนซ้ำของเมท็อดคอลในจำนวนซ้ำที่มาก ซึ่งตำแหน่งของเมท็อดคอลกระจัดกระจายอยู่ภายในคลาสจำนวนมาก สามารถส่งผลให้ค่าเฉลี่ยมาตรวัด RFC ของ

ซอฟต์แวร์มีค่าลดลงและปรับปรุงคุณภาพซอฟต์แวร์ด้านความสามารถในการทำความเข้าใจและความสามารถการทดสอบ

4.6.2 ผลของรูปแบบลักษณะเมท็อดคอลต่อค่ามาตรวัดจากการทำแอสเป็กรีแฟคทอริง

จากการพิจารณาลักษณะเมท็อดคอลของเมท็อดคอลทั้งหมดในแต่ละจำนวนซ้ำทำให้ผู้วิจัยสามารถจำแนกรูปแบบของลักษณะเมท็อดคอลได้ทั้งหมด 5 รูปแบบดังอธิบายเอาไว้ในตอนต้นของบทที่ 4 ดังนี้

(1) ลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดจากภายในคลาสโดยตรง

(2) ลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดผ่านชื่อคลาสโดยตรง

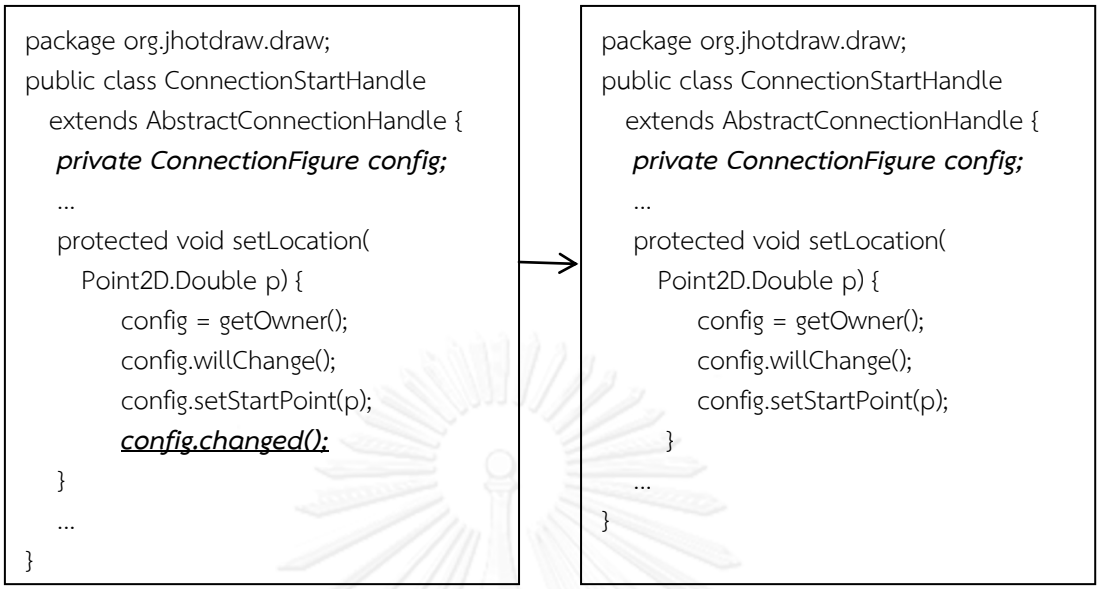
(3) ลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจ็กต์คลาส

(4) ลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์คลาสของคลาสแม่

(5) ลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์ที่เป็นพารามิเตอร์

จากการวิเคราะห์รูปแบบของลักษณะเมท็อดคอลต่อผลมาตรวัด พบว่ารูปแบบลักษณะของเมท็อดคอลทั้ง 5 รูปแบบส่งผลกระทบต่อค่ามาตรวัดที่เกี่ยวข้องกับคัพพลิงที่พิจารณาความสัมพันธ์ระหว่างคลาสที่เกิดจากการเรียกใช้งานเมท็อด โดยมาตรวัดที่มีนิยามบางส่วนพิจารณาความสัมพันธ์ระหว่างคลาสที่เกิดจากการเรียกใช้งานเมท็อด มาตรวัดเชิงวัตถุ ได้แก่ มาตรวัดคัพพลิงบีทวีนอ็อบเจกต์คลาส (Coupling Between Object classes - CBO) มาตรวัดเชิงแอสเป็ก ได้แก่ มาตรวัดคัพพลิงออนเมท็อดคอล (Coupling on Method Call - CMC) และมาตรวัดคัพพลิงบีทวีนมอดูล (Coupling between Modules - CBM) โดยจากการพิจารณาคลาสที่มีค่ามาตรวัดลดลงพบว่าลักษณะเมท็อดคอลที่จัดการด้วยแอสเป็กแล้วทำให้ค่ามาตรวัดของคลาสลดลง คือลักษณะเมท็อดคอลรูปแบบที่ 2 และ 4 เป็นการเรียกใช้งานเมท็อดผ่านชื่อคลาสโดยตรงและการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์คลาสของคลาสแม่ หลังจากทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลทั้งสองรูปแบบด้วยแอสเป็ก มีโอกาสที่จะช่วยลดความสัมพันธ์ระหว่างคลาสได้ ทำให้ค่ามาตรวัดของคลาสมีค่าลดลง

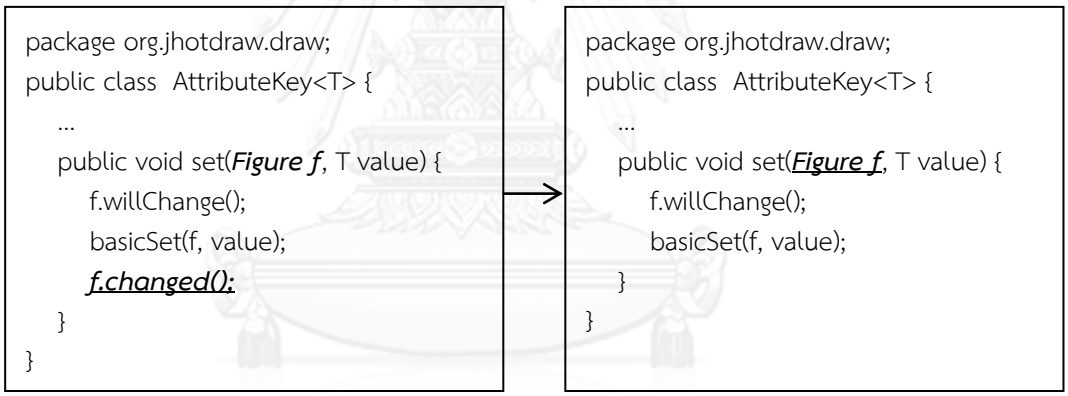
ขณะที่ลักษณะเมท็อดคอลรูปแบบที่ 1 ที่เรียกใช้งานเมท็อดจากภายในคลาสโดยตรง เนื่องจากการเรียกใช้งานเมท็อดจากภายในคลาสได้ แสดงว่าเมท็อดดังกล่าวอิมพลีเม้นต์ไว้ภายในคลาสเดียวกันเลย หรือเมท็อดอิมพลีเม้นต์เอาไว้ภายในคลาสแม่ที่อยู่ลำดับชั้นคลาสเดียวกัน ดังนั้นถึงแม้จัดการเมท็อดคอลที่เป็นเพียงคำสั่งหนึ่งคำสั่งในคลาสออกมาไว้ในแอสเป็ก แต่ความสัมพันธ์ระหว่างคลาสแม่และคลาสลูกก็ไม่เปลี่ยนแปลง ค่ามาตรวัดของคลาสจึงมีค่าคงเดิม ส่วนลักษณะเมท็อดคอลรูปแบบที่ 3 และ 5 ที่เป็นการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจ็กต์คลาสและการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์ที่เป็นพารามิเตอร์ เนื่องจากถึงแม้จะจัดการเมท็อดคอลออกไปจากคลาสได้ แต่ตำแหน่งอื่นภายในคลาสยังคงมีการเข้าถึงหรือประกาศตัวแปรอ็อบเจกต์ของคลาสเอาไว้ การทำแอสเป็กรีแฟคทอริงจึงไม่สามารถลดความสัมพันธ์ระหว่างคลาสได้ ค่ามาตรวัดของคลาสจึงมีค่าคงเดิม แสดงตัวอย่างการทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลที่มีลักษณะเมท็อดคอลรูปแบบที่ 3 และ 5 ดังรูปที่ 4.64 และ 4.65 ตามลำดับ



```
package org.jhotdraw.draw;
public class ConnectionStartHandle
   extends AbstractConnectionHandle {
    private ConnectionFigure config;
    ...
    protected void setLocation(
       Point2D.Double p) {
           config = getOwner();
           config.willChange();
           config.setStartPoint(p);
           config.changed();
    }
    ...
}
```

```
package org.jhotdraw.draw;
public class ConnectionStartHandle
   extends AbstractConnectionHandle {
    private ConnectionFigure config;
    ...
    protected void setLocation(
       Point2D.Double p) {
           config = getOwner();
           config.willChange();
           config.setStartPoint(p);
       }
    ...
}
```

รูปที่ 4.64 ผลจากการจัดการเมท็อดคอลที่มีลักษณะการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์คลาส



```
package org.jhotdraw.draw;
public class  AttributeKey<T> {
    ...
    public void set(Figure f, T value) {
        f.willChange();
        basicSet(f, value);
        f.changed();
    }
}
```

```
package org.jhotdraw.draw;
public class  AttributeKey<T> {
    ...
    public void set(Figure f, T value) {
        f.willChange();
        basicSet(f, value);
    }
}
```

รูปที่ 4.65 ผลจากการจัดการเมท็อดคอลที่มีลักษณะการเรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์ที่เป็นพารามิเตอร์

4.6.3 การเปรียบเทียบผลมาตรวัดระหว่างมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็ก

การทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลที่มีจำนวนซ้ำแตกต่างกันด้วยแอสเป็ก ในงานวิจัยนี้ได้ประเมินความแตกต่างของคุณภาพซอฟต์แวร์จากการเก็บรวบรวมค่ามาตรวัดของสองชุดมาตรวัด ได้แก่ มาตรวัดเชิงวัตถุของ Chidamber และ Kemerer (1993) มีทั้งหมด 6 มาตรวัด ได้แก่

(1) เวทเทดเมท็อดเปอคลาส (Weighted Method per Class - WMC)

(2) เดพออฟอินเฮอริแทนทรี (Depth of Inheritance Tree - DIT)

(3) นัมเบอร์ออฟชิลเดรน (Number of Children - NOC)

(4) คัพพลิงบีทวีนอ็อบเจกต์คลาส (Coupling Between Object classes - CBO)

(5) เรสปอนซ์ฟอร์อะคลาส (Response For a Class - RFC)

(6) แลคออฟโคฮีชันอินเมท็อด (Lack of Cohesion in Methods - LCOM)

และมาตรวัดเชิงแอสเป็กของ Ceccato และ Tonella (2004) มีทั้งหมด 10 มาตรวัด ได้แก่

(1) เวทเทดโอเปอร์เรชั่นอินมอดูล (Weighted Operations in Module - WOM)

(2) เดพออฟอินเฮอริแทนทรี (Depth of Inheritance Tree - DIT)

(3) นัมเบอร์ออฟชิลเดรน (Number Of Children - NOC)

(4) คัพพลิงออนฟิลด์แอคเซส (Coupling on Field Access - CFA)

(5) คัพพลิงออนเมท็อดคอล (Coupling on Method Call - CMC)

(6) คัพพลิงบีทวีนมอดูล (Coupling between Modules - CBM)

(7) ครอสคัทติ้งดีกรีออฟแอนแอสเป็ก (Crosscutting Degree of an Aspect - CDA)

(8) คัพพลิงออนแอดไวซ์เอ็กซิคิวชั่น (Coupling on Advice Execution - CAE)

(9) เรสปอนซ์ฟอร์อะมอดูล (Response For a Module - RFM)

(10) แลคออฟโคฮีชันอินโอเปอร์เรชั่น (Lack of Cohesion in Operations - LCO)

เนื่องจากมาตรวัดเชิงแอสเป็กของ Ceccato และ Tonella (2004) ทั้ง 10 มาตรวัด เป็นชุดมาตรวัดที่ปรับปรุงจากมาตรวัดเชิงวัตถุของ Chidamber และ Kemerer (1993) จึงทำให้บางมาตรวัดของมาตรวัดเชิงแอสเป็กมีนิยามคล้ายกันกับมาตรวัดเชิงวัตถุ แต่มาตรวัดเชิงแอสเป็กจะเพิ่มเติมนิยามสำหรับการพิจารณาแอสเป็ก จากตารางที่ 2.2 ที่กล่าวไว้ในบทที่ 2 แสดงการเปรียบเทียบนิยามระหว่างมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็กที่มีนิยามคล้ายกัน ดังนั้นจากผลการทดลองสำหรับการทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลซ้ำกันด้วยแอสเป็ก สามารถแสดงผลการทำแอสเป็กรีแฟคทอริงต่อค่ามาตรวัด โดยเปรียบเทียบผลระหว่างมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็กที่มีนิยามคล้ายกัน แสดงได้ดังตารางที่ 4.88

ตารางที่ 4.88 การเปรียบเทียบผลการทำแอสเป็กรีแฟคทอริงต่อค่ามาตรวัด
ระหว่างมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็กที่มีนิยามคล้ายกัน

ลำดับ	มาตรวัดเชิงวัตถุ/มาตรวัดเชิงแอสเป็ก	ผลการทำแอสเป็กรีแฟคทอริงต่อค่ามาตรวัด	ปัจจัยที่ส่งผลต่อค่ามาตรวัด
1	Weighted Method per Class (WMC)	—	-
	Weighted Operations in Module (WOM)	↑	จำนวนแอดไวซ์ที่อิมพลีเม้นต์ภายในแอสเป็ก
2	Depth of Inheritance Tree (DIT)	—	-
	Depth of Inheritance Tree (DIT)	—	-
3	Number of Children (NOC)	—	-
	Number Of Children (NOC)	—	-
4	Coupling Between Object classes (CBO)	↓	ลักษณะเมท็อดคอล
	Coupling on Method Call (CMC) Coupling between Modules (CBM)	↓	ลักษณะเมท็อดคอล
5	Response For a Class (RFC)	↓	จำนวนซ้ำของเมท็อดคอล จำนวนคลาสที่สามารถจัดการเมท็อดคอลทั้งหมดด้วยแอสเป็ก
	Response For a Module (RFM)	↑	จำนวนแอดไวซ์ที่เข้ามาขัดขวางการทำงานคลาส
6	Lack of Cohesion in Methods (LCOM)	—	-
	Lack of Cohesion in Operations (LCO)	—	-

การเปรียบเทียบระหว่างมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็กแสดงผลของมาตรวัดที่สอดคล้องไปในทางเดียวกันได้ดังนี้

- คู่มาตรวัด DIT-DIT คู่มาตรวัด NOC-NOC และคู่มาตรวัด LCOM-LCO จากมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็ก เมื่อเปรียบเทียบผลการทำแอสเป็กรีแฟคทอริงต่อค่ามาตรวัดแสดงให้เห็นว่าจากเจฮอตดรอว์ทั้งหมด 12 เวอร์ชั่นที่จัดการจำนวนซ้ำของเมท็อดคอลแตกต่างกัน การทำแอสเป็กรีแฟคทอริงไม่ส่งผลให้ค่ามาตรวัดเกิดการเปลี่ยนแปลงเหมือนกันในทุกมาตรวัด ผลของมาตรวัดจึงมีความสอดคล้องกันระหว่างมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็กในทั้งสามคู่มาตรวัด

- คู่มาตรวัด CBO-CMC และคู่มาตรวัด CBO-CBM จากมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็ก เนื่องจากเป็นมาตรวัดที่เกี่ยวข้องกับคัพพลิง และทั้งสามมาตรวัดมีส่วนของนิยามที่คล้ายกัน คือการพิจารณาความสัมพันธ์ระหว่างคลาสที่เกิดจากการเรียกใช้งานเมท็อด การทำแอสเป็กรีแฟคทอริงเมื่อจัดการเมท็อดคอลออกจากคลาสจึงส่งผลให้ค่ามาตรวัดของคลาสมีค่าลดลง ผลของมาตรวัดจึงมีความสอดคล้องกันระหว่างมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็กในทั้งสองคู่มาตรวัด

นอกจากนี้การเปรียบเทียบระหว่างมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็กในบางคู่มาตรวัดยังแสดงผลกระทบจากการทำแอสเป็กรีแฟคทอริงต่อค่ามาตรวัดในทิศทางที่ตรงข้ามกัน คู่มาตรวัดที่ให้ผลมาตรวัดไม่สอดคล้องกันแสดงได้ดังนี้

- คู่มาตรวัด WMC-WOM จากมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็ก ผลจากการทำแอสเป็กรีแฟคทอริง เมื่อพิจารณาด้วยมาตรวัด WMC พบว่าค่ามาตรวัดของคลาสมีค่าคงเดิมไม่เปลี่ยนแปลงหลังทำแอสเป็กรีแฟคทอริง โดยการพิจารณาคลาสด้วยมาตรวัด WOM แสดงผลในทางเดียวกันคือค่ามาตรวัด WOM ของคลาสมีค่าไม่เปลี่ยนแปลงเช่นกัน แต่ข้อแตกต่างที่เกิดคือค่าเฉลี่ยมาตรวัด WOM มีค่าเพิ่มขึ้นจากจำนวนแอดไวซ์ที่อิมพลีเม้นต์ในแอสเป็ก เนื่องจากมาตรวัด WMC ไม่พิจารณาแอสเป็กแต่มาตรวัด WOM มีนิยามสำหรับพิจารณาแอสเป็กจึงทำให้ผลมาตรวัดบางส่วนจากการพิจารณาด้วยทั้งสองมาตรวัดแตกต่างกัน

- คู่มาตรวัด RFC-RFM จากมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็ก ผลจากการทำแอสเป็กรีแฟคทอริง เมื่อพิจารณาด้วยมาตรวัด RFC พบว่าหลังทำแอสเป็กรีแฟคทอริงค่ามาตรวัด RFC ของคลาสทั้งหมดมีค่ามาตรวัดลดลง ขณะที่หากพิจารณาด้วยมาตรวัด RFM พบว่าหลังทำแอสเป็กรีแฟคทอริงค่ามาตรวัด RFM ของคลาสทั้งหมดกลับมีค่ามาตรวัดเพิ่มขึ้น เนื่องจากมาตรวัด RFC ไม่ได้พิจารณาผลการทำงานที่มาจากแอดไวซ์ภายในแอสเป็ก ขณะที่มาตรวัด RFM มีนิยามที่เกี่ยวข้องกับแอสเป็กโดยการพิจารณาค่ามาตรวัด RFM ของคลาสจากจำนวนแอดไวซ์ที่เข้ามาขัดขวางการทำงานของคลาสด้วย ผลของมาตรวัดของคลาสหลังทำแอสเป็กรีแฟคทอริงจึงมีความขัดแย้งกันระหว่างมาตรวัด RFC และมาตรวัด RFM

การประเมินคุณภาพซอฟต์แวร์ด้วยมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็กในการทำแอสเป็กรีแฟคทอริง โดยมาตรวัดเชิงวัตถุของ Chidamber และ Kemerer (1993) ทั้งหมด 6 มาตรวัด สามารถแสดงความสัมพันธ์กับคุณภาพซอฟต์แวร์ทั้ง 4 ด้าน ได้แก่ ความสามารถในการทำความเข้าใจ (Understandability) ความสามารถการบำรุงรักษา (Maintainability) ความสามารถการนำกลับมาใช้ใหม่ (Reusability) และความสามารถการทดสอบ (Testability) อ้างอิงจากงานวิจัยของ Kulkarni, Kalshetty และ Arde (2010) ขณะที่มาตรวัดเชิงแอสเป็กของ Ceccato และ Tonella (2004) ทั้งหมด 10 มาตรวัด มีเพียงบางมาตรวัดคือ มาตรวัด WOM, CBM, RFM และ LCO ที่สามารถแสดงความสัมพันธ์กับคุณภาพทั้ง 6 ด้าน โดยมีคุณภาพสี่ด้านที่เหมือนกันกับมาตรวัดเชิงวัตถุ ได้แก่ ความสามารถในการทำความเข้าใจ ความสามารถการบำรุงรักษา ความสามารถการนำกลับมาใช้ใหม่ และความสามารถการทดสอบ แต่มีคุณภาพอีกสองด้านที่พิจารณาเพิ่มเติม ได้แก่ ประสิทธิภาพ (Efficiency) และความซับซ้อน (Complexity) อ้างอิงจากงานวิจัยของ Sirbi และ Kulkarni (2013)

ผลการการทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลที่มีจำนวนซ้ำแตกต่างกันด้วยแอสเป็ก สามารถแสดงผลการทำแอสเป็กรีแฟคทอริงต่อคุณภาพซอฟต์แวร์ได้ โดยเปรียบเทียบผลคุณภาพซอฟต์แวร์ระหว่างมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็กที่มีนิยามคล้ายกันดังตารางที่ 4.89

ตารางที่ 4.89 การเปรียบเทียบผลการทำแอสเป็กรีแฟคทอริงต่อคุณภาพซอฟต์แวร์ระหว่างมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็กที่มีนิยามคล้ายกัน

ลำดับ	มาตรวัดเชิงวัตถุ/ มาตรวัดเชิงแอสเป็ก	Efficiency	Complexity	Understandability	Maintainability	Reusability	Testability	ผลการทำแอสเป็กรีแฟคทอริงต่อคุณภาพซอฟต์แวร์
1	Weighted Method per Class (WMC)	-	-	-	✓	✓	-	—
	Weighted Operations in Module (WOM)	-	✓	-	✓	-	-	↓
2	Depth of Inheritance Tree (DIT)	-	-	✓	-	✓	-	—
	Depth of Inheritance Tree (DIT)	-	-	-	-	-	-	—
3	Number of Children (NOC)	-	-	-	-	✓	✓	—
	Number Of Children (NOC)	-	-	-	-	-	-	—
4	Coupling Between Object classes (CBO)	-	-	✓	✓	✓	✓	↑
	Coupling between Modules (CBM)	-	-	-	✓	✓	-	↑
5	Response For a Class (RFC)	-	-	✓	-	-	✓	↑
	Response For a Module (RFM)	-	-	✓	-	-	✓	↓
6	Lack of Cohesion in Methods (LCOM)	-	-	-	-	-	-	—
	Lack of Cohesion in Operations (LCO)	✓	-	-	-	✓	-	—

การเปรียบเทียบระหว่างมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็กแสดงผลของมาตรวัดที่สอดคล้องไปในทางเดียวกันได้ดังนี้

- ผลคุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริง โดยพิจารณาจากคู่มาตรวัด DIT-DIT คู่มาตรวัด NOC-NOC และคู่มาตรวัด LCOM-LCO ไม่แสดงผลกระทบต่อคุณภาพซอฟต์แวร์ เนื่องจากการทำแอสเป็กรีแฟคทอริงไม่ส่งผลกระทบต่อค่ามาตรวัด
- ผลคุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริง โดยพิจารณาจากคู่มาตรวัด CBO-CBM แสดงผลคุณภาพที่สอดคล้องกันคือ การทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลที่ซ้ำกันด้วยแอสเป็กส่งผลให้คุณภาพซอฟต์แวร์ปรับปรุงดีขึ้น เนื่องจากการทำแอสเป็กรีแฟคทอริงส่งผลให้ค่ามาตรวัดของคลาสลดลง โดยมาตรวัด CBO และ CBM สามารถชี้วัดคุณภาพด้านความสามารถการบำรุงรักษาและความสามารถการนำกลับมาใช้ใหม่ทั้งสองด้านได้เหมือนกัน ขณะที่มาตรวัด CBO แสดงการปรับปรุงของคุณภาพสองด้านเพิ่มเติม ได้แก่ ความสามารถในการทำความเข้าใจและความสามารถการทดสอบที่ปรับปรุงดีขึ้นด้วยเช่นกัน

นอกจากนี้การประเมินคุณภาพจากการทำแอสเป็กรีแฟคทอริงด้วยมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็กในบางคู่มาตรวัดให้ผลคุณภาพแตกต่างกัน ดังนี้

- ผลคุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริง โดยพิจารณาจากคู่มาตรวัด WMC-WOM แสดงผลคุณภาพในทิศทางตรงข้ามกัน โดยการพิจารณาด้วยมาตรวัด WMC แสดงให้เห็นว่าการทำแอสเป็กรีแฟคทอริงไม่ส่งผลกระทบต่อคุณภาพ เนื่องจากหลังทำแอสเป็กรีแฟคทอริงค่ามาตรวัด WMC ของคลาสมีค่าไม่เปลี่ยนแปลง ขณะที่การพิจารณาด้วยมาตรวัด WOM ถึงแม้หลังการทำแอสเป็กรีแฟคทอริงค่ามาตรวัดของคลาสจะมีค่าไม่เปลี่ยนแปลงเช่นกัน แต่เนื่องจากค่ามาตรวัด WOM มีการพิจารณาจำนวนแอดไวซ์ที่อิมพลีเม้นต์ภายในแอสเป็กเพิ่มเติม จึงส่งผลให้ซอฟต์แวร์มีความซับซ้อนมากขึ้นและความสามารถการบำรุงรักษาลดลง
- ผลคุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริง โดยพิจารณาจากคู่มาตรวัด RFC-RFM แสดงผลคุณภาพในทิศทางตรงข้ามกัน ทั้งสองมาตรวัดสามารถใช้ประเมินคุณภาพด้านความสามารถการทำความเข้าใจและความสามารถการทดสอบเหมือนกัน โดยการพิจารณาด้วยมาตรวัด RFC แสดงให้เห็นว่าการทำแอสเป็กรีแฟคทอริงส่งผลให้คุณภาพทั้งสองด้านปรับปรุงดีขึ้น เนื่องจากหลังทำแอสเป็กรีแฟคทอริงค่ามาตรวัด RFC ของคลาสส่วนใหญ่มีค่ามาตรวัดลดลง ขณะที่การพิจารณาด้วยมาตรวัด RFM พบว่าการทำแอสเป็กรีแฟคทอริงส่งผลให้คุณภาพทั้งสองด้านลดลง เนื่องจากการพิจารณานิยามของแอสเป็ก ทำให้มาตรวัด RFM มีการพิจารณาจำนวนแอดไวซ์ที่เข้ามาขัดขวางการทำงานคลาส จึงส่งผลให้ค่ามาตรวัด RFM ของคลาสส่วนใหญ่มีค่าเพิ่มขึ้น

บทที่ 5
สรุปผลการวิจัย

5.1 บทนำ

ในบทนี้จะนำเสนอถึงการสรุปผลของงานวิจัยเพื่อตอบวัตถุประสงค์งานวิจัย ประกอบด้วยหัวข้อย่อยดังนี้ (1) สรุปผลการวิจัย (2) การนำงานวิจัยไปประยุกต์ใช้ และ (3) ข้อจำกัดของงานวิจัยและข้อเสนอแนะ เพื่อเป็นแนวทางสำหรับการศึกษางานวิจัยในอนาคตที่เกี่ยวข้องกับการโปรแกรมเชิงแอสเป็กและการทำแอสเป็กรีแฟคทอริง

5.2 สรุปผลการวิจัย

จากการศึกษาการทำแอสเป็กรีแฟคทอริงโดยกรณีศึกษาที่เลือกใช้ในงานวิจัยคือเจฮอตดรอว์เวอร์ชั่น 7.1 งานวิจัยนี้สนใจศึกษาผลคุณภาพซอฟต์แวร์จากการจัดการเมท็อดคอลที่ซ้ำกันภายในซอร์สโค้ดด้วยแอสเป็กโดยมีจำนวนซ้ำเมท็อดคอลที่แตกต่างกัน การประเมินคุณภาพซอฟต์แวร์พิจารณาจากมาตรวัดเชิงวัตถุของ Chidamber และ Kemerer (1993) ทั้งหมด 6 มาตรวัดและมาตรวัดเชิงแอสเป็กของ Ceccato และ Tonella (2004) ทั้งหมด 10 มาตรวัด ผลการทดลองสามารถสรุปผลแยกตามมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็ก ดังตารางที่ 5.1 และ 5.2 ตามลำดับ

ตารางที่ 5.1 สรุปผลการทำแอสเป็กรีแฟคทอริงจากการพิจารณาด้วยมาตรวัดเชิงวัตถุ

มาตรวัดเชิงวัตถุ	ผลจากการทำแอสเป็กรีแฟคทอริง	ผลจากจำนวนซ้ำเมท็อดคอล	ผลต่อคุณภาพซอฟต์แวร์	ปัจจัยที่ส่งผลต่อค่ามาตรวัด
Weighted Method per Class (WMC)	X	X	—	-
Depth of Inheritance Tree (DIT)	X	X	—	-
Number of Children (NOC)	X	X	—	-
Coupling Between Object classes (CBO)	✓	X	↑	ลักษณะเมท็อดคอล
Response For a Class (RFC)	✓	✓	↑	จำนวนซ้ำเมท็อดคอล จำนวนคลาสที่สามารถจัดการเมท็อดคอลทั้งหมด
Lack of Cohesion in Methods LCOM)	✓	X	—	-

ตารางที่ 5.2 สรุปผลการทำแอสเป็กรีแฟคทอริงจากการพิจารณาด้วยมาตรวัดเชิงแอสเป็ก

มาตรวัดเชิงแอสเป็ก	ผลจากการทำแอสเป็กรีแฟคทอริง	ผลจากจำนวนซ้ำเมท็อดคอล	ผลต่อคุณภาพซอฟต์แวร์	ปัจจัยที่ส่งผลต่อค่ามาตรวัด
Weighted Operations in Module (WOM)	✓	X	↓	จำนวนแอดไวซ์ภายในแอสเป็ก
Depth of Inheritance Tree (DIT)	X	X	—	-
Number Of Children (NOC)	X	X	—	-
Coupling on Field Access (CFA)	✓	X	—	จำนวนการเข้าถึงแอตทริบิวต์ของคลาสอื่น
Coupling on Method Call (CMC)	✓	X	—	ลักษณะเมท็อดคอล
Coupling between Modules (CBM)	✓	X	↑	ลักษณะเมท็อดคอล
Crosscutting Degree of an Aspect (CDA)	✓	X	—	จำนวนคลาสที่แอสเป็กสามารถเข้าไปขัดขวางการทำงาน
Coupling on Advice Execution (CAE)	✓	X	—	จำนวนคลาสที่แอสเป็กสามารถเข้าไปขัดขวางการทำงาน
Response For a Module (RFM)	✓	X	↓	จำนวนแอดไวซ์ที่เข้ามาขัดขวางการทำงานคลาส
Lack of Cohesion in Operations (LCO)	X	X	—	-

จากตารางที่ 5.1 และ 5.2 สามารถสรุปผลโดยแสดงมาตรวัดที่ได้รับผลกระทบจากการทำแอสเป็กรีแฟคทอริง มาตรวัดที่ได้รับผลกระทบจากการจัดการจำนวนซ้ำของเมท็อดคอลที่แตกต่างกัน มาตรวัดที่ส่งผลต่อคุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริง และปัจจัยที่ส่งผลให้ค่ามาตรวัดเปลี่ยนแปลง ข้อมูลสรุปผลดังกล่าวสามารถนำมาอธิบายเพื่อตอบวัตถุประสงค์ของงานวิจัย ได้แก่ (1) เปรียบเทียบมาตรวัดเชิงวัตถุของซอฟต์แวร์ที่ปรับปรุงด้วยการทำแอสเป็กรีแฟคทอริงจากจำนวนโค้ดโคลนที่แตกต่างกัน (2) เปรียบเทียบมาตรวัดเชิงแอสเป็กของซอฟต์แวร์ที่ปรับปรุงด้วยการทำแอสเป็กรีแฟคทอริงจากจำนวนโค้ดโคลนที่แตกต่างกัน และ (3) วิเคราะห์คุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริงด้วยจำนวนโค้ดโคลนที่แตกต่างกัน สามารถอภิปรายสรุปผลการวิจัยแยกตามประเด็นเพื่อตอบวัตถุประสงค์งานวิจัยได้ดังนี้

5.2.1 ผลการเปรียบเทียบมาตรวัดเชิงวัตถุของซอฟต์แวร์ที่ปรับปรุงด้วยการทำแอสเป็กรีแฟคทอริงโดยจัดการจำนวนซ้ำของเมท็อดคอลแตกต่างกัน

จากผลการทำแอสเป็กรีแฟคทอริงทำให้ได้เจฮอตดรอว์ ทั้งหมด 12 เวอร์ชั่น ที่จัดการจำนวนซ้ำของเมท็อดคอลแตกต่างกันด้วยแอสเป็ก จำนวนซ้ำของเมท็อดคอลที่ศึกษา ได้แก่ 4, 5, 9, 12, 12, 13, 15, 18, 21, 29, 37 และ 44 เมื่อพิจารณาเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นด้วยมาตรวัดเชิงวัตถุของ Chidamber และ Kemerer (1993) ทั้งหมด 6 มาตรวัด โดยนิยามมาตรวัดเชิงวัตถุพิจารณาเฉพาะคลาสภายในซอร์สโค้ดของซอฟต์แวร์ ผลการทดลองแสดงให้เห็นว่ามาตรวัดเวทเทดเมท็อดเปอคลาส (Weighted Method per Class - WMC) มาตรวัดเดพออฟอินเฮอริแทนทรี (Depth of Inheritance Tree - DIT) และมาตรวัดนัมเบอร์ออฟชิลเดรน (Number of Children - NOC) ทั้งสามมาตรวัดไม่ได้รับผลกระทบจากการทำแอสเป็กรีแฟคทอริง กล่าวคือค่ามาตรวัดของคลาสมีค่าไม่เปลี่ยนแปลงหลังจากจัดการเมท็อดคอลซ้ำกันด้วยแอสเป็ก

ขณะที่มาตรวัดคัพพลิงบีทวีนอ็อบเจกต์คลาส (Coupling Between Object classes - CBO) และมาตรวัดแลคออฟโคฮีชันอินเมท็อด (Lack of Cohesion in Methods - LCOM) ทั้งสองมาตรวัดได้รับผลกระทบจากการทำแอสเป็กรีแฟคทอริง คือหลังจัดการเมท็อดคอลที่ซ้ำกันด้วยแอสเป็กค่ามาตรวัดของคลาสมีค่าเปลี่ยนแปลง จากการวิเคราะห์ผลมาตรวัดพบว่าจำนวนซ้ำของเมท็อดคอลไม่ใช่ปัจจัยที่ส่งผลต่อค่ามาตรวัด CBO และมาตรวัด LCOM ดังนั้นการจัดการจำนวนซ้ำของเมท็อดคอลที่น้อยหรือมากแตกต่างกันจึงไม่ส่งผลให้ค่ามาตรวัด CBO หรือมาตรวัด LCOM มีค่าเพิ่มขึ้นหรือลดลง ค่าเฉลี่ยของทั้งสองมาตรวัดจึงไม่แสดงแนวโน้มหรือความแตกต่างจากการจัดการจำนวนซ้ำของเมท็อดคอลที่แตกต่างกัน จากการวิเคราะห์ปัจจัยที่ส่งผลให้ค่ามาตรวัด CBO เปลี่ยนแปลงพบว่าลักษณะเมท็อดคอลเป็นปัจจัยที่ส่งผลให้ค่ามาตรวัด CBO ของคลาสมีค่าลดลงหลังทำแอสเป็กรีแฟคทอริง โดยผลจากการทดลองในงานวิจัยนี้พบว่าลักษณะเมท็อดคอลที่เรียกใช้งานเมท็อดผ่านชื่อคลาสโดยตรงหรือลักษณะเมท็อดคอลที่เรียกใช้งานเมท็อดผ่านตัวแปรอ็อบเจกต์ของคลาสแม่ หลังจากจัดการเมท็อดคอลลักษณะดังกล่าวด้วยแอสเป็กสามารถลดความสัมพันธ์ระหว่างคลาสที่เกิดจากการเรียกใช้งานเมท็อด ส่งผลให้ค่ามาตรวัด CBO ของคลาสมีค่าลดลง ส่วนมาตรวัด LCOM ของคลาสที่มีค่าเพิ่มขึ้นหลังจากทำแอสเป็กรีแฟคทอริง เนื่องจากหากเมท็อดคอลที่จัดการมีการเข้าถึงแอตทริบิวต์ของคลาส หลังจากจัดการเมท็อดคอลด้วยแอสเป็กจึงทำให้การเข้าถึงแอตทริบิวต์คลาสระหว่างเมท็อดเกิดการเปลี่ยนแปลง จึงส่งผลให้ความเกี่ยวข้องกันระหว่างเมท็อดของคลาสลดลง

จากมาตรวัดเชิงวัตถุทั้งหมด 6 มาตรวัด มีเพียงมาตรวัดเดียวคือมาตรวัดเรสปอนซ์ฟอร์อะคลาส (Response For a Class - RFC) ที่ได้รับผลกระทบจากการทำแอสเป็กรีแฟคทอริงและปัจจัยส่วนหนึ่งที่ทำให้ค่ามาตรวัด RFC ของคลาสเปลี่ยนแปลงเกิดจากจำนวนซ้ำของเมท็อดคอล โดยจากผลมาตรวัดแสดงแนวโน้มค่าเฉลี่ยมาตรวัด RFC ที่ลดลงจากการจัดการเมท็อดคอลซ้ำกันมากกว่า 29 ตำแหน่ง สอดคล้องกับงานวิจัยของ Marin, Deursen, และ Moonen (2007) ที่แนะนำว่าโค้ดซ้ำกันมากกว่า 10 ตำแหน่งเหมาะสมเป็นครอสคัทติ้งคอนเซิร์นที่จะนำมาจัดการด้วยแอสเป็ก และการจัดการเมท็อดคอลในจำนวนซ้ำที่มากขึ้นก็แสดงผลมาตรวัด RFC ในแง่บวก โดยการจัดการจำนวนซ้ำของเมท็อดคอล 37 และ 44 ตำแหน่งแสดงผลค่าเฉลี่ยมาตรวัด RFC ของซอฟต์แวร์ที่ลดลงมาก

จำนวนซ้ำเมท็อดคอลดังกล่าวจึงให้ผลไปในทางเดียวกันกับงานวิจัยของ Hanenberg, Kleinschmager และ Walter (2009) ที่นำเสนอว่าการจัดการโค้ดซ้ำกันมากกว่า 36 ตำแหน่ง การใช้แอสเป็กจะให้ผลเวลาที่ใช้พัฒนาในแง่บวก นอกจากนี้จากผลมาตรวัด RFC ยังพบว่าหากเมท็อดคอลที่ต้องการจัดการมีจำนวนซ้ำของเมท็อดคอลมาก ตำแหน่งของเมท็อดคอลก็อาจกระจัดกระจายอยู่ภายในคลาสที่มากด้วยเช่นกัน ดังผลของเจฮอตดรอว์ เวอร์ชั่น 12 ที่มีจำนวนซ้ำของเมท็อดคอลสูงสุดเท่ากับ 44 ตำแหน่ง และมีตำแหน่งเมท็อดคอลกระจัดกระจายอยู่ภายในคลาสทั้งหมด 43 คลาส จึงมีค่าเฉลี่ยมาตรวัด RFC ลดลงมากสุด เนื่องจากภายในคลาสหนึ่งถึงแม้จะมีตำแหน่งของเมท็อดคอลซ้ำกันมาก หากจัดการเมท็อดคอลทั้งหมดด้วยแอสเป็กแล้วจะส่งผลให้ค่ามาตรวัดของคลาสลดลงเท่ากับ 1 ดังนั้นจากผลการวิเคราะห์ ปัจจัยหลักที่ส่งผลต่อมาตรวัด RFC คือ จำนวนคลาสที่สามารถจัดการเมท็อดคอลซ้ำกันทั้งหมดด้วยแอสเป็กหรือจำนวนคลาสที่ได้รับผลกระทบจากการทำแอสเป็กรีแฟคทอริง หากในซอฟต์แวร์หนึ่งมีจำนวนคลาสที่สามารถจัดการเมท็อดคอลทั้งหมดด้วยแอสเป็กในจำนวนคลาสที่มากจะส่งผลให้ค่ามาตรวัด RFC ลดลงอย่างมีนัยสำคัญเมื่อเทียบกับการจัดการเมท็อดคอลภายในจำนวนคลาสที่น้อย

5.2.2 ผลการเปรียบเทียบมาตรวัดเชิงแอสเป็กของซอฟต์แวร์ที่ปรับปรุงด้วยการทำแอสเป็กรีแฟคทอริงโดยจัดการจำนวนซ้ำของเมท็อดคอลแตกต่างกัน

จากการทำแอสเป็กรีแฟคทอริงทำให้ได้เจฮอตดรอว์ ทั้งหมด 12 เวอร์ชั่น ที่จัดการจำนวนซ้ำของเมท็อดคอลแตกต่างกันด้วยแอสเป็ก จำนวนซ้ำของเมท็อดคอลที่ศึกษา ได้แก่ 4, 5, 9, 12, 12, 13, 15, 18, 21, 29, 37 และ 44 เมื่อพิจารณาเจฮอตดรอว์ทั้ง 12 เวอร์ชั่นด้วยมาตรวัดเชิงแอสเป็กของ Ceccato และ Tonella (2004) ทั้งหมด 10 มาตรวัด โดยนิยามของมาตรวัดเชิงแอสเป็กพิจารณาทั้งคลาสและแอสเป็กภายในซอร์สโค้ดของซอฟต์แวร์ ผลการทดลองแสดงให้เห็นว่ามาตรวัดเดพออฟอินเฮอริแทนทรี (Depth of Inheritance Tree - DIT) มาตรวัดนัมเบอร์ออฟชิลเดรน (Number of Children - NOC) และมาตรวัดแลคออฟโคฮีชันอินโอเปอร์เรชั่น (Lack of Cohesion in Operations - LCO) ทั้งสามมาตรวัดไม่ได้รับผลกระทบจากการทำแอสเป็กรีแฟคทอริง กล่าวคือค่ามาตรวัดของคลาสมีค่าไม่เปลี่ยนแปลงหลังจากจัดการเมท็อดคอลซ้ำกันด้วยแอสเป็ก

ส่วนมาตรวัดเชิงแอสเป็กอีก 7 มาตรวัด ที่ได้รับผลกระทบจากการทำแอสเป็กรีแฟคทอริง โดยหลังทำแอสเป็กรีแฟคทอริงค่ามาตรวัดมีค่าเปลี่ยนแปลง จากการวิเคราะห์มาตรวัดเชิงแอสเป็กทั้ง 7 มาตรวัด การจัดการเมท็อดคอลในจำนวนซ้ำเมท็อดคอลที่น้อยหรือมากไม่ส่งผลต่อค่ามาตรวัดทั้ง 7 มาตรวัด ดังนั้นจำนวนซ้ำของเมท็อดคอลจึงไม่ใช่ปัจจัยที่ทำให้ค่ามาตรวัดเกิดการเปลี่ยนแปลงในการทำแอสเป็กรีแฟคทอริง ดังนั้นจึงสามารถอธิบายปัจจัยที่ส่งผลต่อค่ามาตรวัดทั้ง 7 มาตรวัดได้ดังนี้

มาตรวัดเวทเทดโอเปอร์เรชั่นอินมอดูล (Weighted Operations in Module - WOM) จากผลการทดลองพบว่าปัจจัยที่ส่งผลให้ค่าเฉลี่ยมาตรวัด WOM ของเจฮอตดรอว์มีค่าเพิ่มขึ้น คือจำนวนแอดไวซ์ของแอสเป็กในเจฮอตดรอว์แต่ละเวอร์ชั่นที่สร้างหลังการทำแอสเป็กรีแฟคทอริง ดังนั้นหากภายในแอสเป็กมีการอิมพลีเม้นต์แอดไวซ์ในจำนวนที่น้อยอาจส่งผลให้ค่าเฉลี่ยมาตรวัด WOM เพิ่มสูงขึ้นไม่มากนัก

มาตรวัดคัพพลิงออนฟิลด์แอคเซส (Coupling on Field Access - CFA) จากผลการทดลองพบว่าส่วนใหญ่การจัดการเมท็อดคอลซ้ำกันด้วยแอสเป็กไม่ส่งผลต่อค่ามาตรวัด CFA ของคลาส แต่หากพิจารณาแอสเป็กซึ่งเป็นส่วนที่สร้างขึ้นมาใหม่ดังในเจฮอตดรอว์ เวอร์ชั่น 9 เนื่องจากในแอสเป็กมีการเข้าถึงฟิลด์ของคลาสอื่นโดยตรงทั้งหมด 2 คลาส จึงส่งผลให้ค่าเฉลี่ยมาตรวัด CFA ของเจฮอตดรอว์ เวอร์ชั่น 9 มีค่าสูง ดังนั้นจำนวนการเข้าถึงแอตทริบิวต์ของคลาสอื่นเป็นปัจจัยที่ส่งผลต่อมาตรวัด CFA

มาตรวัดคัพพลิงออนเมท็อดคอล (Coupling on Method Call - CMC) และมาตรวัดคัพพลิงบีทวีนมอดูล (Coupling between Modules - CBM) ทั้งสองมาตรวัดเป็นมาตรวัดที่เกี่ยวข้องกับคัพพลิง และมีส่วนของนิยามที่เหมือนกันคือพิจารณาความสัมพันธ์ระหว่างคลาสที่เกิดจากการเรียกใช้งานเมท็อด จากผลการทดลองจึงพบปัจจัยที่ส่งผลต่อมาตรวัดทั้งสองมาตรวัดเหมือนกันคือลักษณะเมท็อดคอล โดยจากงานวิจัยนี้พบว่าลักษณะเมท็อดคอลที่เรียกใช้งานเมท็อดผ่านชื่อคลาสโดยตรง หากจัดการเมท็อดคอลลักษณะดังกล่าวด้วยแอสเป็กจะสามารถลดความสัมพันธ์ระหว่างคลาสได้ ส่งผลให้ค่ามาตรวัดของคลาสมีค่าลดลง ผลการทดลองจึงแสดงในทิศทางเดียวกันคือเจฮอตดรอว์ เวอร์ชั่น 8 ที่จัดการเมท็อดคอลทั้งหมด 18 ตำแหน่ง และเมท็อดคอลทั้ง 18 ตำแหน่งมีลักษณะเมท็อดคอลที่เรียกใช้งานเมท็อดผ่านชื่อคลาสโดยตรงทั้งหมด หลังการทำแอสเป็กรีแฟคทอริงค่าเฉลี่ยมาตรวัด CMC และ CBM จึงมีค่าลดลงมากสุด

มาตรวัดครอสคัทติ้งดีกรีออฟแอนแอสเป็ก (Crosscutting Degree of an Aspect - CDA) และมาตรวัดคัพพลิงออนแอดไวซ์เอ็กซิคิวชั่น (Coupling on Advice Execution - CAE) พบว่าจากการทดลองปัจจัยที่ส่งผลต่อมาตรวัด CDA และ CAE คือจำนวนคลาสที่แอสเป็กสามารถเข้าไปขัดขวางการทำงาน โดยแสดงให้เห็นถึงความสามารถของแอสเป็กในการแทรกการทำงานของแอดไวซ์ไปยังคลาสต่างๆ ดังนั้นเจฮอตดรอว์ เวอร์ชั่น 12 ที่จัดการเมท็อดคอลซ้ำกัน 44 ตำแหน่ง มีจำนวนคลาสที่แอสเป็กสามารถเข้าไปขัดขวางการทำงานมากสุดถึง 44 คลาส ส่งผลให้ค่าเฉลี่ยของมาตรวัด CDA และ CAE มีค่ามากสุดเหมือนกันเมื่อเทียบกับเจฮอตดรอว์เวอร์ชั่นอื่นๆ

มาตรวัดเรสปอนซ์ฟอร์อะมอดูล (Response For a Module - RFM) พบว่าจากการทดลองปัจจัยที่ส่งผลต่อค่ามาตรวัด RFM คือจำนวนแอดไวซ์ที่สามารถขัดขวางการทำงานของคลาส เนื่องจากการทำแอสเป็กรีแฟคทอริงสำหรับจัดการเมท็อดคอลที่ซ้ำกันภายในคลาสหนึ่งคลาส จะส่งผลให้จำนวนเมท็อดที่ทำงานตอบสนองต่อคลาสลดลงเพียงหนึ่งเมท็อด ดังนั้นเมื่อพิจารณาด้วยมาตรวัด RFM ที่เพิ่มเติมนิยามการพิจารณาแอสเป็กเข้ามา จึงทำให้ค่ามาตรวัด RFM ของคลาสมีการนับจำนวนแอดไวซ์ที่เข้ามาขัดขวางการทำงานคลาสด้วย ค่ามาตรวัด RFM ของคลาสส่วนใหญ่จึงมีค่าเพิ่มขึ้นหลังจากทำแอสเป็กรีแฟคทอริง

5.2.3 ผลการวิเคราะห์คุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริงที่จัดการจำนวนซ้ำของเมท็อดคอลแตกต่างกัน

การวิเคราะห์คุณภาพของซอฟต์แวร์พิจารณาจากความสัมพันธ์ระหว่างคุณภาพซอฟต์แวร์และมาตรวัด โดยมาตรวัดที่พิจารณามีทั้งมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็ก แสดงรายละเอียดแยกตามชุดมาตรวัดได้ดังนี้

ความสัมพันธ์ระหว่างคุณภาพซอฟต์แวร์และมาตรวัดเชิงวัตถุทั้ง 6 มาตรวัดของ Chidamber และ Kemerer (1993) สามารถอ้างอิงจากงานวิจัยของ Kulkarni, Kalshetty และ Arde (2010) โดยคุณภาพซอฟต์แวร์ที่สามารถแสดงความสัมพันธ์กับมาตรวัด ได้แก่ ความสามารถในการทำความเข้าใจ (Understandability) ความสามารถการบำรุงรักษา (Maintainability) ความสามารถการนำกลับมาใช้ใหม่ (Reusability) และความสามารถการทดสอบ (Testability) จากการทำแอสเป็กรีแฟคทอริงสามารถแสดงผลคุณภาพซอฟต์แวร์ได้ดังตารางที่ 5.3 โดยจากมาตรวัดเชิงวัตถุทั้งหมด 6 มาตรวัด มีเพียงสองมาตรวัดแสดงผลคุณภาพซอฟต์แวร์ที่ปรับปรุง

ตารางที่ 5.3 ผลคุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริงพิจารณาด้วยมาตรวัดเชิงวัตถุ

ลำดับ	มาตรวัดเชิงวัตถุ	Understandability	Maintainability	Reusability	Testability	ผลการทำแอสเป็กรีแฟคทอริงต่อคุณภาพซอฟต์แวร์
1	Weighted Method per Class (WMC)	-	✓	✓	-	—
2	Depth of Inheritance Tree (DIT)	✓	-	✓	-	—
3	Number of Children (NOC)	-	-	✓	✓	—
4	Coupling Between Object classes (CBO)	✓	✓	✓	✓	↑
5	Response For a Class (RFC)	✓	-	-	✓	↑
6	Lack of Cohesion in Methods (LCOM)	-	-	-	-	—

มาตรวัดคัพพลิงบีทวีนอ็อบเจกต์คลาส (Coupling Between Object classes - CBO) เนื่องด้วยจากการวิเคราะห์ผลมาตรวัด CBO พบว่าจำนวนซ้ำของเมท็อดคอลไม่ใช่ปัจจัยที่ส่งผลให้ค่ามาตรวัด CBO เปลี่ยนแปลงหลังจากทำแอสเป็กรีแฟคทอริง ดังนั้นผลคุณภาพซอฟต์แวร์ที่ปรับปรุงทั้งสี่ด้านจึงไม่ได้เกิดจากการจัดการเมท็อดคอลในจำนวนซ้ำที่มากหรือน้อยด้วยแอสเป็ก โดยลักษณะเมท็อดคอลที่จัดการด้วยแอสเป็กเป็นปัจจัยที่สามารถส่งผลให้ค่ามาตรวัด CBO ของคลาสลดลง จากความสัมพันธ์ระหว่างมาตรวัด CBO และคุณภาพซอฟต์แวร์ แสดงให้เห็นว่าหากค่ามาตรวัด CBO ของคลาสลดลงจะส่งผลให้คุณภาพซอฟต์แวร์ทั้งสี่ด้านปรับปรุงดีขึ้น ได้แก่ ความสามารถในการทำความเข้าใจ ความสามารถการบำรุงรักษา ความสามารถการนำกลับมาใช้ใหม่ และความสามารถการทดสอบ เนื่องจากมาตรวัด CBO เป็นมาตรวัดที่เกี่ยวข้องกับคัพพลิงแสดงถึงการขึ้นต่อกันระหว่างคลาส ดังนั้นการทำแอสเป็กรีแฟคทอริงสามารถช่วยลดความสัมพันธ์ระหว่างคลาสได้เมื่อพิจารณาลักษณะเมท็อดคอลที่เหมาะสมและช่วยให้การขึ้นต่อกันระหว่างคลาสลดลง

มาตรวัดเรสปอนซ์ฟอร์อะคลาส (Response For a Class - RFC) จากการวิเคราะห์ผลมาตรวัด RFC พบว่าจำนวนซ้ำของเมท็อดคอลและจำนวนคลาสที่สามารถจัดการเมท็อดคอลทั้งหมดด้วยแอสเป็กมีผลต่อมาตรวัด RFC กล่าวได้ว่าการจัดการเมท็อดคอลในจำนวนซ้ำที่มากและมีตำแหน่งของเมท็อดคอลที่กระจายอยู่ภายในคลาสจำนวนมากด้วย ส่งผลให้หลังทำแอสเป็กรีแฟคทอริงมีคลาสที่มีค่ามาตรวัดลดลงจำนวนมากและค่าเฉลี่ยมาตรวัด RFC จึงมีค่าลดลงมากเช่นกัน จากความสัมพันธ์ระหว่างมาตรวัด RFC และคุณภาพซอฟต์แวร์ หากค่ามาตรวัด RFC ของคลาสมีค่าลดลงจะส่งผลให้คุณภาพซอฟต์แวร์ทั้งสองด้านปรับปรุงดีขึ้น ได้แก่ ความสามารถในการทำความเข้าใจและความสามารถการทดสอบ ดังนั้นการทำแอสเป็กรีแฟคทอริงที่จัดการจำนวนซ้ำเมท็อดคอลในจำนวนซ้ำที่มากและตำแหน่งของเมท็อดคอลกระจัดกระจายอยู่ในหลายคลาส การทำแอสเป็กรีแฟคทอริงสามารถช่วยปรับปรุงความสามารถในการทำความเข้าใจและความสามารถการทดสอบของซอฟต์แวร์ โดยช่วยลดจำนวนเมท็อดที่ทำงานเพื่อตอบสนองต่อคลาส จำนวนเมท็อดที่ต้องทำความเข้าใจการทำงานจะลดลง และช่วยลดจำนวนเมท็อดที่ต้องทดสอบความถูกต้องของการเรียกใช้งานจากในคลาส

ความสัมพันธ์ระหว่างคุณภาพซอฟต์แวร์และมาตรวัดเชิงแอสเป็กของ Ceccato และ Tonella (2004) โดยสามารถนำมาใช้ประเมินคุณภาพได้เพียงบางมาตรวัด ได้แก่ มาตรวัด WOM, RFM, CBM และ LCO สามารถอ้างอิงจากงานวิจัยของ Sirbi และ Kulkarni (2013) โดยคุณภาพซอฟต์แวร์ที่สามารถแสดงความสัมพันธ์กับมาตรวัด ได้แก่ ประสิทธิภาพ (Efficiency) ความซับซ้อน (Complexity) ความสามารถในการทำความเข้าใจ (Understandability) ความสามารถการบำรุงรักษา (Maintainability) ความสามารถการนำกลับมาใช้ใหม่ (Reusability) และความสามารถการทดสอบ (Testability) จากการทำแอสเป็กรีแฟคทอริงสามารถแสดงผลคุณภาพซอฟต์แวร์ได้ดังตารางที่ 5.4 โดยผลแสดงว่ามีเพียงสามมาตรวัดที่มีคุณภาพซอฟต์แวร์เปลี่ยนแปลงหลังทำแอสเป็กรีแฟคทอริง

ตารางที่ 5.4 ผลคุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริงพิจารณาด้วยมาตรวัดเชิงแอสเป็ก

ลำดับ	มาตรวัดเชิงแอสเป็ก	Efficiency	Complexity	Understandability	Maintainability	Reusability	Testability	ผลการทำแอสเป็กรีแฟคทอริงต่อคุณภาพซอฟต์แวร์
1	Weighted Operations in Module (WOM)	-	✓	-	✓	-	-	↓
2	Coupling between Modules (CBM)	-	-	-	✓	✓	-	↑
3	Response For a Module (RFM)	-	-	✓	-	-	✓	↓
4	Lack of Cohesion in Operations (LCO)	✓	-	-	-	✓	-	—

มาตรวัดเวทเทดโอเปอร์เรชั่นอินมอดูล (Weighted Operations in Module - WOM) จากการวิเคราะห์ผลมาตรวัด WOM พบว่าจำนวนซ้ำของเมท็อดคอลไม่ใช่ปัจจัยที่ส่งผลให้ค่ามาตรวัด WOM เพิ่มขึ้นหลังทำแอสเป็กรีแฟคทอริง ดังนั้นจำนวนซ้ำเมท็อดคอลจึงไม่ได้ส่งผลต่อคุณภาพซอฟต์แวร์จากการพิจารณาด้วยมาตรวัด WOM โดยจำนวนแอดไวซ์ที่อิมพลีเม้นต์ภายในแอสเป็กเป็นปัจจัยที่ส่งผลให้ค่ามาตรวัด WOM เพิ่มขึ้น จากความสัมพันธ์ระหว่างมาตรวัด WOM และคุณภาพซอฟต์แวร์ แสดงให้เห็นว่าหากค่ามาตรวัด WOM เพิ่มขึ้นจะส่งผลต่อคุณภาพซอฟต์แวร์ทั้งสองด้าน ได้แก่ ความซับซ้อนและความสามารถการบำรุงรักษา เนื่องจากหากมีการเปลี่ยนแปลงชื่อเมท็อดหรือลำดับของคำสั่งภายในคลาสอาจส่งผลกระทบต่อพอยท์คัทและแอดไวซ์ ได้แก่ การแก้ไขพอยท์คัทของแต่ละแอดไวซ์เพื่อระบุไปยังตำแหน่งของคลาสที่ต้องการให้โค้ดทำงานให้ถูกต้อง หรือการแก้ไขประเภทแอดไวซ์ เป็นต้น

มาตรวัดคัพพลิงบีทวีนมอดูล (Coupling between Modules - CBM) จากการวิเคราะห์ผลมาตรวัด CBM พบว่าจำนวนซ้ำของเมท็อดคอลไม่ใช่ปัจจัยที่ส่งผลให้ค่ามาตรวัด CBM เปลี่ยนแปลงหลังจากทำแอสเป็กรีแฟคทอริง ดังนั้นผลคุณภาพซอฟต์แวร์ที่ปรับปรุงจึงไม่ได้เกิดจากการจัดการเมท็อดคอลในจำนวนซ้ำที่มากหรือน้อยด้วยแอสเป็ก โดยลักษณะเมท็อดคอลที่จัดการด้วยแอสเป็กเป็นปัจจัยที่สามารถส่งผลให้ค่ามาตรวัด CBM ของคลาสลดลง จากความสัมพันธ์ระหว่างมาตรวัด CBM และคุณภาพซอฟต์แวร์ แสดงให้เห็นว่าหากค่ามาตรวัด CBM ของคลาสลดลงจะส่งผลให้คุณภาพซอฟต์แวร์ทั้งสองด้านปรับปรุงดีขึ้น ได้แก่ ความสามารถการบำรุงรักษาและความสามารถการนำกลับมาใช้ใหม่

มาตรวัดเรสปอนซ์ฟอร์อะมอดูล (Response For a Module - RFM) จากการวิเคราะห์ผลมาตรวัด RFM พบว่าจำนวนซ้ำของเมท็อดคอลไม่ใช่ปัจจัยที่ส่งผลให้ค่ามาตรวัด RFM เปลี่ยนแปลงหลังจากทำแอสเป็กรีแฟคทอริง ดังนั้นผลคุณภาพซอฟต์แวร์ที่ลดลงจึงไม่ได้เกิดจากการจัดการเมท็อดคอลในจำนวนซ้ำที่มากหรือน้อยด้วยแอสเป็ก โดยจำนวนแอดไวซ์ภายในแอสเป็กที่เข้ามาขัดขวางการทำงานของคลาสเป็นปัจจัยที่ทำให้ค่ามาตรวัด RFM ของคลาสมีค่าเพิ่มขึ้น จากความสัมพันธ์ระหว่างมาตรวัด RFM และคุณภาพซอฟต์แวร์ แสดงให้เห็นว่าหากค่ามาตรวัด RFM เพิ่มขึ้นจะส่งผลต่อคุณภาพซอฟต์แวร์ทั้งสองด้าน ได้แก่ ความสามารถในการทำความเข้าใจและความสามารถการทดสอบทำให้การทำความเข้าใจการทำงานของแต่ละคลาสยากขึ้น เนื่องจากต้องศึกษาการทำงานของแอดไวซ์ในแอสเป็กที่เข้ามาขัดขวางการทำงานของคลาส นอกจากนี้ยังส่งผลต่อการทดสอบทำให้ต้องตรวจสอบการทำงานระหว่างคลาสและแอสเป็กด้วย

จากการพิจารณาด้วยมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็กพบว่า มาตรวัดที่เกี่ยวข้องกับคัพพลิง ได้แก่ มาตรวัดคัพพลิงบีทวีนอ็อบเจกต์คลาส (Coupling Between Object classes - CBO) ของมาตรวัดเชิงวัตถุ และมาตรวัดคัพพลิงบีทวีนมอดูล (Coupling between Modules - CBM) ของมาตรวัดเชิงแอสเป็ก แสดงผลคุณภาพซอฟต์แวร์ที่ปรับปรุงหลังจากการทำแอสเป็กรีแฟคทอริงในทิศทางเดียวกัน โดยคุณภาพซอฟต์แวร์ที่ปรับปรุงเหมือนกัน ได้แก่ ความสามารถการบำรุงรักษา และความสามารถการนำกลับมาใช้ใหม่ โดยปัจจัยที่ส่งผลให้สามารถลดจำนวนการขึ้นต่อกันระหว่างคลาสคือลักษณะเมท็อดคอล จากการทดลองในงานวิจัยนี้การจัดการเมท็อดคอลที่มีลักษณะการเรียกใช้งานเมท็อดผ่านชื่อคลาสโดยตรงสามารถช่วยลดความสัมพันธ์ระหว่างคลาสได้มาก และจากผลคุณภาพด้านความสามารถการบำรุงรักษาและความสามารถการนำกลับมาใช้ใหม่ที่ปรับปรุงขึ้นหลังจากการนำแอสเป็กเข้ามาใช้จัดการเมท็อดคอลที่ซ้ำกัน จากผลดังกล่าวแสดงผลสอดคล้องกับงานวิจัยในอดีตที่แสดงให้เห็นว่าการนำแอสเป็กเข้ามาใช้ปรับปรุงซอร์สโค้ดสามารถช่วยปรับปรุงคุณภาพซอฟต์แวร์ด้านความสามารถการบำรุงรักษา (Kulesza et al., 2006 ; Kumar et al., 2007 ; Mguni and Ayalew, 2013) และความสามารถการนำกลับมาใช้ใหม่ (Sant'Anna et al., 2003)

นอกจากนี้ยังพบว่าจากการพิจารณาด้วยมาตรวัดเรสปอนซ์ฟอร์อะคลาส (Response For a Class - RFC) ที่พิจารณาแต่เพียงนิยามของคลาส โดยถึงแม้การทำแอสเป็กรีแฟคทอริงเพื่อจัดการเมท็อดคอลที่ซ้ำกันด้วยแอสเป็กจะสามารถช่วยลดจำนวนเมท็อดที่ทำงานตอบสนองต่อคลาสได้ แต่เมื่อพิจารณาผลจากการใช้แอสเป็กด้วยมาตรวัดเรสปอนซ์ฟอร์อะมอดูล (Response For a Module - RFM) กลับพบจำนวนแอดไวซ์ที่เข้ามาขัดขวางการทำงานของคลาสในจำนวนที่มากกว่าเมื่อเทียบกับจำนวนเมท็อดที่ลดลง ดังนั้นการนำแอสเป็กมาใช้จัดการโค้ดโคลนจึงส่งผลให้คุณภาพด้านความสามารถการทำความเข้าใจและความสามารถการทดสอบลดลง เพราะต้องใช้ความพยายามในการทำความเข้าใจการทำงานของแอสเป็กเพิ่มขึ้น และต้องเพิ่มการตรวจสอบการทำงานระหว่างคลาสกับแอสเป็ก

5.3 การนำงานวิจัยไปประยุกต์ใช้

ข้อค้นพบจากการศึกษาในงานวิจัยนี้สามารถช่วยต่อยอดองค์ความรู้สำหรับการนำการโปรแกรมเชิงแอสเป็กมาประยุกต์ใช้กับซอฟต์แวร์ที่พัฒนาด้วยการโปรแกรมเชิงวัตถุ โดยปรับปรุงซอร์สโค้ดจากการทำแอสเป็กรีแฟคทอริง ผลจากการทดลองอาจสามารถบอกเพียงแนวทางของการนำแอสเป็กมาใช้กับลักษณะของโค้ดที่เหมาะสมเพื่อให้ได้คุณภาพซอฟต์แวร์ที่ดีขึ้น มีรายละเอียดดังนี้

1. จากการจัดการปัญหาโค้ดกระจัดกระจายโดยเลือกจัดการเมท็อดคอลซ้ำกันด้วยแอสเป็ก ผู้วิจัยพบว่าถึงแม้เมท็อดคอลจะมีตำแหน่งที่ซ้ำภายในซอร์สโค้ดจำนวนมาก การนำแอสเป็กเข้ามาจัดการเมท็อดคอลที่ซ้ำกันเหล่านี้ส่วนใหญ่ไม่ส่งผลต่อทั้งมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็ก ยกเว้นมาตรวัดเรสปอนซ์ฟอร์อะคลาส (Response For a Class - RFC) ของมาตรวัดเชิงวัตถุที่ทั้งจำนวนซ้ำของเมท็อดคอลและจำนวนคลาสทั้งหมดที่สามารถจัดการเมท็อดคอลมีผลต่อมาตรวัด RFC ทำให้คุณภาพซอร์ฟแวร์ด้านความสามารถในการทำความเข้าใจและความสามารถการทดสอบปรับปรุงดีขึ้น ดังนั้นการเลือกปรับปรุงซอร์สโค้ดที่มีโค้ดซ้ำกันในหลายตำแหน่งซึ่งกระจายอยู่ภายในคลาสจำนวนมาก การปรับปรุงด้วยแอสเป็กสามารถส่งผลให้คุณภาพดีขึ้น

2. จากมาตรวัดที่เกี่ยวข้องกับคัพพลิงทั้งในมาตรวัดเชิงวัตถุและมาตรวัดเชิงแอสเป็ก ได้แก่ มาตรวัดคัพพลิงบีทวีนอ็อบเจกต์คลาส (Coupling Between Object classes - CBO) และมาตรวัดคัพพลิงบีทวีนมอดูล (Coupling between Modules - CBM) ตามลำดับ ผลจากการวิเคราะห์การจัดการเมท็อดคอลที่ซ้ำกันด้วยแอสเป็กจากทั้งสองมาตรวัด พบว่าลักษณะของเมท็อดคอลที่จัดการด้วยแอสเป็กมีผลต่อคัพพลิงของซอฟต์แวร์ โดยจากการทดลองในงานวิจัยนี้พบว่าการเลือกจัดการเมท็อดคอลที่มีลักษณะเมท็อดคอลเป็นการเรียกใช้งานเมท็อดผ่านชื่อคลาสโดยตรง การจัดการเมท็อดคอลลักษณะนี้สามารถช่วยลดความสัมพันธ์ระหว่างคลาสที่เกิดจากการเรียกใช้งานเมท็อดได้ ซึ่งเหมาะสมต่อการนำแอสเป็กเข้ามาใช้ปรับปรุงซอร์สโค้ด ช่วยปรับปรุงคุณภาพของซอฟต์แวร์ด้านความสามารถการบำรุงรักษา และความสามารถการนำกลับมาใช้ใหม่ให้มีคุณภาพดีมากขึ้น

3. ผลจากการวิเคราะห์มาตรวัดเชิงแอสเป็ก พบว่าจำนวนแอดไวซ์ภายในแอสเป็กเป็นปัจจัยที่กระทบในหลายมาตรวัด ได้แก่ มาตรวัดเวทเทดโอเปอร์เรชั่นอินมอดูล (Weighted Operations in Module - WOM) และมาตรวัดเรสปอนซ์ฟอร์อะมอดูล (Response For a Module - RFM) ส่งผลให้มีค่ามาตรวัดเพิ่มขึ้นหลังทำแอสเป็กรีแฟคทอริงเพื่อปรับปรุงซอร์สโค้ด ดังนั้นการปรับปรุงซอร์สโค้ดโดยการใช้แอดไวซ์ภายในแอสเป็กจำนวนน้อย และมีจำนวนแอดไวซ์เข้าไปขัดขวางการทำงานของคลาสในจำนวนที่น้อยเช่นกัน สามารถช่วยให้การนำแอสเป็กเข้ามาปรับปรุงซอร์สโค้ดไม่ส่งผลให้ความซับซ้อนภายในซอร์สโค้ดเพิ่มมาก และอาจช่วยปรับปรุงคุณภาพซอฟต์แวร์ได้

5.4 ข้อจำกัดของงานวิจัยและข้อเสนอแนะ

1. การศึกษานี้เป็นการศึกษาการจัดการโค้ดโคลนด้วยแอสเป็ก โดยงานวิจัยนี้สนใจเมท็อดคอลซ้ำกันภายในซอร์สโค้ดของซอฟต์แวร์ เพื่อแสดงคุณลักษณะของการแก้ไขโค้ดกระจัดกระจาย ดังนั้นเมท็อดคอลที่ซ้ำกันอาจยังไม่สะท้อนถึงลักษณะของครอสคัทติ้งคอนเซิร์นอย่างแท้จริง ซึ่งงานวิจัยต่างๆได้นำเสนอการใช้แอสเป็กสำหรับจัดการครอสคัทติ้งคอนเซิร์น ดังนั้นการศึกษาลักษณะของครอสคัทติ้งคอนเซิร์นอื่นๆที่เหมาะสมต่อการนำแอสเป็กมาใช้เพื่อปรับปรุงคุณภาพซอฟต์แวร์จึงเป็นข้อเสนอแนะสำหรับการศึกษาในอนาคต

2. จากความซับซ้อนของโค้ดในเจฮอตดรอว์ เวอร์ชั่น 7.1 ซึ่งเป็นซอฟต์แวร์ขนาดใหญ่และข้อจำกัดของการใช้ภาษาแอสเป็กเจ ทำให้ไม่สามารถจัดการเมท็อดคอลซ้ำกันทั้งหมดที่มีด้วยแอสเป็กได้ และไม่สามารถกำหนดพอยท์คัทได้ครอบคลุมทุกตำแหน่งของเมท็อดคอลที่ซ้ำกัน อาจทำให้ผลมาตรวัดที่ได้ไม่สะท้อนต่อผลคุณภาพซอฟต์แวร์อย่างแท้จริง จึงอาจเลือกกรณีศึกษาที่มีความซับซ้อนของโค้ดน้อยและเลือกใช้ภาษาโปรแกรมเชิงแอสเป็กอื่นๆที่รองรับสำหรับการเขียนแอสเป็ก เพื่อให้สามารถสะท้อนผลคุณภาพซอฟต์แวร์จากการทำแอสเป็กรีแฟคทอริง รวมถึงสามารถแนะนำภาษาโปรแกรมเชิงแอสเป็กสำหรับเป็นทางเลือกการเลือกใช้ภาษาที่เหมาะสมต่อการทำแอสเป็กรีแฟคทอริง

3. การศึกษานี้เป็นการทดลองกับกรณีศึกษาเพียงกรณีเดียวคือเจฮอตดรอว์ เวอร์ชั่น 7.1 เพื่อศึกษาข้อมูลเบื้องต้น (Exploratory Research) ผลการทดลองจึงอาจไม่สามารถอ้างอิงต่อการนำไปใช้จริงกับการปรับปรุงซอร์สโค้ดของซอฟต์แวร์ในเชิงธุรกิจ ดังนั้นการศึกษาโดยใช้หลายกรณีศึกษาเพิ่มขึ้นอาจให้ผลการทดลองที่สามารถนำมาประยุกต์ใช้จริงได้

รายการอ้างอิง

Abreu, F. B. (1995). Design Metrics for Object-Oriented Software Systems. Paper presented at the ECOOP'95 Quantitative Methods Workshop.

Aggarwal, K. K., Singh, Y., Kaur, A., and Malhotra, R. (2006). Empirical Study of Object-Oriented Metrics. Journal of Object Technology, *5*, 149-173.

Ahuja, R., and Goel, A. (2008). A Study Of Applications Of Aspect Oriented Programming. Paper presented at the 2nd National Conference on Challenges & Opportunities in Information Technology, Mandi Gobindgarh.

Al-Jamimi, H. A., Alshayeb, M., and Elish, M. O. (2011). Investigating the Effect of Aspect-Oriented Refactoring on Software Maintainability. Springer-Verlag Berlin Heidelberg.

Almasri, A., and Albayouk, I. (2006). Experiences In Migrating An Industrial Application To Aspects. (Master), Vrije Universiteit Brussel.

Bhatti, M. U. (2008). Object Identification and Aspect Mining in Procedural Object-Oriented Code. (Doctoral dissertation), Universite Paris.

Binkley, D., Ceccato, M., Harman, M., Ricca, F., and Tonella, P. (2006). Tool-Supported Refactoring of Existing Object-Oriented Code into Aspects. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, *32*, 698-717.

Boehm, B. W., Brown, J. R., and Lipow, M. (1978). Quantitative Evaluation of Software Quality. Paper presented at the the 2nd international conference on Software engineering.

Breu, S., and Krinke, J. (2004). Aspect mining using event traces. Paper presented at the Automated Software Engineering (ASE 2004).

Bruntink, M., Deursen, A. V., Engelen, R. V., and Tourw´e, T. (2004). An evaluation of clone detection techniques for identifying crosscutting concerns. Paper presented at the the IEEE International Conference on Software Maintenance (ICSM).

Cavallaro, L., and Miraz, M. (2010). Refactoring object oriented software: cross-cutting concerns identification and isolation.

Ceccato, M., et al. (2006). Applying and Combining Three Different Aspect Mining Techniques. Delft University of Technology.

Ceccato, M., and Tonella, P. (2004). Measuring the Effects of Software Aspectization. Paper presented at the 1st Workshop on Aspect Reverse Engineering (WARE).

Chidamber, S. R., and Kemerer, C. F. (1993). A Metrics suite for Object oriented design. M.I.T. Sloan School of Management.

Elish, K. O., and Alshayeb, M. (2012). Using Software Quality Attributes to Classify Refactoring to Patterns. Journal of Software, *7*, 408-419.

Fenton, N. E., and Pfleeger, S. L. (1998). Software Metrics: A Rigorous & Practical Approach (2nd Ed.): PWS Publishing Company.

Filman, R. E., Elrad, T., Clarke, S., and Mehmet, A. (2005). Aspect-Oriented Software Development. Boston: Addison-Wesley.

Fitzpatrick, R. (1996). Software Quality: Definitions and Strategic Issues. Staffordshire University.

Garvin, D. (1984). What does Product Quality Really Mean? Sloan Management Review, 25-43.

Grady, R., and Caswell, D. L. (1987). Software Metrics: Establishing a Company-wide Program: Prentice Hall.

Griswold, W., Kato, Y., and Yuan, J. (1999). Aspect browser: Tool support for managing dispersed aspects. Paper presented at the Multi-Dimensional Separation of Concerns in Object-oriented Systems.

Hanenberg, S., Kleinschmager, S., and Walter, M. J. (2009). Does Aspect-Oriented Programming Increase the Development Speed for Crosscutting Code? An Empirical Study. Paper presented at the the 2009 3rd International Symposium on Empirical Software Engineering and Measurement.

Hannemann, J., and Kiczales, G. (2001). Overcoming the prevalent decomposition of legacy code. Paper presented at the Workshop on Advanced Separation of Concerns at the International Conference on Software Engineering (ICSE).

He, L., Bai, H., Zhang, J., and Hu, C. (2005). Amuca algorithm for aspect mining. Paper presented at the SEKE.

Hilliard, R. (2002). IEEE Recommended Practice for Architectural Description of Software-Intensive Systems. IEEE Computer Society, 1-23.

Immonen, M. (2009). Software Product Quality Analysis System. (Master's Thesis), TAMK University of Applied Science.

Ince, D. (1994). ISO 9001 and software quality assurance. England McGraw-Hill.

ISO/IEC9126-1. (2001). ISO/IEC 9126-1 Software engineering Product quality Part 1: Quality Model.

Kamiya, T., Kusumoto, S., and Inoue, K. (2002). CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 654-670.

Kan, S. H. (2002). Metrics and Models in Software Quality Engineering (Second ed.). USA: Addison-Wesley Longman Publishing.

Kapser, C. J. (2009). Toward an Understanding of Software Code Cloning as a Development Practice. (Doctoral dissertation), the University of Waterloo.

Kaur, A., and Johari, K. (2009). Identification of Crosscutting Concerns: A Survey. International Journal of Engineering Science and Technology, 166-172.

Kellens, A., Mens, K., and Tonella, P. (2007). A Survey of Automated Code-Level Aspect Mining Techniques. Transactions on aspect-oriented software development IV, 143-162.

Kiczales, G., et al. (2001). An Overview of AspectJ. Paper presented at the European Conference on Object-Oriented Programming.

Kiczales, G., et al. (1997). Aspect-Oriented Programming. Paper presented at the the European Conference on Object-Oriented Programming (ECOOP), Finland.

Kulesza, U., et al. (2006). Quantifying the Effects of Aspect-Oriented Programming:A Maintenance Study. Paper presented at the In 22nd IEEE International Conference on Software Maintenance.

Kulkarni, U. L., Kalshetty, Y. R., and Arde, V. G. (2010). Validation of CK metrics for Object-Oriented Design Measurement. Paper presented at the Third International Conference on Emerging Trends in Engineering and Technology.

Kumar, A., Grover, P. S., and Kumar, R. (2009). A Quantitative Evaluation of Aspect-Oriented Software Quality Model (AOSQUAMO). ACM SIGSOFT Software Engineering Notes, *34*, 1-9.

Kumar, A., Kumar, R., and Grover, P. S. (2007). An Evaluation of Maintainability of Aspect-oriented Systems : a Practical Approach. International Journal of Computer Science and Security, 1-9.

Kumar, P. (2012). Aspect-oriented software quality model: The AOSQ Model. Advanced Computing: An International Journal (ACIJ), *3*, 105-118.

Laddad, R. (2003a). Aspect-oriented refactoring Series Retrieved 15 May, 2013, from http://www.theserverside.com/news/1366873/Part-1-Overview-and-Process

Laddad, R. (2003b). AspectJ in Action. Bruce Park Avenue Greenwich: CT: Manning Publications.

Laukkanen, J. (2008). Aspect-Oriented Programming. University of HELSINKI.

Marin, M., Deursen, A. V., and Moonen, L. (2007). Identifying Crosscutting Concerns Using Fan-in Analysis. ACM Transactions on Software Engineering and Methodology (TOSEM), 1-34.

McCall, J. A., Richards, P. K., and Walters, G. F. (1977). Factors In Software Quality - Concept and Definitions of Software Quality: Rome Air Development Center.

Mguni, K., and Ayalew, Y. (2013). An Assessment of Maintainability of an Aspect-Oriented System. ISRN Software Engineering, 1-11.

Mills, E. E. (1988). Software Metrics. Carnegie Mellon University.

Ordonez, M. J., and Haddad, H. M. (2008). The state of metrics in software industry. Paper presented at the the Fifth International Conference on Information Technology: New Generations.

Panovski, G. (2008). Product Software Quality. (Master's Thesis), Eindhoven University of Technology.

Pawlak, R., Seinturier, L., and Retaillé, J. P. (2005). Foundations of AOP for J2EE Development. Springer-Verlag.

Robillard, M. P., and Murphy, G. C. (2002). Concern graphs: Fnding and describing concerns using structural program dependencies. Paper presented at the the 24th International Conference on Software engineering.

Rosenberg, L. H., and Hyatt, L. E. (1996). Software Quality Metrics for Object-Oriented Environments. Crosstalk Jounal, 1-6.

Roy, C. K., and Cordy, J. R. (2007). A Survey on Software Clone Detection Research. School of Computing Queen's University.

Roy, C. K., Uddin, M. G., Roy, B., and Dean, T. R. (2007). Evaluating Aspect Mining Techniques: A Case Study. Paper presented at the 15th IEEE International Conference on Program Comprehension.

Sant'Anna, C., Garcia, A., Chavez, C., Lucena, C., and Staa, A. V. (2003). On the Reuse and Maintenance of Aspect-Oriented Software:An Assessment Framework. Paper presented at the XVII Brazilian Symposium on Software Engineering.

Shepherd, D., Gibson, E., and Pollock, L. (2004). Design and evaluation of an automated aspect mining tool. Paper presented at the International Conference on Software Engineering Research and Practice.

Shepherd, D., and Pollock, L. (2005). Interfaces, aspects and views. Paper presented at the Linking Aspect Technology and Evolution (LATE) Workshop.

Singh, N., and Gill, N. S. (2012). The Early Identification of Functional and Non-Functional Crosscutting Concerns. International Journal of Computer Applications, 25-32.

Sirbi, K., and Kulkarni, P. J. (2013). AOP and its impact on software quality. Elixir Computer Science and Engineering.

Srivisut, K., and Muenchaisri, P. (2006). Determining Threshold of Aspect-Oriented Software Metrics. Chulalongkorn University.

Stamelos, I. G., and Sfetsos, P. (2007). Agile Software Development Quality Assurance: IGI Global publishing.

Sutton, S. M., and Rouvellou, I. (2002). Modeling of Software Concerns in Cosmos. Paper presented at the the 1st International Conference on Aspect-Oriented Software Development.

Tonella, P., and Ceccato, M. (2004). Aspect mining through the formal concept analysis of execution traces. Paper presented at the 11th IEEE Working Conference on Reverse Engineering.

Tourwe, T., and Mens, K. (2004). Mining aspectual views using formal concept analysis. Paper presented at the Source Code Analysis and Manipulation Workshop (SCAM).

Vidal, S., Abait, E. S., Marcos, C., Casas, S., and Pace, J. A. P. (2009). Aspect Mining meets Rule-based Refactoring. Paper presented at the the 1st workshop on Linking aspect technology and evolution.

Yuen, I. (2009). Improve Software Modularity through Crosscutting Concern Extraction. (Master's Thesis), McGill University.

Zhao, J. (2002). Towards A Metrics Suite for Aspect-Oriented Software. Information Processing Society of Japan, 1-7.

กัลยา วานิชย์บัญชา. (2551a). การใช้ *SPSS for Windows* ในการวิเคราะห์ข้อมูล (11 ed.). กรุงเทพมหานคร: ธรรมสาร.

กัลยา วานิชย์บัญชา. (2551b). หลักสถิติ (10 ed.). กรุงเทพมหานคร: ธรรมสาร.

ประวัติผู้เขียนวิทยานิพนธ์

นางสาวธนาภรณ์ กังพานิชกุล เกิดวันที่ 7 มกราคม พ.ศ.2531 สำเร็จการศึกษาวิทยาศาสตร์บัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์ จากภาควิชาคณิตศาสตร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศรีนครินทรวิโรฒ ในปี พ.ศ.2553 จากนั้นได้เข้าศึกษาต่อในระดับปริญญาโทวิทยาศาสตรมหาบัณฑิต สาขาการพัฒนาซอฟต์แวร์ด้านธุรกิจ ภาควิชาสถิติ คณะพาณิชยศาสตร์และการบัญชี จุฬาลงกรณ์มหาวิทยาลัย