การศึกษาการตรวจทานโค้ดที่มีต่อความสามารถในการบำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์ส

The Study of the Peer Code Review on Software Maintainability in Open Source Projects

ธัญญรัตน์ กิจพาณิชย์

Thanyarat Kitpanich

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญา
วิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
มหาวิทยาลัยสงขลานครินทร์

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Information Technology
Prince of Songkla University

2561

การศึกษาการตรวจทานโค้ดที่มีต่อความสามารถในการบำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์ส

The Study of the Peer Code Review on Software Maintainability in Open Source Projects

ธัญญรัตน์ กิจพาณิชย์
Thanyarat Kitpanich

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญา
วิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
มหาวิทยาลัยสงขลานครินทร์
A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Information Technology
Prince of Songkla University
2561

ชื่อวิทยานิพนธ์ การศึกษาการตรวจทานโค้ดที่มีต่อความสามารถในการบำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์ส

ผู้เขียน นางสาวธัญญรัตน์ กิจพาณิชย์

สาขาวิชา เทคโนโลยีสารสนเทศ

อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

.....

(ผู้ช่วยศาสตราจารย์ ดร.อซีส นันทอมรพงศ์)

คณะกรรมการสอบ

.....ประธานกรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.รัตนา เวทย์ประสิทธิ์)

.....กรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.อซีส นันทอมรพงศ์)

.....กรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.ทรงศรี ตั้งศรีไพโรจน์)

บัณฑิตวิทยาลัย มหาวิทยาลัยสงขลานครินทร์ อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ

.....

(ศาสตราจารย์ ดร.ดำรงศักดิ์ ฟ้ารุ่งสาง)

คณบดีบัณฑิตวิทยาลัย

ขอรับรองว่า ผลงานวิจัยนี้มาจากการศึกษาวิจัยของนักศึกษาเอง และได้แสดงความขอบคุณบุคคลที่มีส่วนช่วยเหลือแล้ว

ลงชื่อ.....
(ผู้ช่วยศาสตราจารย์ ดร.อซีส นันทอมรพงศ์)
อาจารย์ที่ปรึกษาวิทยานิพนธ์

ลงชื่อ.....
(นางสาวธัญญรัตน์ กิจพาณิชย์)
นักศึกษา

ข้าพเจ้าขอรับรองว่า ผลงานวิจัยนี้ไม่เคยเป็นส่วนหนึ่งในการอนุมัติปริญญาในระดับใดมาก่อน และไม่ได้ถูกใช้ในการยื่นขออนุมัติปริญญาในขณะนี้

ลงชื่อ.....

(นางสาวธัญญรัตน์ กิจพาณิชย์)

นักศึกษา

ชื่อวิทยานิพนธ์ การศึกษาการตรวจทานโค้ดที่มีต่อความสามารถในการบำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์ส
ผู้เขียน นางสาวธัญญรัตน์ กิจพาณิชย์
สาขาวิชา เทคโนโลยีสารสนเทศ
ปีการศึกษา 2560

บทคัดย่อ

ในหลายปีที่ผ่านมาซอฟต์แวร์โอเพนซอร์สได้ถูกนำไปใช้งานกันอย่างกว้างขวาง เนื่องจากซอฟต์แวร์โอเพนซอร์สได้รับการพัฒนาโดยนักพัฒนาซอฟต์แวร์ที่มีความเชี่ยวชาญและมีประสบการณ์ในการพัฒนาระบบซอฟต์แวร์ ด้วยเหตุนี้จึงทำให้ซอฟต์แวร์โอเพนซอร์สค่อนข้างได้รับความนิยมและความน่าเชื่อถือในแง่ความถูกต้องของฟังก์ชันการทำงาน (Functionality) รวมถึงยังช่วยลดต้นทุนในการลงทุนทางด้านซอฟต์แวร์ อย่างไรก็ตามจากงานวิจัยก่อนหน้านี้พบว่า ปัญหาหลักของการพัฒนาโครงการโอเพนซอร์ส คือ ยังขาดขั้นตอนการดำเนินงานอย่างเป็นระบบและขาดเอกสารที่เกี่ยวข้องกับการพัฒนาซอฟต์แวร์อย่างเป็นทางการ ได้แก่ เอกสารความต้องการด้านซอฟต์แวร์ การออกแบบและการทดสอบ จากข้างต้นจึงทำให้เกิดปัญหาเกี่ยวกับคุณภาพของซอฟต์แวร์ เช่น ด้านความมั่นคง และด้านความสามารถในการบำรุงรักษา

งานวิจัยนี้ผู้วิจัยได้ทำการศึกษาคุณภาพของซอฟต์แวร์ทางด้านความสามารถในการบำรุงรักษาซอฟต์แวร์ เพราะคุณลักษณะนี้สามารถลดค่าใช้จ่ายของกระบวนการพัฒนาซอฟต์แวร์และยังช่วยเพิ่มผลผลิตของการพัฒนาซอฟต์แวร์ จากการทบทวนวรรณกรรมที่เกี่ยวข้องกับการตรวจทานโค้ดในโครงการโอเพนซอร์ส ผู้วิจัยพบว่ายังไม่มีการศึกษาว่านักพัฒนาซอฟต์แวร์โอเพนซอร์สให้ความสำคัญกับการบำรุงรักษาซอฟต์แวร์ เพื่อที่จะทำความเข้าใจวิธีการที่นักพัฒนาโอเพนซอร์สปรับปรุงซอร์สโค้ดด้านการบำรุงรักษาซอฟต์แวร์ งานวิจัยครั้งนี้จึงมีจุดมุ่งหมายที่จะตอบคำถามวิจัยหลัก คือ *“นักพัฒนาให้ความสนใจกับความสามารถในการบำรุงรักษาซอฟต์แวร์ภายใต้การตรวจทานโค้ดในโครงการซอฟต์แวร์โอเพนซอร์สหรือไม่?”* เพื่อตอบคำถามงานวิจัยที่ได้ระบุไว้ ผู้วิจัยได้ทำการตรวจสอบกระบวนการตรวจทานโค้ดที่นักพัฒนาในโครงการโอเพนซอร์สได้ทำการแก้ไขโค้ดเกี่ยวกับความสามารถในการบำรุงรักษาซอฟต์แวร์ตามที่ได้รับคำแนะนำและทำการรวบรวมคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์จากวรรณกรรมที่เกี่ยวข้อง โดยผู้วิจัยได้ทำการตรวจสอบข้อเสนอแนะจากโครงการโอเพนซอร์สจำนวน 2 โครงการ ได้แก่ โครงการ Eclipse และโครงการ Qt

ผลการวิจัยพบว่า จำนวนของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดทางด้านการบำรุงรักษามีจำนวนปานกลาง อย่างไรก็ตามนักพัฒนาในชุมชนโอเพนซอร์สก็มีแนวโน้มที่จะทำการปรับปรุงคุณภาพของซอร์สโค้ดมากขึ้น โดยแนวโน้มของข้อเสนอแนะที่ได้รับการแก้ไขปรับปรุงเกี่ยวกับการบำรุงรักษามีจำนวนที่เพิ่มสูงขึ้นเรื่อย ๆ ในแต่ละปี จึงทำให้สามารถวิเคราะห์ได้ว่า นักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์สให้ความสนใจกับความสามารถในการบำรุงรักษาซอฟต์แวร์ ผู้วิจัยคาดหวังว่า งานวิจัยนี้จะเป็นการเพิ่มหลักฐานเชิงประจักษ์เกี่ยวกับคุณภาพของโครงการซอฟต์แวร์โอเพนซอร์ส และเป็นการเสนอแนะแนวทางแก่นักพัฒนาซอฟต์แวร์และนักวิจัยทางด้านวิศวกรรมซอฟต์แวร์ได้ตระหนักถึงความสำคัญของการบำรุงรักษาซอฟต์แวร์ ก่อนที่จะทำการปรับปรุงซอร์สโค้ด

คำสำคัญ: วิศวกรรมซอฟต์แวร์ วิศวกรรมซอฟต์แวร์เชิงประจักษ์ ซอฟต์แวร์โอเพนซอร์ส ความสามารถในการบำรุงรักษาซอฟต์แวร์ การตรวจทานโค้ด

Thesis Title The Study of the Peer Code Review on Software Maintainability in Open Source Projects
Author Ms. Thanyarat Kitpanich
Major Program Information Technology
Academic Year 2017

ABSTRACT

Recently, open source software (OSS) applications have been widely adopting because OSS is developed by software developers who are the experts and have various experiences for software system development. Thus, OSS is considerably popular and reliable regarding functionality's corrections. The OSS also allows software developers to reduce technical debt in software development. However, previous studies show that the main problem of OSS development is the lacks of the systematic process and formal documents related to the system development such as requirements, designs, and testing. This characteristic of OSS development causes the problems in the software quality, such as security and maintainability.

In this research, the author focused on the software maintainability because this quality attribute can mostly reduce the cost of a software development process and also increase the productivity of software development. According to the existing literature related code reviews in OSS projects, the author found that it has currently no research studying whether the OSS developers pay attention to software maintainability. To better understand how the OSS developers improve the software maintainability; this research aims to answer the main question: "*Are the developers interested in the software maintainability under the modern code review of the open source software projects?*" To answer the research question, the author investigated the code review process in which the OSS developers changed the code based on the code review's comments due to the maintenance reason and collected the sub-characteristics associated with software maintainability from existing literature. The author examined the review comments from two OSS projects, including Eclipse and Qt.

The evidence from this study suggests that the number of code revisions based on the maintenance reasons was moderate. Nevertheless, the OSS developers tend to improve the quality of source code. This direction could be observed from the increasing number of modifications on given maintenance-based comments over the years. Therfore, an implication of this is the possibility that the developers in the OSS projects are interested in the software maintainability. Finally, the author expects that this research increases the empirical evidence about the quality of OSS projects and provides the guidelines for the software developers and software engineering researchers who realize in improving source code in the perspective of software maintainability.

Keyword: Software Engineering, Empirical Software Engineering, Open Source Software, Software Maintainability, Code Review

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้ด้วยความกรุณาอย่างยิ่งจากผู้มีพระคุณหลาย ๆ ฝ่าย โดยเฉพาะผู้ช่วยศาสตราจารย์ ดร.อชีส นันทอมรพงศ์ อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ถ่ายทอดประสบการณ์และความรู้ที่จำเป็นในการทำวิจัย พร้อมทั้งชี้แนะแนวทางที่สามารถแก้ไขปัญหาต่าง ๆ ที่เกิดขึ้น ตลอดระยะเวลาของการทำวิทยานิพนธ์ ผู้วิจัยรู้สึกซาบซึ้งในความกรุณาของอาจารย์เป็นอย่างยิ่ง จึงขอกราบขอบพระคุณอาจารย์เป็นอย่างสูงมา ณ ที่นี้

ขอกราบขอบพระคุณผู้ช่วยศาสตราจารย์ ดร.รัตนา เวทย์ประสิทธิ์ ประธานกรรมการสอบวิทยานิพนธ์ และผู้ช่วยศาสตราจารย์ ดร.ทรงศรี ตั้งศรีไพโรจน์ คณะกรรมการสอบวิทยานิพนธ์ ที่ให้ความรู้และคำแนะนำเกี่ยวกับการแก้ไขข้อผิดพลาด เพื่อเป็นประโยชน์ต่อการปรับปรุงวิทยานิพนธ์ฉบับนี้ให้ถูกต้อง สมบูรณ์ และทรงคุณค่ามากยิ่งขึ้น

ขอขอบคุณทุนสนับสนุนการวิจัยจากบัณฑิตวิทยาลัยและกองทุนวิจัยจากวิทยาลัยการคอมพิวเตอร์ มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตภูเก็ต ที่มอบโอกาสในการศึกษาและทุนสนับสนุนการวิจัยในครั้งนี้ รวมไปถึงคณาจารย์ทุกท่านในวิทยาลัยการคอมพิวเตอร์ที่ให้วิชาความรู้ ตลอดจนคำแนะนำที่เป็นประโยชน์ให้แก่ผู้วิจัยเสมอมา และขอขอบคุณเจ้าหน้าที่งานบัณฑิตศึกษา หลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต รวมถึงบุคลากรทุกท่านที่ให้ความช่วยเหลือและคำแนะนำในการติดต่อประสานงานต่าง ๆ จนทำให้วิทยานิพนธ์ฉบับนี้ สำเร็จลุล่วงด้วยดี

ขอกราบขอบพระคุณครอบครัวกิจพาณิชย์ ผู้เป็นกำลังใจยิ่งใหญ่ที่สุดของผู้วิจัย คอยอยู่เคียงข้างและให้การสนับสนุนทุกสิ่งทุกอย่างแก่ผู้วิจัย สุดท้ายนี้หากวิทยานิพนธ์ฉบับนี้มีสิ่งใดขาดตกบกพร่องประการใด ผู้วิจัยขออภัยเป็นอย่างสูงในข้อบกพร่องนั้น และหวังเป็นอย่างยิ่งว่าวิทยานิพนธ์ฉบับนี้จะเป็นประโยชน์แก่ผู้ที่สนใจจะนำไปทำการศึกษาต่อไม่มากก็น้อย

ธัญญรัตน์ กิจพาณิชย์

สารบัญ

	หน้า
บทคัดย่อ (ภาษาไทย)	(5)
บทคัดย่อ (ภาษาอังกฤษ)	(7)
กิตติกรรมประกาศ	(9)
สารบัญ	(10)
รายการตาราง	(13)
รายการรูปภาพ	(15)
บทที่ 1 บทนำ	1
1.1 ความเป็นมาและความสำคัญของปัญหา	1
1.2 วัตถุประสงค์	6
1.3 ขอบเขตของการวิจัย	6
1.4 ประโยชน์ของการวิจัย	8
บทที่ 2 วรรณกรรมและเทคโนโลยีที่เกี่ยวข้อง	9
2.1 หลักการ ความรู้ และเทคโนโลยีที่เกี่ยวข้อง	9
2.1.1 ซอฟต์แวร์โอเพนซอร์ส	9
2.1.2 ปัจจัยที่มีผลต่อคุณภาพของซอฟต์แวร์โอเพนซอร์ส	12
2.1.3 การตรวจทานโค้ด	14
2.1.4 คุณภาพของซอฟต์แวร์และความสามารถในการบำรุงรักษาซอฟต์แวร์	23
2.1.5 การทำเหมืองข้อความ	27
2.1.6 อัลกอริทึม Latent Dirichlet Allocation	29
2.2 ซอฟต์แวร์ที่เกี่ยวข้อง	31
2.2.1 เกอริต (Gerrit) และกิต (Git)	31
2.2.2 โปรแกรมอาร์ (R)	36
2.2.3 โปรแกรมเวิร์ดเน็ต (WordNet)	37
2.3 วรรณกรรมที่เกี่ยวข้อง	38
2.3.1 คุณภาพของซอฟต์แวร์โอเพนซอร์ส	38
2.3.2 กระบวนการตรวจทานโค้ดและเครื่องมือที่ใช้ในการตรวจทานโค้ด	40
2.3.3 ความสามารถในการบำรุงรักษา	43

สารบัญ (ต่อ)

	หน้า
บทที่ 3 วิธีดำเนินการวิจัย	46
3.1 การศึกษาวิจัยเชิงพรรณนา	47
3.2 การเก็บรวบรวมข้อมูลของโครงการโอเพนซอร์ส	47
3.2.1 การกำหนดโครงการโอเพนซอร์สที่ใช้ในการศึกษา	47
3.2.2 การพัฒนาซอฟต์แวร์ที่ช่วยในการสืบค้นและจัดเก็บข้อมูลจากระบบเกอริต	48
3.2.3 การตรวจสอบข้อมูลที่ได้ทำการสืบค้นจากระบบเกอริต	49
3.3 การค้นหาคุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์	51
3.4 การวิเคราะห์ข้อมูล	59
3.5 จัดทำรายงานผลการวิจัย	61
บทที่ 4 ผลการวิจัย	63
4.1 คุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์สแบ่งออกเป็นกี่คุณลักษณะ	64
4.1.1 การกำหนดคำหลัก	64
4.1.2 ผลของการสืบค้นและรวบรวมคำที่มีความหมายสื่อถึงคำหลัก	67
4.1.3 ผลที่ได้จากการค้นหาคุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์เพิ่มเติมจากที่มีอยู่	72
4.2 นักพัฒนาซอฟต์แวร์ให้ความสนใจกับความสามารถในการบำรุงรักษาซอฟต์แวร์ภายใต้การตรวจทานโค้ดในโครงการโอเพนซอร์สหรือไม่	76
4.2.1 ผลของการค้นหาข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษา	79
4.2.2 ผลของจำนวนการแก้ไขโค้ดที่ได้รับคำแนะนำทางด้านการบำรุงรักษา	82
4.2.3 การสรุปผลลัพธ์ของการตรวจสอบความสนใจของชุมชนนักพัฒนาในโครงการโอเพนซอร์สที่มีต่อการบำรุงรักษาซอฟต์แวร์	86
4.2.4 ความสัมพันธ์ระหว่างจำนวนข้อเสนอแนะที่ได้จากการค้นหาและจำนวนการแก้ไขโค้ดตามคำแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ของทั้ง 2 โครงการ	89
4.2.5 แนวโน้มของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษา และแนวโน้มของการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ของทั้ง 2 โครงการ	91

สารบัญ (ต่อ)

	หน้า
4.3 คุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ประเภทใดที่นักพัฒนาให้ความสนใจเป็นพิเศษ	106
4.4 ขนาดกลุ่มของผู้ตรวจทานโค้ดส่งผลต่อจำนวนข้อเสนอแนะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์หรือไม่	110
4.4.1 บทบาทของนักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์ส	110
4.4.2 ขนาดกลุ่มของผู้ตรวจทานโค้ดที่ให้คำแนะนำเกี่ยวกับการบำรุงรักษา	112
4.5 ผู้ตรวจทานโค้ดทำงานเกี่ยวกับการตรวจทานโค้ดในโครงการโอเพนซอร์สมากน้อยเพียงใดต่อสัปดาห์	119
บทที่ 5 สรุปผลการวิจัย	126
5.1 การอภิปรายและสรุปผลการวิจัย	126
5.2 ข้อเสนอแนะและข้อจำกัดของงานวิจัย	133
5.2.1 ข้อเสนอแนะเกี่ยวกับข้อมูลที่นำมาใช้ในงานวิจัย	133
5.2.2 ข้อเสนอแนะทางด้านเครื่องมือ	133
5.3 ภัยคุกคามต่อความถูกต้องที่เกิดขึ้นในการทำงานวิจัย	134
5.3.1 ความถูกต้องเชิงโครงสร้าง (Construct Validity)	134
5.3.2 ความถูกต้องภายใน (Internal Validity)	135
5.3.3 ความถูกต้องภายนอก (External Validity)	136
5.4 งานวิจัยในอนาคต	137
เอกสารอ้างอิง	138
ภาพผนวก	148
ประวัติผู้เขียน	155

รายการตาราง

ตารางที่		หน้า
1.1	รายละเอียดลักษณะของโครงการโอเพนซอร์สทั้ง 2 โครงการ	7
1.2	รายละเอียดของข้อมูลโครงการโอเพนซอร์ส	7
2.1	การจำแนกกลุ่มตามจำนวนของนักพัฒนาในโครงการโอเพนซอร์ส	11
2.2	รายละเอียดของเทคนิคการตรวจทานโค้ด	15
2.3	การเปรียบเทียบเทคนิคในการตรวจทานโค้ด	18
2.4	รายละเอียดของเครื่องมือที่ใช้ในการตรวจทานโค้ด	19
3.1	ตัวอย่างตารางการเก็บข้อมูลที่ได้จากการค้นหาข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาของโครงการ Eclipse ในช่วงระยะเวลา 5 ปี	57
4.1	คำหลักที่ใช้ในการค้นหาข้อเสนอแนะของผู้ตรวจทานโค้ด	66
4.2	ตัวอย่างผลลัพธ์ที่ได้จากการค้นหาคำที่มีความหมายสื่อถึงคำหลัก “Modular”	67
4.3	ผลลัพธ์ที่ได้จากการค้นหาคำที่มีความหมายสื่อถึงคำหลักทั้ง 33 คำ	69
4.4	ผลลัพธ์ที่ได้จากการใช้อัลกอริทึม LDA ในโปรแกรม R	73
4.5	คุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาที่ค้นพบจากการใช้อัลกอริทึม LDA	74
4.6	ผลลัพธ์ของความสัมพันธ์ระหว่างคำที่มักจะเกิดขึ้นร่วมกับคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ประเภทใหม่	75
4.7	ผลลัพธ์ของการค้นหาคำที่สื่อถึงคำหลักของคุณลักษณะทางด้านการบำรุงรักษาประเภทใหม่ที่ค้นพบ	76
4.8	รายละเอียดข้อมูลของโครงการโอเพนซอร์สทั้ง 2 โครงการที่มีการแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ในช่วงปี ค.ศ. 2012 – 2016	77
4.9	จำนวนความถี่ของคำหลักที่ปรากฏทั้งหมดในข้อเสนอแนะของโครงการ Eclipse	79
4.10	จำนวนความถี่ของคำหลักที่ปรากฏทั้งหมดในข้อเสนอแนะของโครงการ Qt	81
4.11	จำนวนการแก้ไขโค้ดทางด้านการบำรุงรักษาของโครงการ Eclipse	83
4.12	จำนวนการแก้ไขโค้ดทางด้านการบำรุงรักษาของโครงการ Qt	84
4.13	รายละเอียดผลลัพธ์ที่ได้จากการวิเคราะห์ข้อเสนอแนะเกี่ยวกับการบำรุงรักษาที่ได้รับการแก้ไขโค้ดในช่วงปี ค.ศ. 2012-2016	86
4.14	ผลลัพธ์จากการวิเคราะห์ที่เกิดขึ้นในโครงการ Eclipse ในช่วงระยะเวลา 5 ปี	89
4.15	ผลลัพธ์จากการวิเคราะห์ที่เกิดขึ้นในโครงการ Qt ในช่วงระยะเวลา 5 ปี	89
4.16	การเก็บรวบรวมข้อมูลจากข้อเสนอแนะที่เกิดขึ้นในโครงการ Eclipse ระยะเวลา 5 ปี	93

รายการตาราง (ต่อ)

ตารางที่	หน้า
4.17 การเก็บรวบรวมข้อมูลจากข้อเสนอแนะที่เกิดขึ้นในโครงการ Qt ระยะเวลา 5 ปี	93
4.18 การเก็บรวบรวมข้อเสนอแนะที่ได้รับการแก้ไขโค้ดในโครงการ Eclipse ระยะเวลา 5 ปี	99
4.19 การเก็บรวบรวมข้อเสนอแนะที่ได้รับการแก้ไขโค้ดในโครงการ Qt ระยะเวลา 5 ปี	99
4.20 จำนวนผู้ตรวจทานโค้ดที่ให้คำแนะนำในการปรับปรุงโค้ดและจำนวนข้อเสนอแนะเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ในแต่ละเดือนของโครงการ Eclipse	112
4.21 จำนวนผู้ตรวจทานโค้ดที่ให้คำแนะนำในการปรับปรุงโค้ดและจำนวนข้อเสนอแนะเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ในแต่ละเดือนของโครงการ Qt	114
4.22 จำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาของโครงการ Eclipse ในแต่ละวันของช่วงสัปดาห์	120
4.23 จำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาของโครงการ Qt ในแต่ละวันของช่วงสัปดาห์	120

รายการรูปภาพ

รูปที่		หน้า
2.1	กระบวนการตรวจทานโค้ดในโครงการโอเพนซอร์ส	22
2.2	รูปแบบคุณภาพของผลิตภัณฑ์ซอฟต์แวร์ที่กำหนดไว้ใน ISO/IEC 25010	23
2.3	รูปแบบการทำงานของการทำเหมืองข้อความ	28
2.4	การสร้างแบบจำลองหัวข้อด้วย Latent Dirichlet Allocation	30
2.5	ภาพรวมการทำงานร่วมกันระหว่างระบบเกอริตและระบบจัดการกิต	32
2.6	ตัวอย่างข้อมูลของโครงการ Eclipse ที่จัดเก็บอยู่ในระบบเกอริต	35
3.1	ขั้นตอนในการดำเนินงานวิจัย	46
3.2	ภาพรวมการทำงานของซอฟต์แวร์ที่ใช้ในการสืบค้นข้อมูล	48
3.3	ขั้นตอนในการค้นหาคุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์	51
3.4	ชุดคำสั่ง R ที่ได้ทำการพัฒนาสำหรับใช้ในการทำความสะอาดข้อมูล	53
3.5	ชุดคำสั่ง R ที่ได้ทำการพัฒนาสำหรับใช้ในการค้นหาคำหลักที่ปรากฏในข้อเสนอแนะ	55
3.6	ตัวอย่างข้อเสนอแนะที่ได้รับการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาซอฟต์แวร์	58
4.1	คำสั่งการใช้อัลกอริทึม LDA ในโปรแกรม R	72
4.2	อัตราส่วนของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์	78
4.3	อัตราส่วนของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษา	87
4.4	อัตราส่วนของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดทางด้านการบำรุงรักษาในช่วง 5 ปี	88
4.5	กราฟแสดงความสัมพันธ์ระหว่างข้อเสนอแนะเกี่ยวกับการบำรุงรักษาและการแก้ไขโค้ดที่ได้รับคำแนะนำทางด้านการบำรุงรักษาโครงการ Eclipse และโครงการ Qt	91
4.6	แนวโน้มของข้อเสนอแนะเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ทั้ง 35 คุณลักษณะ	92
4.7	แผนภาพกระจายแสดงความสัมพันธ์ระหว่างจำนวนของข้อเสนอแนะเกี่ยวกับการบำรุงรักษากับปีที่ให้คำแนะนำทางด้านการบำรุงรักษาของผู้ตรวจทานโค้ด	94
4.8	เส้นกราฟการถดถอยที่แสดงแนวโน้มการให้ข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษา	95
4.9	แนวโน้มของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาทั้ง 35 คุณลักษณะ	98
4.10	แผนภาพกระจายแสดงความสัมพันธ์ระหว่างจำนวนการแก้ไขโค้ดตามข้อเสนอแนะเกี่ยวกับการบำรุงรักษากับปีที่มีการแนะนำให้นักพัฒนาทำการปรับปรุงแก้ไขโค้ด	100
4.11	กราฟการถดถอยที่แสดงแนวโน้มของจำนวนการแก้ไขโค้ดทางด้านการบำรุงรักษาซอฟต์แวร์ในอนาคต	101

รายการรูปภาพ (ต่อ)

รูปที่	หน้า
4.12 อัตราส่วนของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดทางด้านการบำรุงรักษาทั้ง 35 ประเภท ในโครงการ Eclipse	103
4.13 อัตราส่วนของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดทางด้านการบำรุงรักษาทั้ง 35 ประเภท ในโครงการ Qt	104
4.14 กราฟแสดงแนวโน้มของข้อเสนอแนะเกี่ยวกับการบำรุงรักษาและแนวโน้มของการแก้ไขโค้ดที่ได้รับคำแนะนำทางด้านการบำรุงรักษาโครงการ Eclipse และโครงการ Qt	105
4.15 จำนวนของข้อเสนอแนะทางด้านการบำรุงรักษาที่ได้รับคำแนะนำจากผู้ตรวจทานโค้ดของโครงการ Eclipse	106
4.16 จำนวนของข้อเสนอแนะทางด้านการบำรุงรักษาที่ได้รับคำแนะนำจากผู้ตรวจทานโค้ดของโครงการ Qt	107
4.17 จำนวนการแก้ไขปรับปรุงซอร์สโค้ดตามข้อเสนอแนะเกี่ยวกับการบำรุงรักษาทั้ง 5 ประเภทของโครงการ Eclipse และโครงการ Qt ที่ได้รับความนิยม	108
4.18 อัตราส่วนของการแก้ไขโค้ดทางด้านการบำรุงรักษาที่นักพัฒนาทำการแก้ไข	109
4.19 อัตราส่วนของบทบาทการเป็นผู้ตรวจทานโค้ดและนักพัฒนาที่ทำการแก้ไขโค้ด	110
4.20 ความสัมพันธ์ระหว่างจำนวนของผู้ตรวจทานโค้ดกับจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ทั้ง 12 เดือนของโครงการ Eclipse ในช่วงปี ค.ศ. 2012 – 2016	116
4.21 ความสัมพันธ์ระหว่างจำนวนของผู้ตรวจทานโค้ดกับจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ทั้ง 12 เดือนของโครงการ Qt ในช่วงปี ค.ศ. 2012 – 2016	117
4.22 แนวโน้มของจำนวนผู้ตรวจทานโค้ดในโครงการ Eclipse และโครงการ Qt ที่ให้คำแนะนำเกี่ยวกับการบำรุงรักษาในช่วงระยะเวลา 5 ปี	118
4.23 ผลลัพธ์จากการวิเคราะห์ค่าความแปรปรวนแบบ Tukey’s HSD ของโครงการ Eclipse	122
4.24 ผลลัพธ์จากการวิเคราะห์ค่าความแปรปรวนแบบ Tukey’s HSD ของโครงการ Qt	123
4.25 กราฟแสดงจำนวนข้อเสนอแนะเกี่ยวกับการบำรุงรักษาแต่ละวันต่อสัปดาห์ในช่วงระยะเวลา 5 ปี (ค.ศ. 2012-2016) ของโครงการ Eclipse และโครงการ Qt	124

บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันซอฟต์แวร์โอเพนซอร์ส (Open Source Software) เป็นซอฟต์แวร์ที่ได้รับความสนใจจากผู้ใช้งานทั่วโลก โดยเฉพาะองค์กรและอุตสาหกรรมเชิงพาณิชย์ได้มีการนำซอฟต์แวร์โอเพนซอร์สมาประยุกต์ใช้งานกันอย่างกว้างขวาง เพราะซอฟต์แวร์โอเพนซอร์สเป็นส่วนหนึ่งที่จะนำไปสู่ความสำเร็จของธุรกิจสำหรับผู้ประกอบการ เนื่องจากซอฟต์แวร์โอเพนซอร์สสามารถช่วยสร้างโอกาสในการเติบโตขององค์กร ช่วยเพิ่มประสิทธิภาพการทำงาน และเพิ่มทางเลือกในการใช้ซอฟต์แวร์ รวมถึงยังช่วยลดต้นทุนในการดำเนินงาน สิ่งที่ได้กล่าวมานั้นเกิดขึ้นได้เพราะซอฟต์แวร์โอเพนซอร์สมีคุณลักษณะที่สำคัญ คือ ผู้ใช้งานทั่วโลกสามารถนำซอฟต์แวร์ไปใช้งานได้อย่างอิสระโดยไม่จำเป็นต้องเสียค่าใช้จ่ายและยังเปิดโอกาสให้นักพัฒนาซอฟต์แวร์จากหลากหลายประเทศทั่วโลกที่สนใจในโครงการซอฟต์แวร์โอเพนซอร์สสามารถเข้าถึงแหล่งเก็บข้อมูลซอร์สโค้ด (Code Repository) (Rigby, et al., 2008) เพื่อร่วมกันพัฒนาซอฟต์แวร์ตามวัตถุประสงค์ที่ได้กำหนดไว้ โดยที่นักพัฒนาซอฟต์แวร์ไม่จำเป็นต้องอยู่ในสถานที่เดียวกัน (Aberdour, 2007) ด้วยเหตุนี้จึงทำให้โครงการซอฟต์แวร์โอเพนซอร์สได้รับการพัฒนาและแก้ไขปรับปรุง รวมถึงมีการวิวัฒนาการซอฟต์แวร์อย่างต่อเนื่อง โดยกลุ่มนักพัฒนาซอฟต์แวร์ที่มีความเชี่ยวชาญและประสบการณ์ทางด้านการพัฒนาระบบ ซึ่งได้มีการนำความรู้เกี่ยวกับเทคนิค กระบวนการและเครื่องมือต่าง ๆ มาประยุกต์ใช้เพื่อสนับสนุนการผลิตซอฟต์แวร์ได้อย่างมีประสิทธิภาพ (Hammond, et al., 2016) ในการแลกเปลี่ยนความรู้เกี่ยวกับการพัฒนาซอฟต์แวร์ของชุมชนนักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์สจะอาศัยเทคโนโลยีเป็นสื่อกลางในการติดต่อสื่อสารระหว่างกัน ซึ่งรูปแบบในการติดต่อสื่อสารทางอิเล็กทรอนิกส์ที่ได้รับความนิยม ได้แก่ อีเมล์ เว็บบอร์ด โปรแกรมการสนทนา เป็นต้น

จากคุณลักษณะข้างต้นทำให้ซอฟต์แวร์โอเพนซอร์สค่อนข้างได้รับความนิยมและความน่าเชื่อถือในแง่ความถูกต้องของฟังก์ชันการทำงาน (Functionality) แต่จากการศึกษางานวิจัยก่อนหน้านี้ (Tiwari and Pandey, 2012) แสดงให้เห็นว่าการพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์สยังไม่มีกระบวนการพัฒนาที่ใช้ปฏิบัติกันอย่างเป็นระบบและขาดเอกสารที่ช่วยสนับสนุนการพัฒนาซอฟต์แวร์

อย่างเป็นทางการ โดยเฉพาะเอกสารที่เกี่ยวข้องกับการกำหนดความต้องการเชิงซอฟต์แวร์ การออกแบบและการทดสอบระบบ จากปัญหาดังกล่าวจึงเป็นสาเหตุที่ทำให้ซอร์สโค้ดของโครงการซอฟต์แวร์โอเพนซอร์สบางโครงการมีคุณภาพไม่ดีเท่าที่ควร (Norick, et al., 2010) เนื่องจากนักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์สส่วนใหญ่จะทำการแก้ไขปัญหาก็ต่อเมื่อได้รับรายงานจากผู้ใช้งานระบบหรือนักพัฒนาซอฟต์แวร์ภายในทีมเดียวกัน ซึ่งปัญหาที่มักจะพบ คือ การเกิดข้อบกพร่อง (Defect) หรือข้อผิดพลาด (Bug) ภายในระบบ และเมื่อทำการแก้ไขปัญหาเหล่านี้หรือปรับปรุงซอร์สโค้ดบ่อยครั้งก็จะส่งผลให้ซอฟต์แวร์มีจำนวนบรรทัดของซอร์สโค้ดมากขึ้น รวมถึงมีแนวโน้มที่จะทำให้ระบบเกิดความซับซ้อน (Complexity) มากยิ่งขึ้นอีกด้วย อีกทั้งซอร์สโค้ดที่ไม่ได้รับการเอาใจใส่เท่าที่ควรจะทำให้ซอร์สโค้ดนั้นยากต่อการทำความเข้าใจ จากลักษณะเช่นนี้อาจส่งผลให้เกิดปัญหาที่ร้ายแรงต่อระบบในอนาคต เช่น เรื่องของเวลาหรือค่าใช้จ่ายในการพัฒนาและปรับปรุงซอฟต์แวร์ที่เพิ่มมากขึ้น นักพัฒนาซอฟต์แวร์หรือผู้ประกอบการเชิงพาณิชย์ที่ต้องการจะนำซอฟต์แวร์ไปพัฒนาต่อยอด ไม่สามารถวิเคราะห์ระบบเดิมที่มีความซับซ้อนได้ การเข้าถึงหรือเรียกใช้งานระบบถูกขัดขวางหรือเกิดความล้มเหลว ข้อมูลสารสนเทศเกิดความผิดพลาดหรือมีค่าผิดเพี้ยนไปจากความคาดหวังของผู้ใช้งานระบบ รวมถึงมีความเสี่ยงที่จะเกิดช่องโหว่ (Vulnerability) ภายในระบบที่อาจก่อให้เกิดภัยคุกคามจากผู้ไม่ประสงค์ดีที่สามารถโจรกรรมข้อมูลหรือทำลายระบบซอฟต์แวร์ขององค์กรและธุรกิจเชิงพาณิชย์ที่นำซอฟต์แวร์โอเพนซอร์สไปใช้งาน ตลอดจนอาจเป็นผลที่ทำให้โครงการซอฟต์แวร์โอเพนซอร์สหยุดการพัฒนาและประสบความล้มเหลวในที่สุด จากสาเหตุที่ได้กล่าวไว้ข้างต้นแสดงให้เห็นถึงข้อจำกัดทางด้านคุณภาพของซอฟต์แวร์โอเพนซอร์ส เช่น ความมั่นคง (Security) และความสามารถในการบำรุงรักษา (Maintainability) (Zhou and Davis, 2005)

เพื่อที่จะแก้ไขปัญหาเกี่ยวกับคุณภาพของซอฟต์แวร์โอเพนซอร์สและป้องกันไม่ให้เกิดปัญหาต่าง ๆ ที่จะส่งผลกระทบต่อระบบในอนาคต ชุมชนนักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์สจึงควรให้ความสำคัญกับกระบวนการพัฒนาซอฟต์แวร์ โดยเฉพาะขั้นตอนของการบำรุงรักษาซอฟต์แวร์ (Software Maintenance) ซึ่งเป็นขั้นตอนที่มีความสำคัญที่สุดของวงจรชีวิตของการพัฒนาซอฟต์แวร์ (Glass, 2002; Nasir and Abbasi, 2010) เพราะหน้าที่หลักของการบำรุงรักษาซอฟต์แวร์ คือ การควบคุมให้ผลิตภัณฑ์หรือซอฟต์แวร์มีคุณภาพสูงด้วยต้นทุนที่ต่ำที่สุด และผลิตซอฟต์แวร์ให้สอดคล้องกับวัตถุประสงค์ที่ได้กำหนดไว้ แต่จากการศึกษาในการพัฒนาซอฟต์แวร์พบว่า นักพัฒนาซอฟต์แวร์ส่วนใหญ่ต้องใช้เวลาประมาณ 40-50% เพื่อค้นหาข้อบกพร่องและข้อผิดพลาดที่เกิดขึ้นระหว่างการพัฒนาซอฟต์แวร์ (Walia and Carver, 2013) หรือหลังจากส่งมอบ (Ghosh, et al., 2011) อีกทั้งต้องเสียค่าใช้จ่ายในกิจกรรมการบำรุงรักษาซอฟต์แวร์สูงถึง 40-80% (Glass, 2001; Wagey, et al., 2015) ด้วยเหตุนี้โครงการซอฟต์แวร์โอเพนซอร์สหลายโครงการ (Mockus, et al., 2002; Mcintosh, et al., 2014; Kononenko, et al., 2015) จึงได้นำการตรวจทานโค้ด (Peer Code Review) หรือที่เรียกกันว่า

การตรวจทานโค้ดสมัยใหม่ (Modern Code Review) มาเป็นแนวทางในการพัฒนาและปรับปรุงโค้ด ซึ่งกระบวนการตรวจทานโค้ดเป็นวิธีการหนึ่งที่จะช่วยลดความพยายามในการทำงานของนักพัฒนาซอฟต์แวร์ โดยเฉพาะการแก้ไขปัญหาเกี่ยวกับข้อบกพร่องหรือข้อผิดพลาดที่เกิดขึ้นในกระบวนการพัฒนาซอฟต์แวร์

การตรวจทานโค้ดเป็นวิธีการหนึ่งที่ได้รับการยอมรับทางด้านวิศวกรรมซอฟต์แวร์ในด้านการควบคุมคุณภาพของซอฟต์แวร์ (Baker, 1997) โดยซอร์สโค้ดที่มีการแก้ไขจากนักพัฒนาจะต้องได้รับการตรวจทานจากผู้ตรวจทาน (Reviewer) ซึ่งเป็นนักพัฒนาที่อยู่ภายในโครงการเดียวกันแต่ไม่ใช่ผู้แก้ไขโค้ดนั้นเอง โดยทั่วไปกระบวนการตรวจทานโค้ดนี้จะไม่มีรูปแบบที่ตายตัว แต่มีวัตถุประสงค์ที่สำคัญ คือ การป้องกันปัญหาที่อาจจะเกิดขึ้นจากโค้ดโปรแกรมในส่วนที่กำลังดำเนินการแก้ไข โดยตรวจสอบเพื่อให้แน่ใจว่าการเปลี่ยนแปลงโค้ดโปรแกรมจะทำให้เกิดข้อผิดพลาดลดน้อยลงและการเปลี่ยนแปลงนั้นจะไม่ส่งผลต่อการบำรุงรักษาซอฟต์แวร์ในระยะยาว สำหรับขั้นตอนการตรวจทานนี้ผู้ตรวจทานโค้ดจะมีการระบุข้อเสนอแนะ (Comment) และพิจารณาว่าโค้ดโปรแกรมที่ได้ทำการแก้ไขนั้นพร้อมที่จะนำเข้าไปเก็บในคลังเก็บข้อมูลซอร์สโค้ดหรือไม่ โดยข้อเสนอแนะนี้จะถูกส่งกลับไปยังนักพัฒนา เพื่อให้โค้ดได้รับการแก้ไขใหม่และส่งให้ผู้ตรวจทานทำการตรวจอีกครั้งก่อนที่จะนำเข้าไปเก็บในคลังเก็บข้อมูลซอร์สโค้ดเพื่อให้นักพัฒนาคนอื่น ๆ สามารถพัฒนาต่อยอดหรือมีการนำซอฟต์แวร์ไปใช้งานจริง (Paulson, et al., 2004) สำหรับประโยชน์ของการตรวจทานโค้ดจะช่วยนักพัฒนาสามารถสรุปได้ดังนี้

1) เพื่อค้นหาและกำจัดข้อบกพร่องหรือข้อผิดพลาดภายในระบบ
2) การเปลี่ยนแปลงโค้ดให้เป็นมาตรฐาน (Coding Standards) ที่นักพัฒนาซอฟต์แวร์สามารถอ่านและวิเคราะห์โค้ด เพื่อทำความเข้าใจและสามารถพัฒนาต่อยอดหรือปรับปรุงคุณภาพของซอฟต์แวร์ให้ดียิ่งขึ้น
3) เป็นเครื่องมือที่ช่วยให้ทีมนักพัฒนาสามารถแบ่งปันความรู้เกี่ยวกับการพัฒนาและการปรับปรุงคุณภาพของซอฟต์แวร์ (Bacchelli and Bird, 2013)

เนื่องจากประโยชน์ที่นักพัฒนาได้รับจากการตรวจทานโค้ดข้างต้นแสดงให้เห็นว่า สิ่งที่สำคัญของการตรวจทานโค้ดนี้ คือ ข้อเสนอแนะที่ได้รับจากผู้ตรวจทานโค้ด โดยข้อเสนอแนะเหล่านั้นจะชี้ให้เห็นถึงข้อบกพร่อง คำแนะนำในการปรับปรุงซอร์สโค้ดและวิธีการแก้ไขปัญหา หรือชี้ให้เห็นถึงการปฏิบัติของทีมนักพัฒนาที่สามารถช่วยให้นักพัฒนาแก้ไขโค้ดได้มีคุณภาพที่สูงขึ้นก่อนที่จะถูกนำไปใช้งาน รวมถึงสามารถพัฒนาทักษะด้านการเขียนโค้ดของนักพัฒนา (Bosu, et al., 2015) ดังนั้นการตรวจทานโค้ดจึงเป็นส่วนที่สำคัญของกระบวนการพัฒนาซอฟต์แวร์ (Zanjani, et al., 2016) เพราะเป็นกระบวนการที่ช่วยในการประกันคุณภาพของซอฟต์แวร์และการประสบความสำเร็จของโครงการซอฟต์แวร์โอเพนซอร์ส (Bosu and Carver, 2012) อย่างไรก็ตามข้อเสนอแนะในกระบวนการตรวจทานโค้ดเหล่านี้ยังไม่ได้รับการศึกษามากนักในปัจจุบัน โดยเฉพาะข้อเสนอแนะที่เกี่ยวข้องกับ

ความสามารถในการบำรุงรักษาซอฟต์แวร์ (Software Maintainability) ซึ่งเป็นคุณลักษณะที่จะช่วยลดค่าใช้จ่ายของขั้นตอนการบำรุงรักษาในกระบวนการพัฒนาซอฟต์แวร์และยังช่วยเพิ่มผลผลิตของการพัฒนาซอฟต์แวร์ อีกทั้งยังเป็นคุณลักษณะที่สำคัญต่อการประสบความสำเร็จทางด้านคุณภาพของซอฟต์แวร์ (Deissenboeck, et al., 2007) โดยมาตรฐานไอเอสโอ (ISO) 25010 ด้านคุณภาพผลิตภัณฑ์ซอฟต์แวร์ (Iso Iec, 2011) ได้นิยามความสามารถในการบำรุงรักษาซอฟต์แวร์ไว้ดังนี้ “คุณลักษณะที่มีความสามารถในการแก้ไขเพื่อปรับปรุงโค้ดให้ดีขึ้น การปรับเปลี่ยนหรือเปลี่ยนแปลงในสภาพแวดล้อมและตามความต้องการของซอฟต์แวร์ที่ได้กำหนดไว้”

ความสามารถในการบำรุงรักษาซอฟต์แวร์ถือว่าเป็นคุณลักษณะหนึ่งที่ส่งผลต่อคุณภาพของซอฟต์แวร์เป็นอย่างยิ่ง ดังนั้นผู้วิจัยจึงสนใจที่จะทำการศึกษาข้อเสนอแนะเกี่ยวกับความสามารถในการบำรุงรักษาซอฟต์แวร์ภายใต้กระบวนการตรวจทานโค้ดในโครงการโอเพนซอร์สจำนวน 2 โครงการ นั่นคือ โครงการ Eclipse[1] และโครงการ Qt[2] โดยข้อมูลเกี่ยวกับข้อเสนอแนะของทั้ง 2 โครงการนั้น ผู้วิจัยจะนำมาจากระบบที่ให้บริการเกี่ยวกับกระบวนการตรวจทานโค้ดที่มีชื่อว่า เกอริต (Gerrit)[3] ซึ่งเป็นระบบที่ช่วยอำนวยความสะดวกในการติดต่อสื่อสารเพื่อแลกเปลี่ยนความคิดเห็นเกี่ยวกับการพัฒนาซอฟต์แวร์ระหว่างนักพัฒนาด้วยกัน โดยไม่จำเป็นต้องอยู่ในสถานที่เดียวกันและไม่ต้องมีการกำหนดเวลาไว้ก่อนล่วงหน้า อีกทั้งเกอริตยังเป็นระบบที่มีการทำงานร่วมกับแหล่งเก็บข้อมูลซอร์สโค้ดสำหรับการควบคุมเวอร์ชันของระบบที่เรียกกันว่ากิต (Git)[4] ที่มีโครงการโอเพนซอร์สจำนวนมากใช้บริการอยู่ในปัจจุบัน สำหรับคำถามวิจัยหลักที่ผู้วิจัยได้กำหนดไว้ในงานวิจัยนี้ คือ

1) คุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์สแบ่งออกเป็นกี่คุณลักษณะ
2) นักพัฒนาซอฟต์แวร์ให้ความสนใจกับความสามารถในการบำรุงรักษาซอฟต์แวร์ภายใต้การตรวจทานโค้ดในโครงการโอเพนซอร์สหรือไม่
3) คุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ประเภทใดที่นักพัฒนาให้ความสนใจเป็นพิเศษ

เพื่อตอบคำถามงานวิจัยที่ได้ระบุไว้ข้างต้น ผู้วิจัยได้ใช้วิธีการทางด้านเหมืองข้อความ (Text Mining) ในการค้นหาคุณลักษณะของความสามารถในการบำรุงรักษาซอฟต์แวร์ที่ปรากฏอยู่ในข้อเสนอแนะภายใต้กระบวนการตรวจทานโค้ดและนำอัลกอริทึม Latent Dirichlet Allocation (LDA)

[1] https://eclipse.org/
[2] https://www.qt.io/
[3] https://www.gerritcodereview.com/
[4] https://git-scm.com/

มาประยุกต์ใช้ในการจัดกลุ่มคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์เพิ่มเติมจากที่มีอยู่ พร้อมทั้งทำการตรวจสอบว่า นักพัฒนาซอฟต์แวร์ได้ทำการแก้ไขโค้ดตามข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์หรือไม่ ซึ่งผลที่ได้จากข้อมูลเชิงปริมาณ (Quantitative) จะถูกนำไปวิเคราะห์เพื่อแสดงให้เห็นถึงแนวโน้มความสนใจของนักพัฒนาที่มีต่อการบำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์ส

นอกจากการตอบคำถามวิจัยหลักข้างต้นแล้ว ทางผู้วิจัยจะนำผลลัพธ์ของข้อมูลที่ได้จากข้างต้นไปทำการศึกษาต่อและใช้การวิเคราะห์เชิงสถิติมาช่วยในการหาคำตอบของคำถามวิจัย โดยคำถามวิจัยที่จะทำการศึกษาเพิ่มเติมมีดังนี้

1) ขนาดกลุ่มของผู้ตรวจทานโค้ดส่งผลต่อจำนวนข้อเสนอแนะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์หรือไม่
2) ผู้ตรวจทานโค้ดทำงานเกี่ยวกับการตรวจทานโค้ดในโครงการโอเพนซอร์สมากน้อยเพียงใดต่อสัปดาห์

สาเหตุที่ผู้วิจัยสนใจที่จะทำการตรวจสอบเกี่ยวกับขนาดกลุ่มของนักพัฒนาซอฟต์แวร์เพราะการศึกษางานวิจัยก่อนหน้านี้ (Raymond, 1999) พบว่าในการตรวจทานโค้ดแต่ละครั้งมีควรมีนักพัฒนาซึ่งทำหน้าที่เป็นผู้ตรวจทานโค้ดอย่างน้อย 3 คน หรือได้รับการตรวจทานโค้ดจากชุมชนนักพัฒนาที่เป็นกลุ่มแกนหลักของโครงการโอเพนซอร์ส จากการสำรวจการทำงานของผู้ตรวจทานโค้ดหลายโครงการพบว่า ผู้ตรวจทานโค้ดจำนวน 2 คน ที่มีประสบการณ์ในการพัฒนาซอฟต์แวร์และปฏิบัติงานประจำในกลุ่มโครงการโอเพนซอร์สขนาดใหญ่ คือ จำนวนที่เหมาะสมที่สุดในการตรวจทานโค้ดในแต่ละครั้ง (Sauer, et al., 2000) ซึ่งผลลัพธ์จากการสำรวจข้อมูลดังกล่าวคล้ายกับงานวิจัยเกี่ยวกับระบบปฏิบัติการลินุกซ์ (Linux) และเซิร์ฟเวอร์อาปาเช่ (Apache Server) ที่พบว่าผู้ตรวจทานโค้ดที่ให้คำแนะนำแก่นักพัฒนาซอฟต์แวร์มีค่าเฉลี่ย 2.35 คนต่อการตรวจทานโค้ดแต่ละครั้ง (Lee and Cole, 2003; Rigby and German, 2006) ซึ่งผลลัพธ์ดังกล่าวนี้จะไม่รวมถึงผู้ตรวจทานโค้ดที่ไม่พบข้อบกพร่องในซอร์สโค้ด จากการศึกษางานวิจัยข้างต้นแสดงให้เห็นว่า ผู้ตรวจทานโค้ดตอบสนองต่อการร้องขอให้ตรวจสอบซอร์สโค้ดมากน้อยแค่ไหน ดังนั้นผู้วิจัยจึงสนใจที่จะทำการศึกษาขนาดกลุ่มของผู้ตรวจทานโค้ดเป็นรายเดือน เพื่อแสดงให้เห็นถึงจำนวนของผู้ตรวจทานโค้ดในโครงการ Eclipse และโครงการ Qt ที่ตอบสนองต่อการตรวจทานซอร์สโค้ดของนักพัฒนาและให้คำแนะนำที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ นอกจากนี้ผู้วิจัยยังได้ทำการศึกษาเกี่ยวกับการทำงานของนักพัฒนาซอฟต์แวร์ภายใต้กระบวนการตรวจทานโค้ดพบว่า ช่วงเวลาในการตรวจทานโค้ดเป็นมาตรการสำคัญในการตรวจทานที่มีประสิทธิภาพ (Porter, et al., 1998) โดยความเร็วในให้คำแนะนำของผู้ตรวจทานโค้ดจะขึ้นอยู่กับจำนวนบรรทัดของซอร์สโค้ด ซึ่งหากบรรทัดของซอร์สโค้ดมีจำนวนมากก็ยิ่งทำให้ผู้ตรวจทานโค้ดใช้เวลาในการตรวจทานซอร์สโค้ดเพิ่มมากขึ้น ดังนั้นผู้วิจัยจึงสนใจที่จะตรวจสอบว่าในแต่ละวันของช่วงสัปดาห์ผู้ตรวจทานโค้ดให้คำแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์มากน้อยแค่ไหน

ผู้วิจัยหวังว่าผลที่ได้จากงานวิจัยนี้จะช่วยให้ชุมชนนักวิจัยและนักพัฒนาซอฟต์แวร์สามารถตระหนักถึงความสำคัญของความสามารถในการบำรุงรักษาซอฟต์แวร์และสามารถเข้าใจเกี่ยวกับคุณลักษณะต่าง ๆ ที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาที่มักจะปรากฏในโครงการโอเพนซอร์ส นอกจากนี้ยังเป็นการเพิ่มหลักฐานเชิงประจักษ์เกี่ยวกับคุณภาพของซอฟต์แวร์แก่ชุมชนนักวิจัยที่ให้ความสนใจในโครงการซอฟต์แวร์โอเพนซอร์ส

1.2 วัตถุประสงค์

1.2.1 เพื่อศึกษาข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ภายใต้กระบวนการตรวจทานโค้ดในโครงการโอเพนซอร์ส

1.2.2 เพื่อพัฒนากระบวนการค้นหาและวิธีการที่จะตรวจสอบว่านักพัฒนาให้ความสนใจในการบำรุงรักษาซอฟแวร์หรือไม่

1.2.3 เพื่อทำการสืบค้นและจัดกลุ่มคุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์ส

1.3 ขอบเขตของการวิจัย

ในงานวิจัยนี้ผู้วิจัยได้เลือกโครงการซอฟต์แวร์โอเพนซอร์สที่จะนำมาศึกษาด้วยกันจำนวน 2 โครงการ ได้แก่ โครงการ Eclipse และโครงการ Qt โดยผู้วิจัยจะทำการศึกษาข้อมูลเกี่ยวกับข้อเสนอแนะของโครงการซอฟต์แวร์โอเพนซอร์สดังกล่าวจากระบบเกอริตในช่วงระยะเวลา 5 ปี นั่นคือตั้งแต่ปี ค.ศ. 2012 - 2016 สำหรับสาเหตุหลักที่เลือกโครงการซอฟต์แวร์โอเพนซอร์สทั้ง 2 โครงการนี้เพราะเป็นโครงการที่มีจุดประสงค์ในการใช้งานที่แตกต่างกันและมีการพัฒนาอยู่ตลอดจนถึงปัจจุบัน จึงทำให้แต่ละโครงการมีข้อมูลเกี่ยวกับข้อเสนอแนะจำนวนมาก และจากการศึกษาพบว่าทั้ง 2 โครงการเป็นโครงการที่เปิดโอกาสให้ชุมชนนักวิจัยวิศวกรรมซอฟต์แวร์สามารถเข้าถึงข้อมูล เพื่อนำมาศึกษาวิจัยเกี่ยวกับการตรวจทานโค้ดได้ง่าย (Hamasaki, et al., 2013; Mcintosh, et al., 2014; Kononenko, et al., 2015; Thongtanunam, et al., 2015; Zanjani, et al., 2016) สำหรับคุณลักษณะของแต่ละโครงการที่ทางผู้วิจัยได้เลือกนำมาศึกษาจะแสดงรายละเอียดดังตารางที่ 1.1

ตารางที่ 1.1 รายละเอียดลักษณะของโครงการโอเพนซอร์สทั้ง 2 โครงการ

ชื่อโครงการ	ลักษณะของโครงการ
Eclipse	โครงการ Eclipse เป็นการพัฒนาเครื่องมือสำหรับพัฒนาแอปพลิเคชันหลายภาษา เช่น Java, C/C++, Python, PERL, Ruby ซึ่งโครงการ Eclipse เป็นโครงการโอเพนซอร์สที่ได้รับการพัฒนามาอย่างต่อเนื่องมาตลอดตั้งแต่ปี พ.ศ. 2544 จนถึงปัจจุบัน โดยรุ่นล่าสุดที่พัฒนาในปี พ.ศ. 2560 เรียกว่า “Oxygen projects”
Qt	โครงการ Qt เป็นโครงการโอเพนซอร์สที่ได้รับการพัฒนาต่อเนื่องมาอย่างยาวนานกว่า 20 ปี และเป็นเครื่องมือที่อำนวยความสะดวกในการพัฒนาส่วนต่อประสานกราฟิกกับผู้ใช้ (GUI) โดยการสร้างแอปพลิเคชันให้สามารถรองรับได้หลายระบบปฏิบัติการหรือเรียกว่า “Cross-platform” ที่มีคุณสมบัติในการทำให้โปรแกรมเปลี่ยนไปตามสิ่งแวดล้อมของระบบปฏิบัติการนั้น ๆ โดยไม่จำเป็นต้องเขียนโปรแกรมขึ้นใหม่

ในเบื้องต้นทางผู้วิจัยได้เก็บรวบรวมข้อมูลข้อเสนอแนะทั้งหมดภายใต้การตรวจทานโค้ดจากโครงการซอฟต์แวร์โอเพนซอร์สทั้ง 2 โครงการ โดยรายละเอียดข้อมูลของโครงการโอเพนซอร์สทั้ง 2 โครงการจะถูกแสดงดังตารางที่ 1.2

ตารางที่ 1.2 รายละเอียดของข้อมูลโครงการโอเพนซอร์ส

โครงการโอเพนซอร์ส	Eclipse	Qt
ภาษาที่ใช้ในการพัฒนา	Java	C++
จำนวนข้อเสนอแนะ	108,357	309,165
จำนวนผู้ตรวจทานโค้ด	748	1,059
จำนวนผู้ร้องขอการตรวจทานโค้ด	995	1,204
จำนวนไฟล์ที่มีการร้องขอให้ตรวจทาน	21,588	25,596

1.4 ประโยชน์ของการวิจัย

1.4.1 เพื่อช่วยให้นักพัฒนาซอฟต์แวร์ได้ตระหนักถึงความสำคัญของความสามารถในการบำรุงรักษาซอฟต์แวร์ในโครงการซอฟต์แวร์โอเพนซอร์ส

1.4.2 เพื่อเป็นแนวทางในการปรับปรุงคุณภาพของซอฟต์แวร์ให้กับนักพัฒนารุ่นใหม่ในโครงการโอเพนซอร์สและช่วยให้มีความเข้าใจเกี่ยวกับการตรวจทานโค้ดในโครงการโอเพนซอร์สมากยิ่งขึ้น

1.4.3 เพิ่มหลักฐานเชิงประจักษ์ทางด้านคุณภาพของซอฟต์แวร์ให้กับชุมชนนักวิจัยวิศวกรรมซอฟต์แวร์ที่สนใจโครงการซอฟต์แวร์โอเพนซอร์ส

บทที่ 2

วรรณกรรมและเทคโนโลยีที่เกี่ยวข้อง

ในบทนี้จะเป็นการสรุปความรู้ ทฤษฎี วิธีการ รวมถึงเครื่องมือต่าง ๆ ที่ใช้ในการดำเนินงานวิจัย และการอภิปรายวรรณกรรมที่เกี่ยวข้อง เพื่อนำมาเป็นความรู้พื้นฐานหรือแนวทางที่จะช่วยให้เข้าใจงานวิจัยที่ศึกษานี้ได้มากยิ่งขึ้น โดยมีรายละเอียดดังต่อไปนี้

2.1 หลักการ ความรู้ และเทคโนโลยีที่เกี่ยวข้อง

สำหรับหัวข้อนี้จะเป็นการรวบรวมหลักการ ความรู้ และเทคโนโลยีที่เกี่ยวข้องกับงานวิจัยที่ศึกษา โดยจะเริ่มจาก 1) ซอฟต์แวร์โอเพนซอร์ส 2) ปัจจัยที่มีผลต่อคุณภาพของซอฟต์แวร์โอเพนซอร์ส 3) การตรวจทานโค้ด 4) คุณภาพของซอฟต์แวร์และคุณลักษณะของความสามารถในการบำรุงรักษาซอฟต์แวร์ 5) การทำเหมืองข้อความ และ 6) อัลกอริทึม Latent Dirichlet Allocation โดยมีรายละเอียดในแต่ละส่วนดังนี้

2.1.1 ซอฟต์แวร์โอเพนซอร์ส

ซอฟต์แวร์โอเพนซอร์ส หมายถึง ซอฟต์แวร์ที่เปิดโอกาสให้นักพัฒนาหรือผู้ใช้ทั่วไปมีอิสระในการใช้งาน เผยแพร่โปรแกรมต้นฉบับ และอนุญาตให้บุคคลอื่นสามารถแก้ไขดัดแปลงซอร์สโค้ดหรือนำเอาระบบนั้น ๆ ไปพัฒนาต่อได้ตามที่ต้องการภายใต้เงื่อนไขของการอนุญาตที่กำหนดไว้ (License) (Tiwari, 2010) ซึ่งซอฟต์แวร์โอเพนซอร์สที่ผู้ใช้สามารถดาวน์โหลดได้จากเว็บไซต์ที่ให้บริการในการ

จัดเก็บซอร์สโค้ดและบริหารจัดการโครงการโอเพนซอร์ส ตัวอย่างเช่น GitHub[5] Sourceforge[6] Google code[7] Codeplex[8] และ Launchpad[9]

ในปัจจุบันซอฟต์แวร์โอเพนซอร์สมีผู้ใช้งานอย่างแพร่หลายทั้งในอุตสาหกรรม หน่วยงานภาครัฐ และเอกชน รวมไปถึงบุคคลทั่วไป เนื่องจากซอฟต์แวร์โอเพนซอร์สเป็นซอฟต์แวร์ทางเลือกที่ช่วยลดค่าใช้จ่ายด้านลิขสิทธิ์ ซึ่งต่างจากซอฟต์แวร์ลิขสิทธิ์ที่นอกจากผู้ใช้ต้องเสียเงินซื้อแล้วยังไม่สามารถนำมาพัฒนาต่อยอดหรือนำไปแก้ไขปรับปรุงให้ตรงกับความต้องการของตนเองหรือองค์กรได้ เพราะซอฟต์แวร์ลิขสิทธิ์ส่วนใหญ่มักจะไม่มีการเปิดเผยซอร์สโค้ดให้ผู้ใช้งานนั่นเอง ถึงแม้ว่าซอฟต์แวร์โอเพนซอร์สบางตัวอาจมีค่าลิขสิทธิ์ แต่การเปิดเผยซอร์สโค้ดหรือโปรแกรมต้นฉบับก็ช่วยประหยัดเวลาในการพัฒนาโดยไม่ต้องเริ่มต้นใหม่ทั้งหมดเพราะผู้ใช้งานสามารถทำการวิเคราะห์ซอฟต์แวร์ที่มีมาตรฐานการเขียนขั้นพื้นฐานสู่การประยุกต์เพื่อการใช้งานได้รวดเร็วขึ้น พร้อมทั้งช่วยให้การพัฒนาต่อยอดสำหรับการดำเนินธุรกิจได้ในระยะเวลาสั้น ด้วยเหตุนี้โครงการที่ส่งเสริมโอเพนซอร์สจึงได้ร่วมมือกับกลุ่มผู้ประกอบธุรกิจที่เกี่ยวข้องกับเทคโนโลยี เพื่อช่วยกันเพิ่มมูลค่าผลิตภัณฑ์ ลดการละเมิดลิขสิทธิ์ และพัฒนาซอฟต์แวร์ให้สอดคล้องกับความต้องการของผู้ใช้งานหรือตามความสนใจของผู้ประกอบการทางด้านธุรกิจซอฟต์แวร์ นอกจากนี้ยังมีการถ่ายทอดความรู้ทางด้านการผลิตซอฟต์แวร์ เครื่องมือหรือกระบวนการจัดการที่ช่วยสนับสนุนการพัฒนาซอฟต์แวร์ รวมไปถึงความรู้เกี่ยวกับการประยุกต์ใช้งานโอเพนซอร์สในรูปแบบต่าง ๆ และยังส่งเสริมให้ผู้ใช้งานมีความมั่นใจในการเลือกใช้งานซอฟต์แวร์โอเพนซอร์สเพิ่มมากขึ้น ด้วยคุณลักษณะดังกล่าวข้างต้นทำให้ซอฟต์แวร์โอเพนซอร์สได้รับการยอมรับและเป็นที่รู้จักกันอย่างแพร่หลาย

ในรอบหลายปีที่ผ่านมาองค์กรต่าง ๆ มีการนำซอฟต์แวร์โอเพนซอร์สมาประยุกต์ใช้ เพื่อเพิ่มประสิทธิภาพการทำงานขององค์กร เช่น การนำซอฟต์แวร์สำนักงาน (Openoffice.org) มาแทนซอฟต์แวร์ของบริษัทไมโครซอฟต์ อย่างเช่น Microsoft Office (สมคิด ทุ่นใจ, 2559) นอกจากนี้ยังมีซอฟต์แวร์โอเพนซอร์สที่ได้รับความนิยมจากผู้ใช้งานทั่วโลก ตัวอย่างเช่น ระบบปฏิบัติการลินุกซ์ (Linux) โปรแกรมเว็บเบราว์เซอร์มอซิลลาไฟร์ฟอกซ์ (Mozilla Firefox) โปรแกรมระบบจัดการฐานข้อมูลมายเอสคิวแอล (MySQL Database) และเว็บเซิร์ฟเวอร์อาปาเช่ (Apache Web Server) (Mockus, et al., 2002) โดยจะเห็นได้จากการบริการต่าง ๆ ในรูปแบบออนไลน์ซึ่งจะอาศัยซอฟต์แวร์โอเพนซอร์สเป็น

---

[5] www.github.com

[6] www.sourceforge.net

[7] https://code.google.com/

[8] https://www.codeplex.com/

[9] https://launchpad.net/launchpad-project

จำนวนมาก เช่น Google, Amazon, eBay, Wikipedia (อ้างอิงจากเว็บ http://www.economist.com/displaystory.cfm?story_id=E1_VGNQJQQ, 5 กันยายน 2559) รวมถึงเว็บไซต์ในปัจจุบันมากกว่า 50% ทำงานโดยอาศัยเว็บเซิร์ฟเวอร์อาปาเช่ โดยทั่วไปซอฟต์แวร์โอเพสซอร์สถูกจำแนกออกเป็นกลุ่ม (Wahyudin, et al., 2007) โดยอาศัยเกณฑ์ดังต่อไปนี้

1) การนำไปใช้งาน (Application Domain) – Sourceforge ซึ่งเป็นผู้ให้บริการเผยแพร่ซอฟต์แวร์โอเพนซอร์สรายใหญ่ของโลกได้แบ่งกลุ่มการใช้งานของซอฟต์แวร์โอเพนซอร์สออกเป็น 18 กลุ่ม (Capiluppi, et al., 2003) โดยกลุ่มการใช้งานซอฟต์แวร์ที่ได้รับความนิยมเป็น 5 อันดับแรก ได้แก่ โปรแกรมประยุกต์หรือเว็บแอปพลิเคชัน (Internet Application) การพัฒนาซอฟต์แวร์ (Software Development) ระบบ (System) การสื่อสาร (Communication) และเกมส์/บันเทิง (Game/Entertainment)

2) จำนวนของนักพัฒนาในโครงการ (Project Size) – Stamelos และคณะ ได้แบ่งกลุ่มของซอฟต์แวร์โอเพนซอร์สตามจำนวนของทีมนักพัฒนาออกเป็น 6 กลุ่ม แสดงดังตารางที่ 2.1 (Stamelos, et al., 2002) และผลจากการศึกษาของ Comino และคณะ ได้มีการเปิดเผยว่ามากกว่า 80% ของโครงการโอเพนซอร์สในแต่ละโครงการมีนักพัฒนาน้อยกว่า 6 คน และน้อยกว่า 1% ที่โครงการโอเพนซอร์สมีนักพัฒนามากกว่า 16 คน (Comino, et al., 2005)

ตารางที่ 2.1 การจำแนกกลุ่มตามจำนวนของนักพัฒนาในโครงการโอเพนซอร์ส

กลุ่ม	จำนวนนักพัฒนา / คน
1	มากกว่า 16
2	7-15
3	5-6
4	3-4
5	2
6	1

3) การให้การสนับสนุนโครงการ (Project Sponsorship) – ส่วนมากแล้วโครงการซอฟต์แวร์โอเพนซอร์สนั้น จะอาศัยนักพัฒนาที่ทำงานด้วยความสมัครใจโดยที่ไม่ได้รับค่าตอบแทน (Aberdour, 2007) แต่ในบางโครงการจะมีผู้ให้การสนับสนุน โดยที่นักพัฒนาจะได้รับค่าตอบแทนจากผู้สนับสนุนโดยบริษัทเท่านั้น ตัวอย่างเช่น โครงการ JBoss, Apache JackRabbit, Myfaces, Sourcefire หรือ OpenOffice โดยโครงการเหล่านี้เป็นโครงการที่พัฒนาฟังก์ชันหรือคุณสมบัติบางส่วนของเครื่องมือที่ทางผู้สนับสนุนต้องการ

2.1.2 ปัจจัยที่มีผลต่อคุณภาพของซอฟต์แวร์โอเพนซอร์ส

คุณภาพเป็นสิ่งสำคัญที่สุดในการผลิตซอฟต์แวร์ โดยทั่วไปคำว่า “คุณภาพ” หมายถึงผลิตภัณฑ์ที่สามารถตอบสนองความต้องการของผู้ใช้ แต่สำหรับผลิตภัณฑ์ซอฟต์แวร์แล้ว คุณภาพยังรวมไปถึงคุณลักษณะในด้านอื่น ๆ ของซอฟต์แวร์อีกด้วย ดังนั้นในการผลิตซอฟต์แวร์จึงต้องมีกระบวนการจัดการคุณภาพซอฟต์แวร์ (Software Quality Management) ซึ่งเป็นกิจกรรมหนึ่งที่มีการวางแผนอย่างเป็นระบบตั้งแต่ระยะเริ่มแรกของการพัฒนาซอฟต์แวร์ โดยนักพัฒนาซอฟต์แวร์จะต้องกำหนดวิธีการสร้างคุณภาพให้กับซอฟต์แวร์ในแต่ละขั้นตอนของการผลิต เพื่อรับประกันว่าระบบจะมีลักษณะเป็นไปตามมาตรฐานคุณภาพที่กำหนดไว้ ดังนั้นการตรวจสอบคุณภาพของซอฟต์แวร์ถือว่าเป็นปัจจัยหนึ่งที่จะนำไปสู่ความสำเร็จของโครงการที่สามารถพัฒนาผลิตภัณฑ์ซอฟต์แวร์ได้อย่างมีประสิทธิภาพในด้านการใช้งาน

เพื่อให้องค์กรหรือผู้ใช้ทั่วไปที่นำซอฟต์แวร์ไปใช้มีความมั่นใจในการใช้งาน และเป็นการเพิ่มคุณภาพของซอฟต์แวร์โอเพนซอร์ส ชุมชนนักพัฒนาซอฟแวร์ในโครงการโอเพนซอร์สจึงได้ค้นหาวิธีที่จะทำให้คุณภาพของซอฟต์แวร์อยู่ในระดับที่องค์กรหรือเจ้าของโครงการยอมรับได้ โดยงานวิจัยของ Mark Aberdour (2007) และงานวิจัยอื่น ๆ (Stamelos, et al., 2002; O'Reilly, 1999) ได้ระบุถึงปัจจัยที่มีผลต่อคุณภาพของโครงการซอฟต์แวร์โอเพนซอร์ส ซึ่งประกอบด้วยกันจำนวน 5 ปัจจัย ดังนี้

1) ความยั่งยืนของชุมชนนักพัฒนาซอฟต์แวร์ (The Sustainable Community)

ซอฟต์แวร์โอเพนซอร์สเป็นโครงการที่มีการพัฒนาอย่างต่อเนื่อง เพราะได้รับการสนับสนุนจากชุมชนนักพัฒนา (Gamalielsson and Lundell, 2012) โดยชุมชนนักพัฒนาเหล่านี้ คือกลุ่มคนที่ต้องการมีส่วนร่วมในการพัฒนาองค์ความรู้และความชำนาญในกระบวนการผลิตซอฟต์แวร์ ด้วยเหตุนี้ทำให้ในปัจจุบันโครงการซอฟต์แวร์โอเพนซอร์สมีเครือข่ายกลุ่มสังคมย่อยจำนวนมากที่ร่วมกันพัฒนาในแต่ละโครงการ (Latent Social Structure) (Bird, et al., 2008) และโครงการซอฟต์แวร์โอเพนซอร์สถือว่าเป็นแหล่งรวบรวมข้อมูลในการแลกเปลี่ยนความรู้และมีการสร้างสรรค์กระบวนการหรือวิธีการใหม่ ๆ เพื่อให้ชุมชนนักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์สมีหลักปฏิบัติที่จะช่วยสนับสนุนการทำงานได้อย่างมีประสิทธิภาพมากยิ่งขึ้น โดยสามารถพัฒนาโปรแกรมที่มีคุณภาพ และสร้างคุณสมบัติหรือคุณลักษณะใหม่ของระบบได้ตามความต้องการหรือวัตถุประสงค์ของผู้ใช้งาน

2) การมีส่วนร่วมและแรงจูงใจของนักพัฒนา (Participation and Motivation)

โครงการซอฟต์แวร์โอเพนซอร์สเกิดจากการรวมตัวของนักพัฒนาซอฟต์แวร์จนกลายเป็นกลุ่มคนที่มีความชอบ ความสนใจ และความเชี่ยวชาญในเรื่องเดียวกันหรือที่เรียกกันว่า กลุ่มที่มีองค์ความรู้เฉพาะทางด้านการพัฒนาระบบ (Knowledge Domain) ซึ่งสมาชิกในกลุ่มมีความสมัครใจที่จะเรียนรู้และแลกเปลี่ยนประสบการณ์ซึ่งกันและกัน (Ye and Kishida, 2003) โดยอาจมีการนัดพบเจอกัน

เช่น การประชุม สัมมนา หรือแม้กระทั่งการจัดฝึกอบรมเชิงปฏิบัติการเกี่ยวกับการเขียนโปรแกรม เพื่อพัฒนาศักยภาพของชุมนักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์ส นอกจากนี้ยังมีการพบปะกันผ่านทางเครื่องมือหรือเทคโนโลยี ได้แก่ ออนไลน์ผ่านทางอินเทอร์เน็ต เพื่อการแลกเปลี่ยนความรู้และประสบการณ์จากการทำงานผ่านเว็บบอร์ด เป็นต้น การที่นักพัฒนาซอฟต์แวร์ได้ทำการผลิตซอฟต์แวร์ร่วมกันจะทำให้เกิดการเสริมสร้างความรู้สึกในการมีส่วนร่วมและความเป็นเจ้าของร่วมกัน จึงก่อให้เกิดความมุ่งมั่นตั้งใจที่จะดำเนินการพัฒนาซอฟต์แวร์ และส่งผลให้ซอร์สโค้ดหรือซอฟต์แวร์ที่ได้รับการพัฒนานั้นมีคุณภาพที่ดี

3) การแบ่งโมดูลของโค้ด (Code Modularity)

ในการพัฒนาซอฟต์แวร์ที่มีขนาดใหญ่และมีความซับซ้อนมาก โดยทั่วไปจะมีการทำงานแบบแยกฟังก์ชันหรือโมดูลออกเป็นหลายส่วน เพื่อให้ง่ายต่อการทำงานของชุมชนนักพัฒนาซอฟต์แวร์ เนื่องจากนักพัฒนาซอฟต์แวร์สามารถพัฒนาระบบได้โดยที่ไม่ต้องรอฟังก์ชันการทำงานของนักพัฒนาคนอื่น ๆ (Bosu and Carver, 2013) อีกทั้งยังง่ายต่อการบำรุงรักษาซอฟต์แวร์เพราะสามารถทำการแก้ไขข้อผิดพลาดภายในฟังก์ชันย่อย ๆ ก่อนที่จะรวมกันเป็นโปรแกรมเดียว และยังเป็นการอำนวยความสะดวกในอนาคต โดยสามารถนำฟังก์ชันย่อย ๆ เหล่านั้นกลับมาใช้งานอีกครั้ง (Reuse) ซึ่งถือว่าเป็นการช่วยลดเวลาและค่าใช้จ่ายในกระบวนการพัฒนาซอฟต์แวร์ เพราะไม่ต้องเสียเวลาในการเขียนฟังก์ชันที่มีการทำงานที่คล้ายกันขึ้นมาใหม่

4) การตรวจทานโค้ดโดยนักพัฒนาผู้อื่น (Peer Code Review)

เพื่อที่จะลดความเสี่ยงของปัญหาที่เกิดขึ้นในระหว่างกำลังดำเนินการพัฒนาซอฟต์แวร์ การตรวจทานโค้ดเป็นวิธีที่จะช่วยเพิ่มประสิทธิผลของซอฟต์แวร์และคุณภาพของซอร์สโค้ด (Hamasaki, et al., 2013) เพื่อให้นักพัฒนาสามารถตรวจสอบปัญหาที่เกิดขึ้นภายในระบบหรือแก้ไขโค้ดในส่วนที่เกิดข้อผิดพลาดได้อย่างเหมาะสม โครงการซอฟต์แวร์โอเพนซอร์สจึงได้กำหนดให้มีการตรวจทานโค้ด โดยอาศัยนักพัฒนาซอฟต์แวร์ที่ไม่ใช่ผู้เขียนโค้ดทำหน้าที่ตรวจทานซอร์สโค้ดที่ได้พัฒนาขึ้น ในการร้องขอการตรวจทานโค้ด นักพัฒนาจะทำการร้องขอไปยังนักพัฒนาคนอื่น ๆ ที่มีทักษะและประสบการณ์เกี่ยวกับการเขียนโค้ดที่อยู่ในระดับเดียวกันหรือใกล้เคียงกันเป็นผู้ตรวจทาน เพราะสามารถตรวจสอบซอร์สโค้ดเพื่อหาข้อผิดพลาดหรือข้อบกพร่องได้อย่างรวดเร็วและให้ข้อเสนอแนะในการแก้ไขปรับปรุงซอร์สโค้ดได้ดีกว่านักพัฒนาที่มีทักษะและประสบการณ์ที่น้อยกว่าผู้เขียนโค้ด

5) กระบวนการทดสอบ (Testing Process)

การทดสอบซอฟต์แวร์ คือ กระบวนการตรวจสอบความถูกต้องของระบบที่พัฒนาเพื่อให้ได้ซอฟต์แวร์ที่มีประสิทธิภาพการทำงานตรงตามความต้องการของผู้ใช้งานหรือเป็นไปตามข้อกำหนดความต้องการเชิงซอฟต์แวร์ที่ได้ระบุไว้ (Scacchi, 2002) โดยใช้เทคนิคหรือวิธีการทดสอบ ซึ่งการทดสอบซอฟต์แวร์มีจุดประสงค์เพื่อตรวจสอบปัญหาที่อาจส่งผลกระทบต่อการทำงานของระบบ โดย

การค้นหาข้อผิดพลาดหรือข้อบกพร่องที่เกิดขึ้นภายในระบบเพื่อทำการแก้ไขก่อนที่จะส่งมอบให้แก่ลูกค้าหรือผู้ใช้งาน ดังนั้นกระบวนการทดสอบซอฟต์แวร์จึงเป็นขั้นตอนหนึ่งในวงจรชีวิตการพัฒนาซอฟต์แวร์ (Software Development Life Cycle : SDLC) ตัวอย่างรูปแบบการทดสอบ เช่น การทดสอบหน่วยย่อยฟังก์ชัน (Unit Testing) การทดสอบแบบบูรณาการ (Integration Testing) และการทดสอบระบบ (System Testing) โดยการทดสอบแต่ละรูปแบบจะมีการทดสอบที่แตกต่างกัน เพื่อตรวจสอบคุณภาพของระบบที่พัฒนาขึ้นได้อย่างคลอบคลุม

นอกจากเทคนิคหรือวิธีต่าง ๆ ที่ชุมชนนักพัฒนาในโครงการซอฟต์แวร์โอเพนซอร์ส ได้นำมาประยุกต์ใช้เพื่อเพิ่มคุณภาพของซอฟต์แวร์แล้ว ปัจจัยที่มีผลต่อคุณภาพของซอฟต์แวร์ทั้ง 5 ข้อข้างต้นก็มีส่วนที่จะช่วยให้โครงการโอเพนซอร์สมีคุณภาพที่เพิ่มสูงขึ้นเช่นกัน โดยเฉพาะปัจจัยข้อที่ 4 การตรวจทานโค้ด เนื่องจากงานวิจัยนี้เป็นการศึกษาเกี่ยวกับการตรวจทานโค้ดในโครงการโอเพนซอร์ส ซึ่งการตรวจทานโค้ดเป็นวิธีการหนึ่งที่เป็นมาตรฐานในการตรวจสอบการทำงานของระบบ เพื่อให้เป็นไปตามวัตถุประสงค์ของโครงการก่อนที่จะถูกนำไปเผยแพร่แก่สาธารณชน อีกทั้งยังเป็นวิธีการที่ช่วยอำนวยความสะดวกในการปรับปรุงซอร์สโค้ด โดยเฉพาะทางด้านการบำรุงรักษาซอฟต์แวร์

2.1.3 การตรวจทานโค้ด

การตรวจทานโค้ด (Code Review) เป็นวิธีการที่ช่วยในการลดความเสี่ยงที่จะเกิดข้อผิดพลาดและปรับปรุงคุณภาพของซอฟต์แวร์ โดยการตรวจสอบซอร์สโค้ดเพื่อค้นหาข้อบกพร่องที่อาจเกิดขึ้นในระหว่างการพัฒนาซอฟต์แวร์ และทำให้ซอร์สโค้ดที่พัฒนาเป็นโค้ดที่มีมาตรฐาน (Bernhart, et al., 2010) โดยเป้าหมายหลักของการตรวจทานโค้ด (อ้างอิงจากเว็บ https://www.codeproject.com/Articles/524235/Codeplusreviewplusguidelines, 20 กุมภาพันธ์ 2561) คือ

1) ตรวจสอบและแก้ไขข้อบกพร่องที่พบระหว่างการพัฒนาระบบ

2) ทำให้ซอร์สโค้ดเข้าใจได้ง่าย

3) การแก้ไขปรับปรุงซอฟต์แวร์ให้สอดคล้องกับการออกแบบและการใช้งาน

4) ช่วยลดการทำงานที่เกิดความซ้ำซ้อนภายในระบบ

5) สร้างความมั่นใจเกี่ยวกับคุณภาพระหว่างการดำเนินงานแก่ผู้มีส่วนได้ส่วนเสีย

6) ช่วยสร้างเสริมและพัฒนาศักยภาพของเพื่อนร่วมทีมพัฒนาซอฟต์แวร์

7) สร้างความภาคภูมิใจในการทำงานของนักพัฒนาซอฟต์แวร์ เพราะการเขียนซอร์สโค้ดถือว่าเป็นรางวัลที่สำคัญสำหรับนักพัฒนาซอฟต์แวร์จำนวนมาก

8) การทำงานร่วมกันของนักพัฒนาซอฟต์แวร์จะช่วยให้สมาชิกในทีมมีความสามัคคีและมีความใกล้ชิดกันมากขึ้น

เมื่อองค์กรหรือโครงการเกี่ยวกับการพัฒนาซอฟต์แวร์นำการตรวจทานโค้ดนี้ไปประยุกต์ใช้ก็จะช่วยให้เกิดความมั่นใจว่าซอร์สโค้ดที่ได้ทำการพัฒนาขึ้นมานั้นตรงกับคุณสมบัติที่ได้ระบุไว้หรือตรงกับความต้องการที่ผู้ใช้งานพึงพอใจ รวมถึงช่วยลดผลกระทบที่อาจเกิดขึ้นระหว่างการแก้ไขซอร์สโค้ด ดังนั้นการตรวจทานโค้ดถือว่าเป็นหลักปฏิบัติที่ดีที่สุดสำหรับวิศวกรรมซอฟต์แวร์มานานกว่า 35 ปี (Fagan, 1976; Broy and Denert, 2002) โดยทั่วไปการตรวจทานโค้ดนี้จะมีวิธีการตรวจทานโค้ดด้วยกัน 3 วิธี ได้แก่ 1) การตรวจทานโค้ดแบบละเอียด 2) การตรวจทานโค้ดแบบเฉพาะทีม และ 3) การตรวจทานโค้ดสมัยใหม่ โดยในแต่ละเทคนิคมีรายละเอียดดังตารางที่ 2.2

ตารางที่ 2.2 รายละเอียดของเทคนิคการตรวจทานโค้ด

ลำดับ	เทคนิค	รายละเอียด
1.	การตรวจทานโค้ดแบบละเอียด	การตรวจทานโค้ดแบบดั้งเดิมที่มักจะใช้ในงานทางด้านวิศวกรรมซอฟต์แวร์ คือ การตรวจทานโค้ดแบบละเอียด (Code Inspections) หรือที่เรียกกันว่า การตรวจทานโค้ดอย่างเป็นทางการ (Formal Inspection) ซึ่งถูกคิดค้นโดย Fagan (Fagan, 1976) โดยการตรวจทานโค้ดนี้จะมีรูปแบบขั้นตอนการดำเนินงาน คือ การวางแผนเพื่อระบุสิ่งที่ต้องการจะตรวจสอบ การเตรียมการเกี่ยวกับขั้นตอนในการตรวจสอบซอร์สโค้ด ตรวจสอบการทำงานซ้ำ พร้อมทั้งติดตามผลการดำเนินงาน ซึ่งหลักการปฏิบัติเหล่านี้จะเกิดขึ้นได้จากการนัดพบสมาชิกทีมพัฒนาซอฟต์แวร์ โดยมีการระบุเวลาในการประชุมเพื่อทำการตรวจทานซอร์สโค้ดล่วงหน้า (Czerwonka and Greiler, 2015) และผู้เข้าร่วมจะต้องอยู่ในสถานที่เดียวกัน ในกระบวนการตรวจทานโค้ดนี้ ผู้เขียนโค้ดจะให้นักพัฒนาซอฟต์แวร์คนอื่น ๆ ในทีมช่วยกันตรวจทานโค้ด เพื่อที่จะได้รับข้อเสนอแนะเกี่ยวกับข้อผิดพลาดที่อาจจะเกิดขึ้นในอนาคต การเขียนโค้ดให้เป็นมาตรฐาน และปัญหาอื่น ๆ ที่พบในระบบ อีกทั้งในขณะที่กำลังตรวจทานจะมีการจดบันทึกรายละเอียดของข้อบกพร่องที่พบในแต่ละบรรทัด ทำให้เป็นเรื่องที่ค่อนข้างยุ่งยากและใช้เวลานาน (Bacchelli and Bird, 2013)

ตารางที่ 2.2 รายละเอียดของเทคนิคการตรวจทานโค้ด (ต่อ)

ลำดับ	เทคนิค	รายละเอียด
2.	การตรวจทานโค้ดแบบเฉพาะทีม	การตรวจทานโค้ดแบบเฉพาะทีม (Walkthrough) (Oh and Choi, 2005; Ciolkowski, et al., 2002) เป็นการตรวจทานโค้ดในลักษณะเดียวกับการตรวจทานแบบละเอียด แต่มีความแตกต่างกัน คือ การนัดเวลาประชุมเพื่อตรวจทานโค้ดจะถูกกำหนดโดยนักพัฒนาซอฟต์แวร์ที่ทำการเขียนโค้ด หรือไม่จำเป็นต้องมีการนัดประชุมล่วงหน้าและไม่ต้องมีการจดบันทึกรายละเอียดของการตรวจทานโค้ดอย่างเป็นทางการเหมือนกับการตรวจทานโค้ดแบบละเอียด เพียงแค่ให้นักพัฒนาที่เป็นผู้เขียนโค้ดทราบถึงข้อบกพร่องและข้อเสนอแนะที่ได้จากผู้ตรวจทานที่เป็นสมาชิกในทีมพัฒนา
3.	การตรวจทานโค้ดสมัยใหม่	การตรวจทานโค้ดในระดับเดียวกัน (Peer Code Review) หรือการตรวจทานโค้ดสมัยใหม่ (Modern Code Review) เป็นหลักปฏิบัติทางด้านวิศวกรรมซอฟต์แวร์ที่องค์กรและชุมชนโอเพนซอร์สนำมาใช้ในการปรับปรุงคุณภาพของซอร์สโค้ด (Thongtanunam, et al., 2015) โดยการตรวจทานโค้ดลักษณะนี้จะเป็นวิธีการที่อำนวยความสะดวกในการแบ่งปันความรู้เกี่ยวกับการพัฒนาระบบเป็นอย่างมาก (Tymchuk, et al., 2015) เพราะมีการใช้เครื่องมือหรือเทคโนโลยีที่ให้บริการในการตรวจทานโค้ดเป็นสื่อกลางในการติดต่อสื่อสาร อีกทั้งการตรวจทานโค้ดสมัยใหม่ยังเป็นวิธีการที่ทำให้ซอร์สโค้ดอยู่ในรูปแบบที่เป็นมาตรฐาน โดยการลบฟังก์ชันการทำงานที่ซ้ำซ้อนและกำจัดโค้ดที่ไม่เกี่ยวข้องหรือไม่มีความจำเป็นต่อระบบ

เมื่อผู้วิจัยได้ทำการศึกษาเกี่ยวกับการตรวจทานโค้ดสมัยใหม่พบว่า โดยทั่วไปการตรวจทานโค้ดสมัยใหม่นี้จะมีด้วยกัน 2 รูปแบบ โดยแต่ละรูปแบบมีรายละเอียดดังต่อไปนี้

1) การตรวจทานโค้ดโดยไม่ใช้เครื่องมือ (Manual Code Review)

การตรวจทานโค้ดโดยนักพัฒนาซอฟต์แวร์ด้วยกันเองจะทำให้เกิดการแลกเปลี่ยนความรู้ระหว่างนักพัฒนาซอฟต์แวร์ที่ทำการพัฒนาโค้ดและนักพัฒนาคนอื่น ๆ ในทีมที่เป็นผู้ตรวจทานโค้ด โดยอาศัยการสื่อสารในรูปแบบออนไลน์ต่าง ๆ ซึ่งผู้ที่ทำหน้าที่ตรวจทานโค้ดจะทำการตรวจทานด้วยวิธีการอ่านโค้ดทีละบรรทัด เพื่อที่จะสามารถตรวจสอบฟังก์ชันการทำงานได้อย่างครอบคลุมและให้ข้อเสนอแนะที่จะช่วยให้นักพัฒนาสามารถแก้ไขโค้ดได้อย่างถูกต้อง พร้อมทั้งทำให้ระบบหรือฟังก์ชันนั้นสมบูรณ์มากยิ่งขึ้น โดยส่วนใหญ่ผู้ที่ทำหน้าที่ตรวจทานโค้ดจะเป็นผู้ที่มีความเชี่ยวชาญและมีประสบการณ์ทางด้านการพัฒนาซอฟต์แวร์ในระดับเดียวกันหรือมากกว่านักพัฒนาซอฟต์แวร์ที่เป็นผู้ร้องขอการตรวจทานโค้ด

2) การตรวจทานโค้ดโดยใช้เครื่องมือที่ให้บริการในการตรวจทานโค้ดแบบอัตโนมัติ (Automatic Code Review หรือ Automatically Recommending Peer Reviewers)

เพื่อการพัฒนาซอฟต์แวร์ให้มีประสิทธิภาพ นักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์สจึงได้มีการใช้เครื่องมือที่ช่วยในการตรวจสอบซอร์สโค้ดที่ได้พัฒนาขึ้น ตัวอย่างเช่น Collaboration ใน Eclipse[10] และ ReviewPal ใน Visual Studio[11] ที่เป็น IDEs หรือส่วนเสริมของระบบ (Plug-in) ที่สามารถติดตั้งเพื่ออำนวยความสะดวกในการตรวจทานโค้ด ซึ่งการใช้เครื่องมือที่ช่วยในการตรวจทานโค้ดแบบอัตโนมัติเหล่านี้จะช่วยให้ทีมนักพัฒนาซอฟต์แวร์สามารถทำงานได้อย่างมีประสิทธิภาพ เช่น การค้นหาข้อผิดพลาดที่เกิดขึ้น พร้อมแสดงบรรทัดหรือสาเหตุที่ทำให้เกิดข้อผิดพลาด สามารถแนะนำการแก้ไขหรือแนวทางในการปรับปรุงซอร์สโค้ดเบื้องต้น พร้อมทั้งอำนวยความสะดวกในการสร้างรายงานเกี่ยวกับรายละเอียดของข้อเสนอแนะ เพื่อให้ง่ายต่อการแก้ไขปรับปรุงฟังก์ชันการทำงานของระบบ

การตรวจทานโค้ดสมัยใหม่ทั้ง 2 รูปแบบนี้ อาจกล่าวได้ว่าการใช้เครื่องมือตรวจทานโค้ดแบบอัตโนมัติจะสามารถช่วยลดเวลาในการตรวจสอบซอร์สโค้ดได้มากกว่าการตรวจทานโค้ดโดยทีมนักพัฒนาซอฟต์แวร์ แต่เครื่องมือเหล่านี้ก็ยังไม่สามารถแทนการตรวจทานโค้ดโดยนักพัฒนาด้วยกันเองได้ เนื่องจากข้อเสนอแนะที่ได้รับเป็นเพียงข้อเสนอแนะหรือการปรับปรุงซอร์สโค้ดพื้นฐาน ส่วนการตรวจทานโค้ดโดยนักพัฒนาด้วยกันเองจะให้ข้อเสนอแนะหรือการปรับปรุงซอร์สโค้ดในเชิงลึก เพราะผู้ตรวจทานโค้ดจะอาศัยประสบการณ์ด้านการพัฒนาซอฟต์แวร์ร่วมด้วย

---

[10] http://eclipse.smartbearsoftware.com/

[11] https://www.codeproject.com/articles/43490/code-review-plug-in-for-visual-studio-reviewp

จากเทคนิคหรือวิธีการเกี่ยวกับการตรวจทานโค้ดที่ได้กล่าวไว้ข้างต้น ผู้วิจัยได้ทำการสรุปและอธิบายวิธีการต่าง ๆ ดังตารางที่ 2.3

ตารางที่ 2.3 การเปรียบเทียบเทคนิคในการตรวจทานโค้ด

ลำดับ	เทคนิคการตรวจทานโค้ด	การเตรียมตัว	รูปแบบการตรวจทาน	ค่าใช้จ่าย
1.	การตรวจทานแบบละเอียด	มี	การประชุมกลุ่ม	สูง
2.	การตรวจทานแบบเฉพาะทีม	ไม่มี	การประชุมกลุ่ม	ปานกลาง
3.	การตรวจทานโค้ดสมัยใหม่ 3.1) การตรวจทานโดยนักพัฒนาด้วยกันเอง 3.2) การตรวจทานแบบอัตโนมัติ	 ไม่มี ไม่มี	 รายบุคคล เครื่องมือ	 น้อย น้อยมาก

สำหรับงานวิจัยนี้จะมุ่งเน้นไปที่การตรวจทานโค้ดสมัยใหม่ เพราะเป็นวิธีการที่ชุมชนนักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์สนิยมนำมาประยุกต์ใช้งานและเป็นแนวปฏิบัติที่เน้นความสะดวกรวดเร็วในการตรวจทาน จากการศึกษาเกี่ยวกับการตรวจทานโค้ดสมัยใหม่พบว่า ชุมชนนักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์สและองค์กรชั้นนำทั่วโลกมีการใช้เครื่องมือในการตรวจทานโค้ดเพื่อตรวจสอบข้อบกพร่องของซอร์สโค้ดและใช้เป็นสื่อกลางในการติดต่อสื่อสารเพื่อแลกเปลี่ยนความรู้ ซึ่งเครื่องมือที่ได้รับความนิยม ได้แก่ CodeFlow[12], Gerrit, Collaborator[13], Crucible[14], Differential[15], Github pull request[16], Review Board[17], และ Upsource[18] โดยตารางที่ 2.4 จะแสดงรายละเอียดของเครื่องมือที่ใช้ในการตรวจทานโค้ด (Code Review Tools) ที่ผู้วิจัยต้องทำการศึกษาเพื่อเลือกใช้ให้เหมาะสมต่องานวิจัย

---

[12] https://channel9.msdn.com/Series/Visual-Studio-2012-Premium-and-Ultimate-Overview/Visual-Studio-Ultimate-2012-Using-Code-Review-to-Improve-Qualities

[13] https://smartbear.com/product/collaborator/overview/

[14] https://www.atlassian.com/software/crucible

[15] http://phabricator.org/applications/dierential

[16] https://help.github.com/articles/about-pull-requests/

[17] https://www.reviewboard.org

[18] https://www.jetbrains.com/upsource

ตารางที่ 2.4 รายละเอียดของเครื่องมือที่ใช้ในการตรวจทานโค้ด

ชื่อเครื่องมือ	พัฒนาโดย	รายละเอียด
CodeFlow	Microsoft	CodeFlow เป็นเครื่องมือการตรวจทานโค้ดที่ใช้ภายในบริษัท Microsoft โดยให้นักพัฒนาและผู้เข้าร่วมสามารถตรวจสอบโค้ดในรูปแบบการสนทนาออนไลน์ ซึ่งนักพัฒนาสามารถจัดการไฟล์ (สร้าง, ลบ และแก้ไข) เลือกผู้ตรวจทานโค้ด และส่งไฟล์ไปยังที่ให้บริการ (Service) การตรวจทานโค้ดของ CodeFlow หลังจากนั้น CodeFlow จะทำการแจ้งผู้ตรวจทานโค้ดผ่านทางอีเมล์ (Bacchelli and Bird, 2013) แต่ในปัจจุบัน CodeFlow ได้ถูกนำไปรวมกับ Visual Studio ในรูปแบบ IDE (Tymchuk, et al., 2015)
Collaborator	Smart Bear	Collaborator เป็นเครื่องมือที่มีอยู่บนเว็บไซต์ในรูปแบบที่เป็นส่วนเสริม (Plug-in) หรือ IDE โดย Collaborator เป็นเครื่องมือที่ช่วยให้นักพัฒนาสามารถดูข้อเสนอแนะของแต่ละบุคคลและมีฟังก์ชันในการเลือกติดตามปัญหา (Issue Tracker) เช่น Bugzilla (Rigby and Bird, 2013) ที่สามารถช่วยในการเชื่อมโยงส่วนที่เกิดปัญหาและมีการแสดงข้อเสนอแนะในบรรทัดที่เกิดปัญหาดังกล่าว อีกทั้งยังเป็นเครื่องมือที่สามารถส่งออกเอกสารเกี่ยวกับข้อเสนอแนะในรูปแบบ Microsoft Word และ PDF
Crucible	Atlassian	Crucible เป็นเครื่องมือที่ให้บริการการตรวจทานโค้ดในรูปแบบเว็บแอปพลิเคชัน โดยการใช้งาน Crucible ผู้ใช้สามารถเลือกแสดงข้อเสนอแนะในแต่ละบรรทัดของซอร์สโค้ด และสามารถเลือกกลุ่มของผู้ตรวจทานโค้ด โดยการระบุบุคคลที่ต้องการให้ตรวจทานหรือใช้วิธีการสุ่มผู้ตรวจทานโค้ด ในปัจจุบัน Crucible ได้ถูกนำไปรวมกับเครื่องมือติดตามข้อบกพร่องที่ชื่อว่า Jira Bug Tracker และเครื่องมืออื่น ๆ ของบริษัท Atlassian (Tymchuk, et al., 2015)

ตารางที่ 2.4 รายละเอียดของเครื่องมือที่ใช้ในการตรวจทานโค้ด (ต่อ)

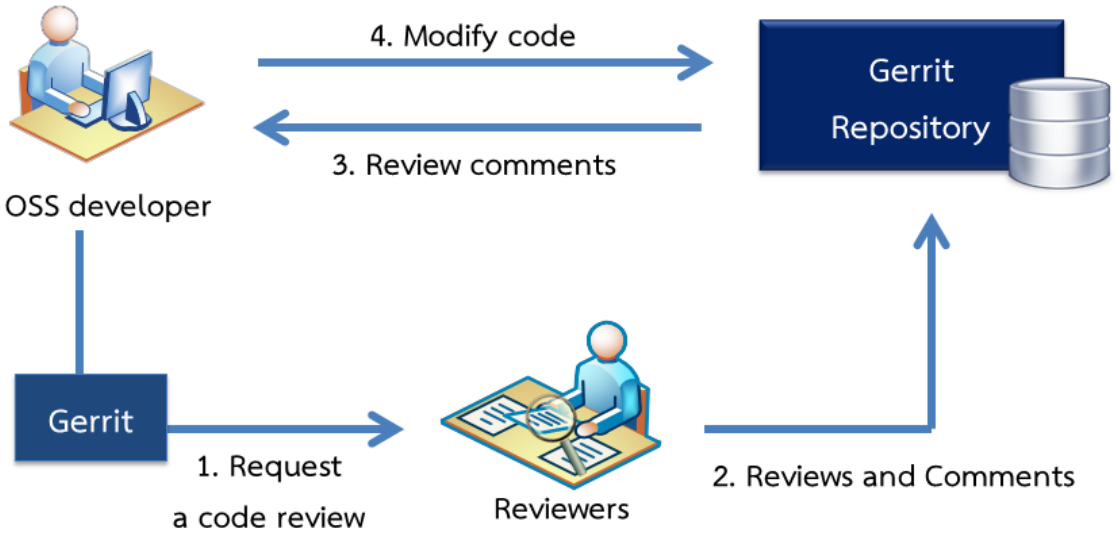
ชื่อเครื่องมือ	พัฒนาโดย	รายละเอียด
Differential	Phacility Inc.	Differential เป็นเครื่องมือที่เอื้ออำนวยให้ผู้ตรวจทานโค้ดสามารถดูรายละเอียดของข้อเสนอแนะผ่านทางหน้าเว็บเพจที่เป็นรายงานการตรวจทานโค้ดภายในระบบ และสามารถดูข้อเสนอแนะในแต่ละบรรทัดของซอร์สโค้ดได้ ในปัจจุบัน Differential เป็นโครงการโอเพนซอร์สที่ได้รับสัญญาอนุญาตโดย Phacility Inc. (Tymchuk, et al., 2015) โดยปกติแล้วเครื่องมือ Differential เป็นส่วนหนึ่งของซอฟต์แวร์ที่มีชื่อว่า Phabricator ซึ่งเป็นเว็บแอปพลิเคชันที่รวมชุดของการใช้งานเครื่องมือต่าง ๆ ที่ถูกพัฒนาโดย Phacility Inc. ที่ใช้ในการอำนวยความสะดวกในเรื่องการพัฒนาซอฟต์แวร์และการตรวจทานโค้ดร่วมกัน นอกจากนี้ Phacility Inc. ยังมีซอฟต์แวร์ตัวใหม่ที่เรียกว่า Pholio ได้ถูกนำมาใช้ในการตรวจสอบรูปภาพและแบบจำลอง (Mocks) ที่เป็นการตรวจทานการออกแบบ (Design Review) (อ้างอิงจากเว็บ https://www.phacility.com/phabricator/pholio/, 7 พฤศจิกายน 2559) โดยสมาชิกในทีมสามารถให้ข้อเสนอแนะบนภาพได้เช่นเดียวกับการตรวจทานซอร์สโค้ด แต่จะแตกต่างกันตรงที่ข้อเสนอแนะดังกล่าวจะปรากฏอยู่บนภาพ
Review Board	MIT	Review Board เป็นเครื่องมือที่ให้บริการในรูปแบบเว็บไซต์ที่อนุญาตให้นักพัฒนาสามารถเข้าไปดูข้อเสนอแนะได้ โดยการคลิกที่หมายเลขบรรทัดเพื่อเปิดหน้าต่างใหม่ (Popup Window) ที่มีการแสดงข้อเสนอแนะดังกล่าว อีกทั้งเครื่องมือนี้ยังมี Review Bot เป็นส่วนเสริมสำหรับการตรวจทานโค้ดที่จะช่วยตรวจทานโค้ดแบบอัตโนมัติและให้ข้อเสนอแนะในการปรับปรุงคุณภาพของซอร์สโค้ดอีกทั้งยังช่วยในการคัดเลือกกลุ่มของผู้ตรวจทานโค้ดโดยวิเคราะห์จากประวัติการแก้ไขซอร์สโค้ดและการให้ข้อเสนอแนะแก่นักพัฒนาที่ทำงานในโมดูลหรือส่วนประกอบเดียวกัน (Balachandran, 2013)

ตารางที่ 2.4 รายละเอียดของเครื่องมือที่ใช้ในการตรวจทานโค้ด (ต่อ)

ชื่อเครื่องมือ	พัฒนาโดย	รายละเอียด
Gerrit	Apache	Gerrit เป็นซอฟต์แวร์โอเพนซอร์สที่ถูกพัฒนาขึ้นเพื่อโครงการ Android (Tymchuk, et al., 2015) แต่ในปัจจุบันเกอริตเป็นระบบที่อำนวยความสะดวกในการตรวจทานโค้ดสำหรับโครงการโอเพนซอร์สที่ถูกจัดเก็บในอยู่แหล่งเก็บซอร์สโค้ดของระบบกิตในการตรวจทานโค้ด ระบบเกอริตจะให้บริการผ่านทางเว็บอินเตอร์เฟซ (Gerrit Web Interface) (Rigby and Bird, 2013) โดยนักพัฒนาจะทำการส่งคำร้องขอการตรวจทานไปยังนักพัฒนาคนอื่น ๆ ที่ไม่ใช่ผู้เขียนโค้ดให้ทำการตรวจทานและให้ข้อเสนอแนะเกี่ยวกับการปรับปรุงซอร์สโค้ดผ่านทางกล่องข้อความบนหน้าเว็บ อีกทั้งระบบเกอริตยังเป็นระบบที่ให้ผู้ใช้งานทั่วไปหรือนักวิจัยสามารถดาวน์โหลดข้อมูลเกี่ยวกับการตรวจทานโค้ดภายในระบบเกอริตได้ (Bosu, 2013)
Github pull request	MIT	GitHub เป็นพื้นที่ให้บริการออนไลน์สำหรับการจัดเก็บซอร์สโค้ด (Git Repository) แต่ภายใน GitHub จะมีฟังก์ชันเกี่ยวกับการตรวจทานโค้ดที่คล้ายกับระบบ Gerrit ที่เรียกว่า Pull-based Code Review ที่สามารถส่งคำร้องขอการตรวจทาน (Pull Request) ให้ผู้อื่นช่วยตรวจทานโค้ดได้
Upsource	JetBrains	Upsource เป็นส่วนหนึ่งของเครื่องมือที่ช่วยในการพัฒนาโปรแกรม เช่น IntelliJ IDEA, TeamCity, YouTrack (Tymchuk, et al., 2015) โดยการใช้งาน Upsource ผู้ใช้สามารถเลือกบรรทัดของซอร์สโค้ดที่ต้องการดูข้อเสนอแนะเช่นเดียวกับเครื่องมือการตรวจทานโค้ดตัวอื่น ๆ แต่ข้อมูลเกี่ยวกับข้อเสนอแนะจะถูกจัดเก็บและแสดงบนเว็บเบราว์เซอร์

จากการศึกษาเกี่ยวกับเครื่องมือในการตรวจทานโค้ดข้างต้น ผู้วิจัยได้เลือกศึกษาข้อมูลที่ได้มาจากเครื่องมือในการตรวจทานโค้ดที่มีชื่อว่าเกอริต ซึ่งเป็นระบบซอฟต์แวร์โอเพนซอร์สที่ให้บริการในการตรวจทานโค้ดที่สามารถใช้งานได้อย่างอิสระและไม่เสียค่าใช้จ่าย โดยระบบเกอริตเป็นคลังเก็บข้อมูลเกี่ยวกับข้อเสนอแนะของโครงการซอฟต์แวร์โอเพนซอร์สมากกว่า 2,000 โครงการ และเป็นระบบที่มีเว็บเซอร์วิสที่อำนวยความสะดวกแก่ผู้วิจัยในการนำข้อมูลไปทำการศึกษาและวิเคราะห์ข้อมูลเกี่ยวกับ

ข้อเสนอแนะต่าง ๆ สำหรับกระบวนการตรวจทานโค้ดในโครงการโอเพนซอร์สแสดงดังรูปที่ 2.1



รูปที่ 2.1 กระบวนการตรวจทานโค้ดในโครงการโอเพนซอร์ส

รูปที่ 2.1 เป็นการแสดงกระบวนการตรวจทานโค้ดภายในโครงการโอเพนซอร์สของระบบเกอริต ซึ่งกระบวนการนี้จะเริ่มขึ้นก็ต่อเมื่อนักพัฒนาซอฟต์แวร์ต้องการที่จะตรวจสอบซอร์สโค้ดที่ได้พัฒนาขึ้น โดยมีขั้นตอนดังนี้

1) นักพัฒนาซอฟต์แวร์จะทำการส่งคำร้องขอไปยังผู้ตรวจทานโค้ด เพื่อขอให้ผู้ตรวจทานโค้ดทำการตรวจสอบซอร์สโค้ดที่ได้ทำการพัฒนาหรือซอร์สโค้ดที่ได้ทำการแก้ไข

2) เมื่อผู้ตรวจทานโค้ดยอมรับคำร้องขอแล้ว ผู้ตรวจทานโค้ดก็จะเริ่มทำการตรวจทานซอร์สโค้ด โดยค้นหาข้อบกพร่องหรือข้อผิดพลาดที่อาจจะเกิดขึ้น

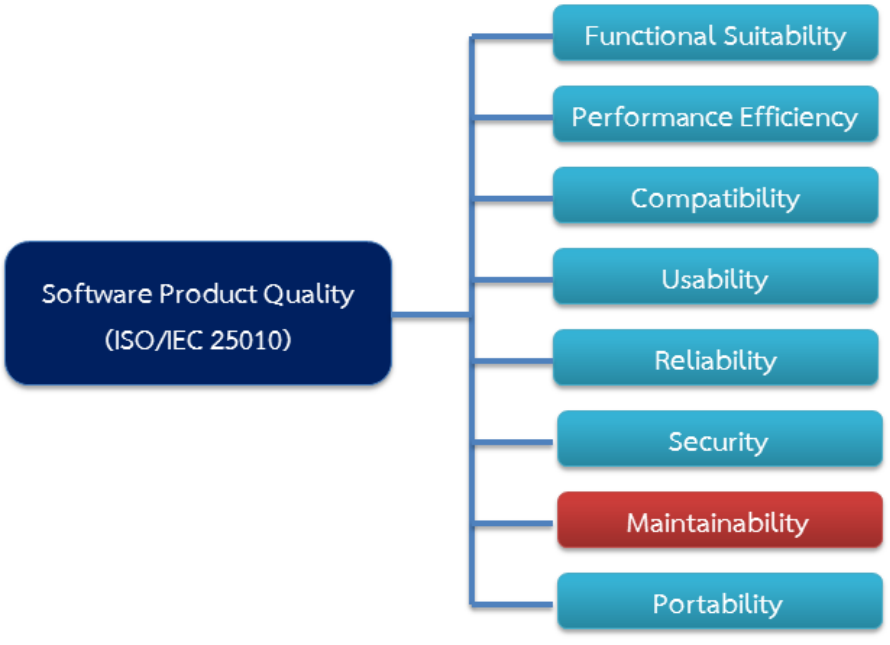
3) หากผู้ตรวจทานตรวจพบข้อบกพร่องในซอร์สโค้ดก็จะทำการเพิ่มข้อเสนอแนะเกี่ยวกับการแก้ไขปรับปรุงซอร์สโค้ดในบรรทัดที่ค้นพบข้อบกพร่องกลับไปยังนักพัฒนา เพื่อให้นักพัฒนาทำการแก้ไขปรับปรุงซอร์สโค้ดตามข้อเสนอแนะที่ได้รับ

4) เมื่อนักพัฒนาซอฟต์แวร์ได้รับข้อเสนอแนะจากผู้ตรวจทานโค้ดแล้ว นักพัฒนาก็จะทำการพิจารณาข้อเสนอแนะดังกล่าวว่าจะดำเนินการแก้ไขหรือไม่ หากนักพัฒนาได้ทำการแก้ไขโค้ดตามข้อเสนอแนะที่ได้รับก็จะกลับเข้าสู่ขั้นตอนที่ 1) อีกครั้ง

ในงานวิจัยนี้ ผู้วิจัยจะทำการศึกษาข้อเสนอแนะของผู้ตรวจทานโค้ดในขั้นตอนที่ 3) และการแก้ไขโค้ดตามคำแนะนำของผู้ตรวจทานโค้ดในขั้นตอนที่ 4) เพื่อหาคำตอบของคำถามวิจัย คือ *“นักพัฒนาซอฟต์แวร์ให้ความสนใจต่อการบำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์สหรือไม่”*

2.1.4 คุณภาพของซอฟต์แวร์และความสามารถในการบำรุงรักษาซอฟต์แวร์

คุณภาพของซอฟต์แวร์ (Software Quality) หมายถึง ระดับที่ผลิตภัณฑ์ซอฟต์แวร์สามารถตอบสนองความต้องการหรือความพึงพอใจของผู้ใช้งานได้ โดยจะต้องมีคุณลักษณะหรือฟังก์ชันการทำงานครบถ้วนตามเงื่อนไขที่ได้ระบุไว้ในเอกสารความต้องการเชิงซอฟต์แวร์ (Seref and Tanriover, 2016) ซึ่งคุณลักษณะของซอฟต์แวร์ (Quality Attributes) เป็นสิ่งที่ใช้กำหนดระดับคุณภาพของซอฟต์แวร์ตามมาตรฐาน ISO/IEC 25010 ซึ่งสามารถแบ่งเป็น 8 คุณลักษณะย่อย ดังรูปที่ 2.2 โดยแต่ละคุณลักษณะของซอฟต์แวร์มักจะขึ้นอยู่กับความต้องการตามลักษณะการทำงานของซอฟต์แวร์หรือแอปพลิเคชัน (Software Domain) ประเภทของผลิตภัณฑ์ซอฟต์แวร์ รวมถึงวัตถุประสงค์ในการใช้งานซอฟต์แวร์ (อ้างอิงจากเว็บ http://asq.org/learn-about-quality/software-quality/overview/overview.html, 20 กันยายน 2559)



รูปที่ 2.2 รูปแบบคุณภาพของผลิตภัณฑ์ซอฟต์แวร์ที่กำหนดไว้ใน ISO/IEC 25010

มาตรฐาน ISO/IEC 25010 เป็นมาตรฐานสากลที่องค์กรธุรกิจทั่วโลกให้ความสำคัญ เพราะเป็นมาตรฐานทางด้านคุณภาพของผลิตภัณฑ์ซอฟต์แวร์ โดยจะมุ่งเน้นทางด้านระบบและความต้องการเชิงคุณภาพของซอฟต์แวร์ (Software Quaility Requirements หรือ SQuaRE) รวมถึงการประเมินผลคุณภาพของซอฟต์แวร์ โดยสาเหตุที่ทางผู้วิจัยได้เลือกศึกษา ISO/IEC 25010 เพราะเป็นมาตรฐานที่มีคุณลักษณะทางด้านคุณภาพที่เกี่ยวข้องกับระบบหรือซอฟต์แวร์ และผลกระทบเชิงซอฟต์แวร์ที่เกี่ยวข้องกับผู้มีส่วนได้ส่วนเสียหรือผู้ใช้งานระบบ ซึ่งมาตรฐานสากลนี้มีคุณลักษณะทางด้าน

คุณภาพที่สำคัญ คือ การกำหนดความต้องการ การวัดคุณภาพ และการประเมินผลที่ได้รับการยอมรับกันอย่างแพร่หลายในแง่ของมาตราวัดและวิธีการวัดคุณภาพของซอฟต์แวร์ (Iso Iec, 2011)

งานวิจัยนี้ผู้วิจัยสนใจที่จะทำการศึกษาเกี่ยวกับความสามารถในการบำรุงรักษาซอฟต์แวร์ (Software Maintainability) เนื่องจากความสามารถในการบำรุงรักษาซอฟต์แวร์เป็นหนึ่งในคุณลักษณะที่มีความสำคัญต่อการเพิ่มคุณภาพของซอฟต์แวร์ และช่วยลดค่าใช้จ่ายในการบำรุงรักษาซอฟต์แวร์ (Seref and Tanriover, 2016) โดย ISO/IEC 25010 ได้นิยามคุณลักษณะ “ความสามารถในการบำรุงรักษาซอฟต์แวร์” ไว้ว่าเป็น “คุณลักษณะที่มีความสามารถในการแก้ไขเพื่อปรับปรุงโค้ดให้ดีขึ้น การปรับเปลี่ยนหรือเปลี่ยนแปลงในสภาพแวดล้อมและตามความต้องการของซอฟต์แวร์ที่ได้กำหนดไว้” ซึ่งจากการศึกษาเกี่ยวกับความสามารถในการบำรุงรักษาซอฟต์แวร์พบว่า Ghosh และคณะ (2011) ได้มีการระบุคุณลักษณะย่อยที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ไว้ทั้งหมด 40 คุณลักษณะ โดยคำนิยามของแต่ละคุณลักษณะย่อยมีดังนี้

1) ความถูกต้อง (Accuracy) เป็นความสามารถของผลิตภัณฑ์ซอฟต์แวร์ เพื่อให้ได้ผลลัพธ์ที่ถูกต้องหรือเป็นที่ยอมรับด้วยระดับของความแม่นยำ

2) ความสามารถในการดัดแปลง (Adaptability) เป็นความสามารถของผลิตภัณฑ์ซอฟต์แวร์ที่จะปรับตัวให้เข้ากับสภาพแวดล้อมที่ต่างกัน โดยไม่ต้องดำเนินการหรือใช้วิธีอื่น ๆ ในการแก้ไข

3) ความสามารถในการวิเคราะห์ (Analyzability) เป็นความสามารถของผลิตภัณฑ์ซอฟต์แวร์ที่จะประเมินผลกระทบของซอฟต์แวร์ เมื่อทำการเปลี่ยนแปลงการทำงานของระบบส่วนต่าง ๆ รวมถึงการวิเคราะห์เพื่อหาข้อบกพร่องหรือสาเหตุของความล้มเหลว พร้อมทั้งระบุส่วนที่จะต้องดำเนินการแก้ไข

4) ความสามารถในการขยาย (Augmentability) เป็นความสามารถของโมเดลหรือแบบจำลองของผลิตภัณฑ์ซอฟต์แวร์ เพื่อรองรับการขยายตัวของข้อมูลที่จะถูกจัดเก็บหรือการเพิ่มองค์ประกอบที่ฟังก์ชันต้องการ

5) ความพร้อมในการใช้งาน (Availability) เป็นระดับการทำงานของระบบที่สามารถดำเนินการได้และสามารถใช้เป็นผลิตภัณฑ์หรือใช้งานได้

6) ความสามารถในการเปลี่ยนแปลง (Changeability) เป็นระดับความยากง่ายในการแก้ไขหรือปรับเปลี่ยน

7) ความครบสมบูรณ์ (Completeness) เป็นระดับการทำงานแบบเต็มรูปแบบหรือการติดตั้งระบบที่สามารถเรียกฟังก์ชันการทำงานได้อย่างประสบความสำเร็จ

8) ความซับซ้อน (Complexity) เป็นระดับที่ส่วนประกอบหรือระบบมีการออกแบบและมีโครงสร้างภายในที่สามารถทำความเข้าใจ บำรุงรักษารักษา และตรวจสอบได้ง่าย

9) ความสามารถในการทำความเข้าใจ (Comprehensibility) เป็นระดับความเข้าใจได้ง่ายของระบบหรือส่วนหนึ่งส่วนใดของระบบ

10) การปฏิบัติตามข้อกำหนดทางด้านการบำรุงรักษา (Maintainability Compliance) เป็นการทำให้ผลิตภัณฑ์ซอฟต์แวร์เป็นไปตามมาตรฐาน อนุสัญญาหรือข้อบังคับในกฎหมายและข้อกำหนดเกี่ยวกับการบำรุงรักษา

11) ความกระชับรัดกุม (Conciseness) เป็นความกระชับของโปรแกรมในแง่ของบรรทัดซอร์สโค้ด

12) ความคงเส้นคงวา (Consistency) เป็นการใช้การออกแบบซอฟต์แวร์ในรูปแบบเดียวกันและใช้เทคนิคการจัดทำเอกสารตลอดการพัฒนาซอฟต์แวร์ภายในโครงการ

13) ความสามารถในการแก้ไขให้ถูกต้อง (Correctability) เป็นความสามารถในการแก้ไขข้อบกพร่องเพียงเล็กน้อยที่ผู้ใช้ค้นพบระหว่างการใช้งานแอปพลิเคชันหรือส่วนประกอบ

14) การส่งมอบ (Deliverable) เป็นผลิตภัณฑ์ใด ๆ (งาน) ที่ต้องจัดส่งให้กับบุคคลอื่นนอกจากผู้พัฒนาผลิตภัณฑ์

15) เอกสาร (Documentation) เป็นการทดสอบคุณภาพของเอกสาร เช่น คู่มือผู้ใช้งานหรือคู่มือการติดตั้งโปรแกรม

16) ความทนทาน (Durability) เป็นมาตรการหรือตัวชี้วัดชีวิตของผลิตภัณฑ์ โดยการวัดผลิตภัณฑ์ทั้งหมดเพื่อตรวจสอบความเสื่อมสภาพที่เป็นไปตามเวลาหรือการใช้งาน

17) มีประสิทธิภาพ (Efficiency) เป็นความสามารถของผลิตภัณฑ์ซอฟต์แวร์ เพื่อให้เกิดประสิทธิภาพที่เหมาะสมเมื่อเทียบกับปริมาณของทรัพยากรที่ถูกใช้

18) ความพยายาม (Effort) เป็นความพยายามที่จะเปลี่ยนแปลงหรือแก้ไขผลิตภัณฑ์ซอฟต์แวร์เพื่อปรับให้เข้ากับสภาพแวดล้อมอื่น ๆ หรือโปรแกรมประยุกต์อื่นที่แตกต่างจากการออกแบบที่กำหนดไว้

19) ความสามารถในการขยายตัว (Expandability) เป็นระดับของการออกแบบสถาปัตยกรรม ข้อมูลหรือขั้นตอนการออกแบบที่สามารถเพิ่มขยายได้

20) ความสามารถในการยืดขยาย (Extensibility) เป็นความง่ายของแอปพลิเคชันหรือส่วนประกอบที่สามารถปรับปรุงได้ในอนาคตเพื่อตอบสนองความต้องการหรือเป้าหมายที่เปลี่ยนแปลงไป

21) ความสามารถในการยืดหยุ่น (Flexibility) เป็นความพยายามในการแก้ไขส่วนที่จำเป็นของโปรแกรมการดำเนินงาน

22) การวิเคราะห์ผลกระทบ (Impact Analysis) เป็นกิจกรรมในการระบุว่าจะแก้ไขส่วนใดเพื่อให้บรรลุเป้าหมายของการเปลี่ยนแปลง เช่น การประมาณผลที่อาจเป็นไปได้ในการดำเนินการเปลี่ยนแปลง

23) การดำเนินงาน (Implementation) หมายถึง การพัฒนาโครงสร้างที่เป็นระบบเพื่อรวมการบริการหรือส่วนประกอบของซอฟต์แวร์เข้าด้วยกันได้อย่างมีประสิทธิภาพ ก่อนที่จะเข้าสู่กระบวนการทำงานของโครงสร้างองค์กรหรือผู้ใช้งานแต่ละคน

24) เครื่องมือ (Instrumentation) เป็นระดับที่โปรแกรมสำหรับใช้ในการตรวจสอบสามารถดำเนินงานและระบุข้อผิดพลาดที่เกิดขึ้นได้

25) ความสามารถในการรวม (Integrability) เป็นความสามารถในการทำให้ส่วนประกอบต่าง ๆ ของระบบสามารถทำงานร่วมกันได้อย่างถูกต้อง

26) การจำกัด (Localization) เป็นกระบวนการปรับซอฟต์แวร์ให้ใช้ภาษาที่เป็นสากลและภาษาสำหรับประเทศต่าง ๆ โดยการเพิ่มส่วนประกอบเกี่ยวกับภาษาและการแปลข้อความ

27) ความสามารถในการแก้ไข (Modifiability) เป็นระดับของซอฟต์แวร์หรือระบบที่สามารถแก้ไขได้อย่างมีประสิทธิภาพ โดยปราศจากข้อบกพร่องหรือส่งผลต่อการลดคุณภาพของซอฟต์แวร์ที่มีอยู่

28) โปรแกรมเชิงโครงสร้างโมดูล (Modularity) เป็นระดับของระบบหรือโปรแกรมคอมพิวเตอร์ที่ประกอบด้วยส่วนประกอบย่อย ๆ หลายส่วน เมื่อมีการเพิ่มหรือแก้ไขส่วนประกอบหนึ่ง ๆ จะต้องทำให้เกิดผลกระทบต่อการเปลี่ยนแปลงส่วนประกอบย่อยอื่น ๆ น้อยที่สุด

29) ความสมบูรณ์ (Perfectness) เป็นการปรับเปลี่ยนผลิตภัณฑ์ซอฟต์แวร์หลังการส่งมอบเพื่อปรับปรุงประสิทธิภาพหรือการบำรุงรักษา

30) การโยกย้าย (Portability) เป็นความสามารถของผลิตภัณฑ์ซอฟต์แวร์ที่สามารถถ่ายโอนจากสภาพแวดล้อมหนึ่งไปยังอีกสภาพแวดล้อมหนึ่งได้

31) ภาษาโปรแกรมมิ่ง (Programming Language) เป็นซอร์สโค้ดของภาษาที่โปรแกรมเมอร์ไม่คุ้นเคยจึงทำให้เกิดความยากลำบากแก่การนำซอร์สโค้ดกลับมาใช้ใหม่

32) ความสามารถในการอ่าน (Readability) เป็นชุดของคุณลักษณะที่เกี่ยวข้องกับความยากลำบากในการทำความเข้าใจส่วนประกอบของซอฟต์แวร์และเอกสารประกอบต่าง ๆ

33) ความสามารถในการนำกลับมาใช้ (Reusability) เป็นระดับของคุณสมบัติที่สามารถนำส่วนประกอบย่อยของระบบเดิมมาใช้ในการสร้างระบบใหม่

34) การอธิบายด้วยตัวเอง (Self-descriptiveness) เป็นการตรวจสอบว่า รูปแบบหรือโมเดลมีข้อมูลที่เพียงพอสำหรับผู้อ่าน หรือตรวจสอบวัตถุประสงค์ สมมติฐาน ข้อจำกัด ส่วนประกอบที่นำเข้า-ส่งออก และสถานะของการแก้ไข

35) ความง่าย (Simplicity) เป็นระดับที่โปรแกรมสามารถทำความเข้าใจได้โดยไม่มีความยากลำบากใด ๆ

36) **ความเสถียร** (Stability) เป็นระดับความเสถียรของซอฟต์แวร์ภายหลังการแก้ไขหรือความสามารถของผลิตภัณฑ์ซอฟต์แวร์ที่สามารถหลีกเลี่ยงผลกระทบจากการแก้ไขปรับปรุงซอฟต์แวร์

37) ความเป็นมาตรฐาน (Standardization) เป็นชุดของมาตรฐานการเขียนโปรแกรมที่สามารถใช้เป็นคำแนะนำสำหรับการพัฒนาซอร์สโค้ด โดยมาตรฐานการเขียนโปรแกรม แนวทางและวิธีการปฏิบัติที่ใช้ในการเขียนโปรแกรมจะช่วยให้สามารถอ่านโค้ด ทำความเข้าใจได้ง่าย และมีผลโดยตรงต่อการเปลี่ยนแปลง รวมถึงการบำรุงรักษา

38) ความสามารถในการทดสอบ (Testability) เป็นระดับของประสิทธิภาพและประสิทธิผลของซอฟต์แวร์ และองค์ประกอบภายในซอฟต์แวร์ที่จะถูกทดสอบ

39) การติดตาม (Traceability) เป็นความสามารถในการติดตามการออกแบบหรือส่วนประกอบของโปรแกรมที่เกิดขึ้นจริงตามความต้องการที่กำหนดไว้

40) ความสามารถในการทำความเข้าใจ (Understandability) เป็นความสามารถของผลิตภัณฑ์ซอฟต์แวร์เพื่อให้ผู้ใช้สามารถเข้าใจได้ว่า ซอฟต์แวร์นั้นเหมาะสมกับความต้องการที่จะนำไปใช้งานหรือไม่ และมีวิธีการใช้งาน รวมถึงเงื่อนไขการใช้งานอย่างไร

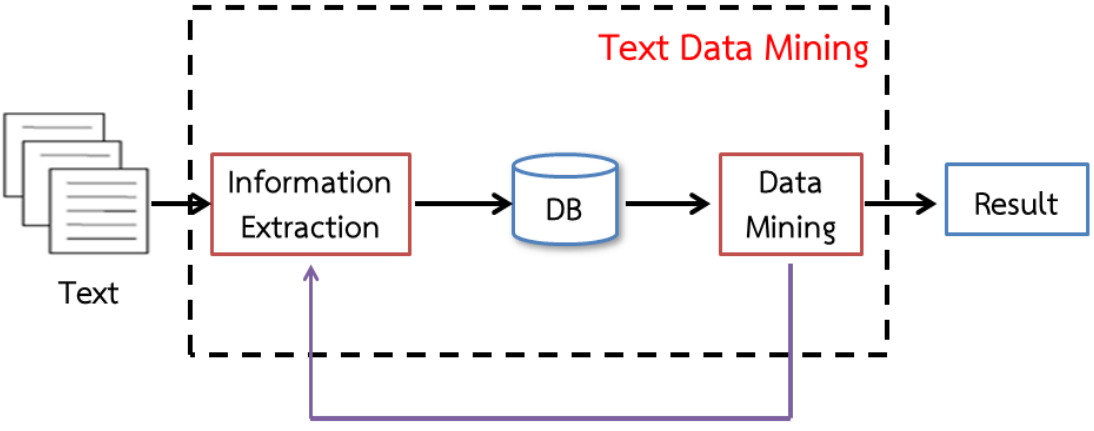
สำหรับคุณลักษณะย่อยของความสามารถในการบำรุงรักษาซอฟต์แวร์ทั้ง 40 คุณลักษณะ ที่กล่าวไว้ข้างต้นนั้น ผู้วิจัยจะทำการพิจารณาคุณลักษณะดังกล่าวเพื่อนำไปใช้เป็นคำหลักในการดำเนินงานวิจัยภายใต้วิธีการทำเหมืองข้อความ โดยรายละเอียดของขั้นตอนการดำเนินงานจะกล่าวถึงในบทที่ 3

2.1.5 การทำเหมืองข้อความ

เพื่อที่จะแก้ปัญหาจากข้อมูลสารสนเทศจำนวนมาก (Information Overload) นักวิจัยในปัจจุบันได้นำวิธีการทางด้านการวิเคราะห์ข้อมูลวิธีต่าง ๆ อาทิ การค้นคืนสารสนเทศ (Information Retrieval) การจัดการความรู้ (Knowledge Management) และการทำเหมืองข้อมูล (Data Mining) มาประยุกต์ใช้ในกระบวนการรวบรวมเอกสาร การวิเคราะห์ข้อความเอกสาร เพื่อหาคำตอบหรือแก้ปัญหาที่กำลังศึกษา

เหมืองข้อความ (Text Mining) หรืออาจเรียกว่า เหมืองข้อมูลประเภทข้อความ (Text Data Mining) ถือว่าเป็นรูปแบบหนึ่งของการทำเหมืองข้อมูลที่มุ่งเน้นการใช้องค์ความรู้ด้านการประมวลผลภาษาธรรมชาติและหลักการทางสถิติ เพื่อจัดการประเภทเอกสาร (Text Categorization) จัดกลุ่มเอกสาร (Text Clustering) สกัดความรู้ (Extraction) วิเคราะห์ความรู้สึก (Sentiment Analysis) สรุปเอกสาร (Document Summarization) และการหาความสัมพันธ์ (Entity Relation Modeling) โดยรูปที่ 2.3 จะเป็นการแสดงรูปแบบการทำงานของการทำเหมืองข้อความ โดยจะเริ่มจาก

การนำข้อความเข้าสู่กระบวนการสกัดข้อมูลที่ต้องการจากคลังข้อความเอกสาร เพื่อให้ได้ความรู้ใหม่ที่เป็นประโยชน์ สามารถนำมาใช้ในการทำนายเหตุการณ์ หรือจัดกลุ่มของข้อมูลได้อย่างมีประสิทธิภาพ (Baghela and S.P.Tripathi, 2012) นอกจากนี้ยังช่วยให้ผู้ใช้งานทำความเข้าใจข้อมูลได้ง่ายและสามารถใช้ประโยชน์จากเอกสารที่ถูกจัดเก็บไว้ได้อีกด้วย



รูปที่ 2.3 รูปแบบการทำงานของการทำเหมืองข้อความ

เหมืองข้อความเป็นวิธีการที่ใช้สกัดข้อมูลสารสนเทศและการค้นหากฎความสัมพันธ์ของข้อความจำนวนมากจนนำไปสู่การนำเสนอผลลัพธ์ที่ชัดเจน โดยการวิเคราะห์ข้อมูลจากชุดข้อมูลที่ไม่มีโครงสร้างหรือกึ่งโครงสร้าง (Mooney and Nahm, 2003) หรือที่เรียกกันว่า การค้นหาความรู้ในฐานข้อมูลประเภทข้อความ (Knowledge Discovery in Textual Databases : KDT) (Fayyad, et al., 1996) ดังนั้นการประมวลผลของชุดข้อความแบบกึ่งโครงสร้างหรือชุดข้อความที่ไม่มีโครงสร้างข้อมูลเป็นสิ่งที่ทำได้ยาก จึงต้องมีการจัดการข้อความให้มีโครงสร้างที่สามารถนำไปประมวลผลข้อความได้ โดยเทคนิคหรือวิธีที่สามารถดำเนินการมีด้วยกันหลายวิธี ได้แก่

1) การแยกคำศัพท์ (Tokenization) เป็นการแยกคำศัพท์ในข้อความของเอกสารเพื่อให้เกิดความชัดเจนว่า ข้อความประกอบด้วยคำศัพท์ใดบ้าง โดยทั่วไปในภาษาอังกฤษจะมีการแยกคำศัพท์ด้วยการเว้นวรรค และตัดอักขระที่ไม่ใช่ตัวอักษร เช่น สัญลักษณ์ () <> ! ? “ ” ดังนั้นจึงต้องมีการคัดเลือกเฉพาะส่วนที่ต้องการด้วยคำเดี่ยว พยางค์ วลี กลุ่มคำ หรือประโยค มาเป็นตัวแทนของเอกสาร และกำจัดคำหยุด (Stopword) โดยตัดคำที่ไม่มีนัยสำคัญต่อเอกสารและไม่ทำให้ใจความของเอกสารเปลี่ยน เช่น คำสรรพนาม คำเชื่อม เป็นต้น

2) การทำให้คำเป็นแก่นคำ (Stemming) เป็นการจัดกลุ่มคำหลายคำที่มีความหมายคล้ายคลึงกันหรือใกล้เคียงกันให้อยู่ในกลุ่มเดียวกัน โดยลดรูปแบบของคำศัพท์ให้อยู่ในรูปแก่นคำ

(Stemmed Word) โดยคำศัพท์ที่มีแก่นคำเดียวกันจะมีความสัมพันธ์หรือมีความหมายที่คล้ายกัน เช่น “player” “playing” “played” มีแก่นคำ คือ “play” เป็นต้น

3) การแบ่งประโยค วิธีนี้จะเป็นการหาจุดสิ้นสุดของประโยคในข้อความ เพื่อเพิ่มประสิทธิภาพของการประมวลผลขั้นต่อไป โดยทั่วไปอักขระที่สามารถบ่งบอกจุดสิ้นสุดของประโยค ได้แก่ “.”, “?”, “!” เป็นต้น

4) การระบุหน้าที่ของคำ (Part-of-speech Tagging) เป็นส่วนที่ทำให้รู้ว่าข้อมูลของคำศัพท์ที่อยู่ในประโยคนั้นมีหน้าที่เป็นคำนาม (Noun) คำกริยา (Verb) หรือคำสรรพนาม (Pronouns) สำหรับเครื่องมือที่นิยมในการแยกหน้าที่ของคำใช้ ได้แก่ Stanford Log-linear Part-Of-Speech Tagger[19] เป็นต้น

งานวิจัยนี้เป็นการวิเคราะห์ข้อเสนอแนะเกี่ยวกับความสามารถในการบำรุงรักษาซอฟต์แวร์ภายใต้กระบวนการตรวจทานโค้ดของโครงการซอฟต์แวร์โอเพนซอร์ส แต่เนื่องจากข้อเสนอแนะที่อยู่ในรูปแบบประโยคหรือข้อความภาษาอังกฤษที่ได้จากความคิดเห็นของผู้ตรวจทานโค้ดนั้น มักจะมีคำหรือสัญลักษณ์ที่ไม่จำเป็นต่อการวิเคราะห์ข้อความ ดังนั้นผู้วิจัยจึงต้องอาศัยเทคนิคการทำเหมืองข้อความในการทำให้ข้อความเหล่านี้อยู่ในรูปแบบที่สามารถนำไปประมวลผลข้อมูล โดยการใช้วิธีการที่ได้กล่าวข้างต้น เช่น การแยกคำศัพท์ การแปลงคำศัพท์เป็นแก่นคำ และการแบ่งประโยค

2.1.6 อัลกอริทึม Latent Dirichlet Allocation

เนื่องจากปัญหาในการจัดกลุ่มของเอกสารหรือข้อความที่มีจำนวนมากเป็นเรื่องที่ยุ่งยาก ดังนั้น Blei และคณะ (2003) จึงได้คิดค้นวิธีการจัดกลุ่มของเอกสารโดยใช้แนวคิด Latent Dirichlet Allocation (LDA)[20] (Blei, et al., 2003) ซึ่งเป็นอัลกอริทึมที่ใช้ในการสร้างแบบจำลองหัวข้อ (Topic Model) ที่ขึ้นอยู่กับค่าความน่าจะเป็นในการเกิดความถี่ของคำ (Chen, et al., 2016) ที่ปรากฏอยู่ในชุดของเอกสารหรือคลังข้อมูล (Corpus) โดยทั่วไป LDA มีพื้นฐานแนวคิดมาจากการรวมหัวข้อที่มีความหมายเหมือนกัน (Synonym) ให้มาอยู่ในกลุ่มเดียวกัน หรือเป็นการค้นหาคำที่มีคำสะกดแตกต่างกัน เช่น คำว่า “vehicle” ที่หมายถึง พาหนะ เมื่อใช้อัลกอริทึม LDA ในการจัดกลุ่มคำที่เกี่ยวข้องจะได้คำว่า “automobile” และ “car” เป็นต้น

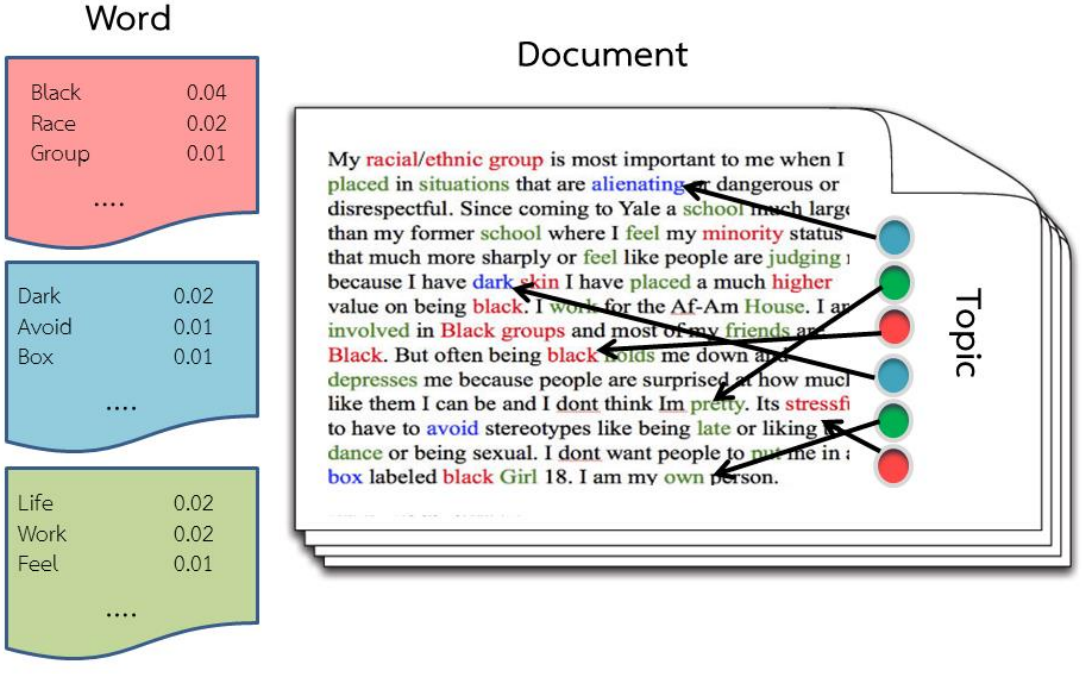
---

[19] http://nlp.stanford.edu/software/tagger.shtml

[20] https://rstudio-pubs-static.s3.amazonaws.com/79360_850b2a69980c4488b1db95987a24867a.html

การใช้อัลกอริทึม LDA จะเป็นประโยชน์สำหรับการจัดกลุ่มคำที่มีความเกี่ยวข้องกับหัวข้อหลักภายในชุดเอกสารที่มีข้อความปรากฏอยู่จำนวนมากและจะช่วยให้สามารถจำแนกหมวดหมู่หรือกลุ่มได้อย่างชัดเจน อีกทั้งยังส่งผลต่อประสิทธิภาพในการค้นคืนเอกสารอีกด้วยซึ่งเครื่องมือหรือซอฟต์แวร์ที่ให้บริการทางด้าน LDA ได้แก่ Gensim[21], R with LDA package[22], GibbsLDA++[23] เป็นต้น

งานวิจัยนี้จะใช้โปรแกรม R ในการนำอัลกอริทึม LDA มาช่วยค้นหาคุณลักษณะย่อยที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาประเภทใหม่ โดยการนำข้อเสนอแนะที่เป็นข้อความภาษาอังกฤษที่มีอยู่จำนวนมากในโครงการมาผ่านกระบวนการ LDA เพื่อค้นหาคำที่มักจะปรากฏอยู่ในข้อเสนอแนะร่วมกับคุณลักษณะของความสามารถในการบำรุงรักษาเดิมที่มีอยู่ทั้ง 40 คุณลักษณะ โดยกระบวนการสร้างแบบจำลองหัวข้อด้วยอัลกอริทึม LDA แสดงดังรูปที่ 2.4



รูปที่ 2.4 การสร้างแบบจำลองหัวข้อด้วย Latent Dirichlet Allocation

[21] https://radimrehurek.com/gensim/models/ldamodel.html
[22] https://cran.r-project.org/web/packages/LDAvis/index.html
[23] http://gibbslda.sourceforge.net

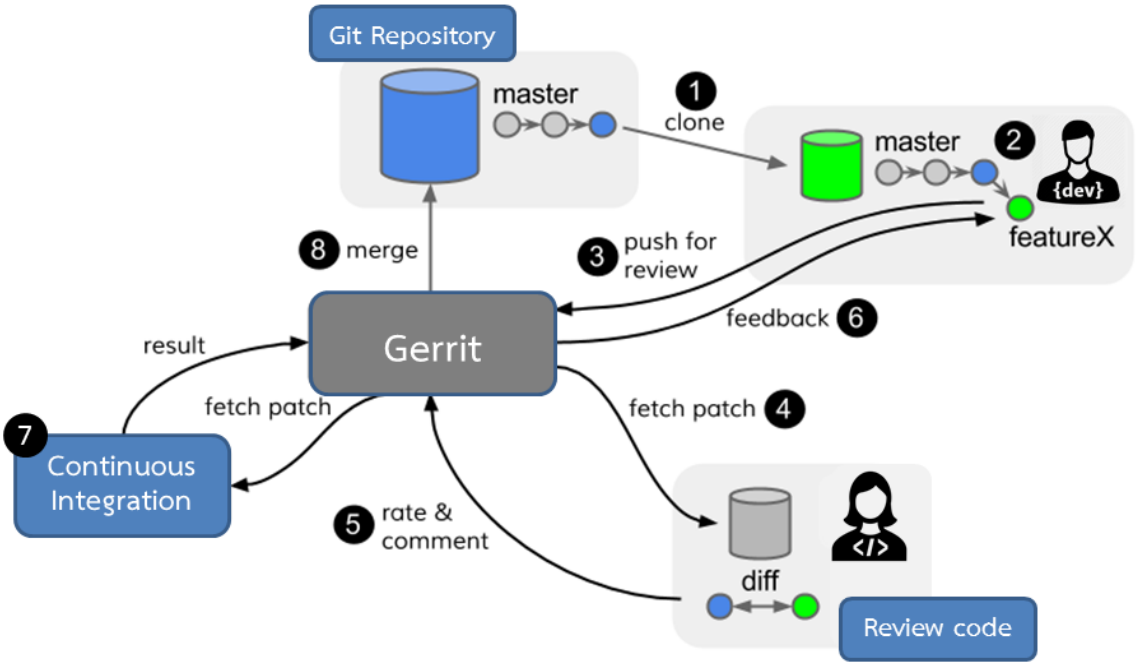
รูปที่ 2.4 เป็นการใช้อัลกอริทึม LDA เพื่อสร้างหัวข้อ โดยเริ่มจากการเรียกใช้ package LDA ในเครื่องมือที่ให้บริการเกี่ยวกับการใช้ LDA และเอกสาร (Document) ที่มีข้อความจำนวนมาก จัดเก็บอยู่เข้าสู่คลังเอกสาร ในการสร้างหัวข้อนั้นอัลกอริทึม LDA จะทำการจัดกลุ่มของหัวข้อหลัก (Topic) ที่มักจะค้นพบในเอกสาร พร้อมทั้งแสดงคำ (Word) ที่มีความเกี่ยวข้องกันหรือมีความหมายที่คล้ายคลึงกันให้มาอยู่ด้วยกัน โดยมีการแสดงค่าความถี่ของคำที่มักจะเกิดขึ้นร่วมกับหัวข้อหลักหรือการเรียงลำดับจากคำที่มีความถี่ในการค้นพบสูงไปยังคำที่มีความถี่ในการค้นพบต่ำ สำหรับผลลัพธ์ที่ได้จากการใช้อัลกอริทึม LDA ผู้วิจัยจะทำการพิจารณาคำที่มีความถี่สูงสุด-ต่ำสุดที่ได้จากแต่ละหัวข้อ เพื่อนำมาเป็นคุณลักษณะของความสามารถในการบำรุงรักษาซอฟต์แวร์ประเภทใหม่

2.2 ซอฟต์แวร์ที่เกี่ยวข้อง

ซอฟต์แวร์ที่เกี่ยวข้องในงานวิจัย ได้แก่ กิต (Git) เกอริต (Gerrit) โปรแกรมอาร์ (R) และเวิร์ดเน็ต (WordNet) ซึ่งเป็นซอฟต์แวร์ที่ผู้วิจัยนำเข้ามาช่วยในการดำเนินงานวิจัยตั้งแต่ขั้นตอนการรวบรวมข้อมูล วิเคราะห์ข้อมูล และประมวลผลข้อมูล

2.2.1 เกอริต (Gerrit) และกิต (Git)

เกอริต เป็นซอฟต์แวร์โอเพนซอร์สที่ถูกพัฒนาขึ้นเพื่ออำนวยความสะดวกในเรื่องการตรวจทานโค้ดให้กับกลุ่มนักพัฒนาซอฟต์แวร์ โดยปกติแล้วเกอริตจะมีการทำงานร่วมกับกิต ซึ่งเป็นระบบที่ให้บริการเกี่ยวกับการจัดการเวอร์ชันของซอฟต์แวร์ (Version Control) โดยจะทำหน้าที่ในการจัดเก็บและควบคุมการเปลี่ยนแปลงที่เกิดขึ้นกับไฟล์ เช่น จัดเก็บประวัติการเปลี่ยนแปลงข้อมูลในไฟล์ การย้อนไฟล์คืนไปยังเวอร์ชันก่อนหน้า การรวมซอร์สโค้ดของทีมนักพัฒนาซอฟต์แวร์ให้เป็นเวอร์ชันล่าสุด (Asundi and Jayant, 2007) โดยรูปที่ 2.5 แสดงภาพรวมการทำงานระหว่างระบบเกอริตและกิต



รูปที่ 2.5 ภาพรวมการทำงานร่วมกันระหว่างระบบเกอริตและระบบจัดการกิต[24]

รูปที่ 2.5 จะเห็นว่าการทำงานของระบบเกอริตและกิต ประกอบด้วย 8 ขั้นตอนดังนี้

1) เมื่อนักพัฒนาซอฟต์แวร์ต้องการมีส่วนร่วมในการพัฒนาซอฟต์แวร์ก็จะทำการสำเนาซอร์สโค้ด (Clone Project) ที่อยู่ในพื้นที่เก็บข้อมูลของระบบกิต

2) นักพัฒนาซอฟต์แวร์นำซอร์สโค้ดที่ได้จากการสำเนามาทำการพัฒนาต่อ โดยการเพิ่มคุณลักษณะหรือคุณสมบัติใหม่ (FeatureX)

3) นักพัฒนาจะทำการอัปโหลดไฟล์ที่ได้ทำการพัฒนาขึ้นระบบเกอริต (Push) โดยที่ซอร์สโค้ดนั้นจะอยู่ในสถานะ “Open” ซึ่งหมายถึง ซอร์สโค้ดที่อยู่ในช่วงที่ผู้ตรวจทานโค้ดทำการตรวจสอบหรือรอการพิจารณาจากทีมนักพัฒนาหลัก เพื่ออนุมัติให้นำไฟล์ที่แก้ไขหรือพัฒนารวมเข้ากับไฟล์ต้นฉบับในระบบกิต

4) ผู้ตรวจทานโค้ดจะทำการดึงไฟล์ข้อมูล (Fetch Patch) ไปยังพื้นที่เก็บข้อมูลในเครื่องคอมพิวเตอร์ เพื่อตรวจสอบการเปลี่ยนแปลงของไฟล์ข้อมูล (Refs Change) ว่ามีอะไรเปลี่ยนแปลงบ้าง โดยที่ “diff” (มาจากคำว่า difference) หรือก็คือความแตกต่างของไฟล์ เช่น ไฟล์ post_item.js คือ

[24] https://eclipsesource.com/blogs/2015/08/03/meet-egerrit/

การเปลี่ยนแปลงของซอร์สโค้ดที่เกิดขึ้นในชุดข้อมูล ซึ่งในกรณีนี้ เมื่อผู้ตรวจทานโค้ดทำการตรวจทานโค้ดในไฟล์ post_item.js แล้ว ระบบเกอริตจะแสดงการเปรียบเทียบระหว่างไฟล์เวอร์ชันเก่า (แทนวงกลมสีน้ำเงิน) และไฟล์เวอร์ชันที่ได้รับการตรวจทานโค้ดขึ้นมาใหม่ (แทนวงกลมสีเขียว) เพื่อให้เห็นถึงความแตกต่างระหว่างไฟล์ก่อนแก้ไขและหลังจากดำเนินการแก้ไขแล้ว

5) ผู้ตรวจทานโค้ดจะทำการตรวจทานซอร์สโค้ด เพื่อค้นหาข้อผิดพลาดหรือข้อบกพร่องที่เกิดขึ้นและส่งคำแนะนำที่จะช่วยในการปรับปรุงแก้ไขซอร์สโค้ดกลับไปยังระบบเกอริต โดยลักษณะของผู้ตรวจทานโค้ดในระบบเกอริตแบ่งออกเป็น 3 แบบ คือ (1) นักพัฒนาซอฟต์แวร์ที่เป็นผู้พัฒนาซอร์สโค้ดส่งคำเชิญให้มาเป็นผู้ตรวจทานโค้ด (2) ทีมนักพัฒนาซอฟต์แวร์แต่งตั้งให้เป็นผู้ตรวจทานโค้ด โดยพิจารณาจากความเชี่ยวชาญและประสบการณ์ และ (3) ผู้ตรวจทานโค้ดส่งอีเมล์ไปยังผู้พัฒนาหรือเจ้าของซอร์สโค้ด เพื่อแจ้งความต้องการที่จะตรวจทานโค้ดชุดนั้น

6) นักพัฒนาซอฟต์แวร์ทำการพิจารณาข้อเสนอแนะที่ได้รับ และทำการแก้ไขซอร์สโค้ดตามข้อเสนอแนะดังกล่าว จากนั้นก็จะกลับเข้าสู่กระบวนการที่ 3) อีกครั้งจนกว่าจะไม่มีการแก้ไขซอร์สโค้ดเพิ่มเติมอีก

7) เมื่อทำการพัฒนาและแก้ไขปรับปรุงโค้ดเสร็จสิ้นแล้ว ทีมนักพัฒนาซอฟต์แวร์จะต้องช่วยกันบูรณาการซอฟต์แวร์ (Continuous Integration) และตรวจสอบสถานะของการตรวจทานโค้ดจากทีมนักพัฒนาหลักในโครงการโอเพนซอร์สว่ามีการอนุมัติให้นำซอร์สโค้ดไปรวมกับแหล่งเก็บโครงการหรือไม่ โดยสถานะในการตรวจทานโค้ดแบ่งออกเป็น 3 ประเภท (Rigby and Bird, 2013; Mukadam, et al., 2013) ดังนี้

(1) Code-Review / Approved เป็นสถานะที่ต้องได้รับการอนุมัติการเปลี่ยนแปลงจากทีมพัฒนาหลักที่มีประสบการณ์และความเชี่ยวชาญที่เหมาะสม สำหรับการตรวจทานโค้ดแต่ละครั้ง ทีมพัฒนาหลักจะต้องทำการให้คะแนนที่แน่นอน (-2, -1, 0, +1, +2) ซึ่งเกณฑ์การให้คะแนนจะขึ้นอยู่กับทีมพัฒนาหลักว่าจะ “ยอมรับ” หรือ “ปฏิเสธ” การตรวจทานซอร์สโค้ดชุดนั้น โดยรายละเอียดการให้คะแนนมีดังนี้

- -2 ซอร์สโค้ดจะไม่ถูกนำมาตรวจสอบ เนื่องจากเกิดข้อผิดพลาดที่ส่งผลให้ระบบไม่สามารถทำงานได้ ด้วยเหตุนี้ชุดของซอร์สโค้ดจะถูกเปลี่ยนสถานะจาก “Open” เป็น “Abandoned” ซึ่งหมายถึง ซอร์สโค้ดชุดนั้นจะถูกละทิ้งหรือเป็นสถานะที่ผู้ตรวจทานโค้ดจะไม่ทำการตรวจทานโค้ดตามคำร้องขอทุกกรณี
- -1 ซอร์สโค้ดจะไม่ถูกนำไปรวมกับโค้ดต้นฉบับของโครงการ เพราะเป็นโค้ดที่มีการทำงานไม่ถูกต้องหรือไม่ตรงตามวัตถุประสงค์การใช้งานของระบบ แต่ผู้ตรวจทานโค้ดก็จะทำการตรวจทานโค้ดและให้คำแนะนำ เพราะซอร์สโค้ดชุดนี้

อาจมีบางส่วนที่สามารถนำไปปรับปรุงแก้ไขเป็นคุณสมบัติใหม่ที่เหมาะสมกับโครงการในอนาคต

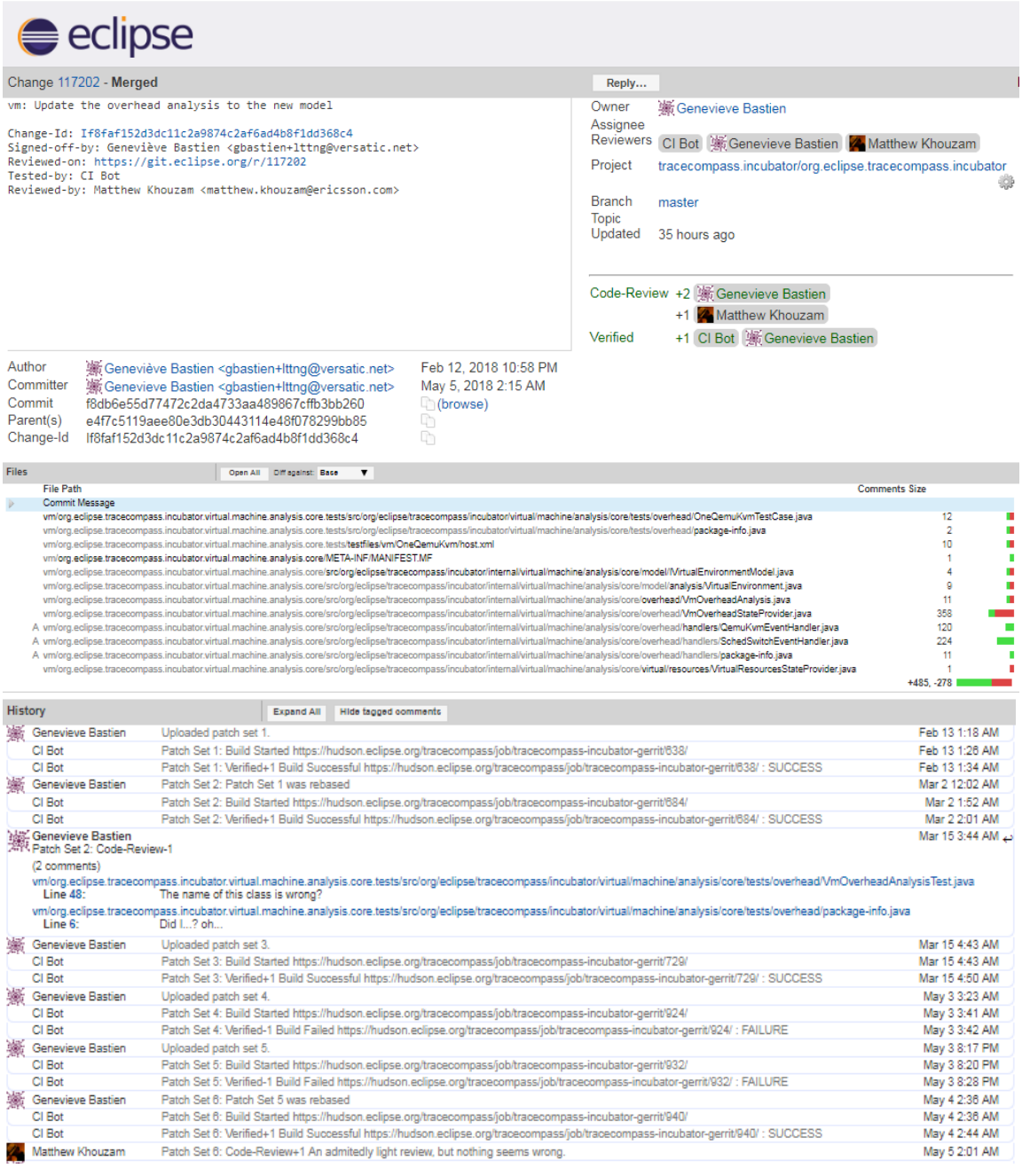
- 0 ไม่มีคะแนน ผู้ตรวจทานโค้ดไม่พยายามที่จะทำการตรวจทานโค้ด
- +1 ซอร์สโค้ดได้รับการอนุมัติจากทีมพัฒนาหลัก แต่ในการอนุมัตินี้จะต้องได้รับความเห็นชอบจากผู้ร่วมพัฒนา (Contributors) ด้วย
- +2 ซอร์สโค้ดได้รับการอนุมัติจากทีมพัฒนาหลักโดยอัตโนมัติ

(2) Verified เป็นสถานะที่ได้รับการยืนยันในการตรวจทานซอร์สโค้ดแล้ว ซึ่งเกณฑ์คะแนนของสถานะการยืนยันมีดังนี้

- -1 ล้มเหลว หมายถึง การตรวจสอบที่ยืนยันแล้วว่า เมื่อนำชุดของซอร์สโค้ดไปรวมกับโค้ดต้นฉบับจะทำให้เกิดข้อผิดพลาดและส่งผลต่อโครงสร้างการทำงานเดิมของระบบ
- +1 หมายถึง ได้รับการยืนยันแล้วว่า ชุดของซอร์สโค้ดสามารถนำไปรวมกับโค้ดต้นฉบับได้ โดยที่ไม่ส่งผลต่อโครงการสร้างการทำงานเดิมของระบบ

(3) Submitted / Merged เมื่อได้รับอนุมัติแล้ว ซอร์สโค้ดชุดนั้นจะถูกนำไปรวมเข้ากับโค้ดต้นฉบับที่จัดเก็บอยู่ในพื้นที่ส่วนกลาง (Shared Centralized Repository) เพื่อให้ทีมนักพัฒนาซอฟต์แวร์สามารถดึงข้อมูลจากไฟล์เป็นเวอร์ชันล่าสุดไปทำการพัฒนาต่อได้

ในส่วนของการแสดงความคิดเห็น เกอริตจะทำหน้าที่ในการเก็บรวบรวมข้อมูลเกี่ยวกับการปรับปรุงโค้ดและแสดงข้อมูลผ่านทางหน้าเว็บไซต์ โดยทั่วไประบบเกอริตจะมีลักษณะคล้ายกับเว็บบอร์ดที่มีพื้นที่ให้สมาชิกในทีมพัฒนาซอฟต์แวร์สามารถสนทนาแลกเปลี่ยนความรู้เกี่ยวกับการพัฒนาและปรับปรุงซอร์สโค้ดได้อย่างอิสระ สำหรับตัวอย่างข้อมูลของโครงการโอเพนซอร์สภายในระบบเกอริตแสดงดังรูปที่ 2.6



รูปที่ 2.6 ตัวอย่างข้อมูลของโครงการ Eclipse ที่จัดเก็บอยู่ในระบบเกอริต[25]

[25] https://git.eclipse.org/r/#/c/117202/

จากรูปที่ 2.6 แสดงให้เห็นถึงการตรวจทานโค้ดในโครงการ Eclipse โดยเริ่มจากเจ้าของซอร์สโค้ด (Genevieve Bastien) ได้ทำการอัปโหลดชุดโปรแกรม (Patch) ที่ต้องการให้ทำการตรวจทานโค้ด ผู้ตรวจทานโค้ดที่ได้รับมอบหมาย (CI Bot และ Matthew Khouzam) ก็จะทำการตรวจทานชุดโปรแกรมที่ผู้พัฒนาได้ทำการร้องขอ โดยผู้ตรวจทานโค้ดสามารถให้คำแนะนำในแต่ละบรรทัดของซอร์สโค้ดที่มีการเปลี่ยนแปลง ซึ่งในที่นี้ผู้ตรวจทานโค้ดคนอื่น ๆ ที่ไม่ได้รับมอบหมายให้ตรวจทานโค้ดก็ยังสามารถแสดงความคิดเห็นได้ สำหรับการอนุมัติแพทซ์ (Matthew Khouzam ให้คะแนน +1 และ Genevieve Bastien ให้คะแนน +2) และชุดของซอร์สโค้ดนี้ได้รับการยืนยัน (Verified) จากทีมพัฒนาหลัก (CI Bot และ Genevieve Bastien) ระบบเกอริตก็จะดำเนินการส่งชุดของซอร์สโค้ดไปยังระบบกิตเพื่อรวมเข้ากับโค้ดของต้นฉบับที่จัดเก็บอยู่ในพื้นที่ส่วนกลาง

8) หลังจากที่ได้รับการตรวจสอบแล้วว่า ซอร์สโค้ดนั้นเหมาะสมที่จะนำไปจัดเก็บเป็นโค้ดฉบับสมบูรณ์ (Code Base) ระบบกิตก็จะทำการรวมซอร์สโค้ด (Merge) ของนักพัฒนาซอฟต์แวร์ให้อยู่ในพื้นที่เก็บข้อมูลเดียวกันภายใต้ระบบกิต พร้อมทั้งทำการตรวจสอบซอฟต์แวร์ในส่วนของการติดตั้งและทดสอบการทำงานของระบบ เพื่อให้ซอร์สโค้ดที่จะทำการเผยแพร่สู่สาธารณะชนเป็นเวอร์ชันล่าสุด (Release Version) ที่สามารถทำงานได้อย่างปกติ

งานวิจัยนี้สามารถนำข้อมูลเกี่ยวกับข้อเสนอแนะที่ได้รับจากผู้ตรวจทานโค้ดมาทำการวิเคราะห์ได้ เนื่องจากระบบเกอริตอนุญาตให้บุคคลทั่วไปสามารถดึงข้อมูลภายใต้กระบวนการตรวจทานโค้ดผ่านทางโพรโทคอล (Hypertext Transfer Protocol : HTTP) โดยข้อมูลที่ได้จะอยู่ในรูปแบบ JSON (JavaScript Object Notation)

2.2.2 โปรแกรมอาร์ (R)

โปรแกรม R เป็นซอฟต์แวร์โอเพนซอร์สที่ผู้ใช้งานสามารถดาวน์โหลดมาใช้ได้ฟรี[26] โดยไม่มีค่าลิขสิทธิ์ และยังเป็นซอฟต์แวร์ที่ได้รับความนิยมในกลุ่มนักวิเคราะห์ข้อมูล เพื่อใช้สำหรับการวิเคราะห์และประมวลผลข้อมูลทางสถิติ รวมถึงยังอำนวยความสะดวกในการทำเหมืองข้อมูลและแสดงผลข้อมูลในรูปแบบกราฟ ซึ่งโปรแกรม R เป็นโปรแกรมที่สามารถทำงานบนระบบปฏิบัติการได้หลาย

[26] https://cran.r-project.org/

แพลตฟอร์ม ได้แก่ ระบบปฏิบัติการยูนิกซ์ (Unix) ระบบปฏิบัติการวินโดวส์ (Windows) ระบบปฏิบัติการแมคโอเอส (Mac OS) และในการใช้งานของโปรแกรม มีส่วนประกอบหลักด้วยกัน 2 ส่วน นั่นคือ

- R Console ที่ใช้สำหรับเขียนคำสั่งพร้อมทั้งแสดงผลลัพธ์
- R Graphic ที่ใช้สำหรับแสดงผลลัพธ์ในรูปแบบกราฟ

จากการสำรวจพบว่า ภาษา R ถูกจัดเป็นอันดับ 8 ของการจัดอันดับภาษาใน TIOBE ในปี พ. ศ. 2017 ซึ่ง TIOBE เป็นบริษัทที่ตรวจสอบและรายงานดัชนีภาษาโปรแกรมมิ่งยอดนิยม (Index of Programming Languages) (อ้างอิงมาจากเว็บ http://blog.revolutionanalytics.com/2018/01/tiobe-2017.html, 20 มีนาคม 2561)

ผู้วิจัยได้นำโปรแกรม R มาใช้ในการทำเหมืองข้อความ เพื่อช่วยในเรื่องการทำความสะอาดข้อมูลและการวิเคราะห์ข้อเสนอแนะที่มีจำนวนมากของโครงการ Eclipse และโครงการ Qt รวมถึงการประมวลผลข้อมูลทางสถิติที่จะใช้ในการสรุปผลของงานวิจัย นอกจากนี้โปรแกรม R ยังมี LDA package ที่ใช้ในการสร้างแบบจำลองหัวข้อ (Topic Model) (อ้างอิงจากเว็บ https://eight2late.wordpress.com/015/09/29/a-gentle-introduction-to-topic-modeling-using-r/, 22 ธันวาคม 2559)

2.2.3 โปรแกรมเวิร์ดเน็ต (WordNet)

โปรแกรม WordNet เป็นซอฟต์แวร์ที่มีฐานข้อมูลคำศัพท์ (Lexical Database) ของภาษาอังกฤษ ซึ่งกลุ่มคำภาษาอังกฤษที่เป็นชุดของคำพ้องความหมาย (Synonyms) ถูกเรียกว่า “synset” ที่มีการเชื่อมโยงกันด้วยความสัมพันธ์ของแนวคิดและความหมายคำศัพท์ โดยโปรแกรม WordNet ถูกพัฒนาโดยมีวัตถุประสงค์เพื่อผสมผสานระหว่างพจนานุกรมคำศัพท์ (Dictionary) และพจนานุกรมคำพ้องและคำที่มีความหมายตรงข้าม (Thesaurus) เข้าด้วยกัน สำหรับการใช้งานโปรแกรม WordNet แบ่งออกเป็น 3 แบบ คือ 1) แบบออนไลน์ 2) โปรแกรมสำเร็จรูป และ 3) รูปแบบของโปรแกรมเสริมระบบ (Plug-in) โดยแบบออนไลน์นั้นสามารถใช้งานได้ผ่านเครือข่ายอินเตอร์เน็ต ส่วนแบบโปรแกรมสำเร็จรูป ผู้ใช้งานสามารถดาวน์โหลดได้ฟรี[27] และใช้งานผ่านเครื่องคอมพิวเตอร์ที่มีระบบปฏิบัติการยูนิกซ์ (Unix) ระบบปฏิบัติการวินโดวส์ (Windows) และระบบปฏิบัติการแมคโอเอส (Mac OS) และสำหรับการใช้งานรูปแบบของโปรแกรมเสริม ผู้ใช้งานสามารถทำการเพิ่มโปรแกรมเสริมไปยังโปรแกรมอื่น ๆ ที่รองรับ เช่น โปรแกรมอาร์ (R) โปรแกรมไพธอน (Python)

[27] https://wordnet.princeton.edu/wordnet/download

งานวิจัยนี้จะนำโปรแกรม WordNet มาช่วยในการค้นหาคำที่มีความหมายสื่อถึงคำหลัก (คุณลักษณะของความสามารถในการบำรุงรักษาซอฟต์แวร์) ตัวอย่างเช่น ผู้วิจัยต้องการค้นหาข้อเสนอแนะที่มีการให้คำแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ประเภท Modifiability เมื่อใช้โปรแกรม WordNet ให้การค้นหาที่มีความหมายสื่อถึงคำว่า “Modify” ก็จะแสดงคำว่า “edit”, “improve”, “solve” ซึ่งสาเหตุที่ผู้วิจัยทำการรวบรวมกลุ่มคำที่มีความหมายเหมือนกับคำหลัก เนื่องจากในการประมวลผลข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ หากผู้วิจัยใช้เพียงคำหลัก “Modify” ผลลัพธ์ที่ได้จากการประมวลผลจะได้เพียงข้อเสนอแนะที่มีการระบุคำว่า “Modify” เท่านั้น แต่เมื่อผู้วิจัยทำการค้นหาคำอื่น ๆ ที่เกี่ยวข้องกับ “Modify” และนำไปใช้ประมวลผลก็จะทำให้ได้ผลลัพธ์ที่ครอบคลุมคำนิยามของคุณลักษณะการบำรุงรักษาซอฟต์แวร์มากยิ่งขึ้น

2.3 วรรณกรรมที่เกี่ยวข้อง

ในบทนี้จะเป็นการอภิปรายวรรณกรรมที่เกี่ยวข้อง โดยแบ่งออกเป็น 3 กลุ่ม ได้แก่ 1) คุณภาพของซอฟต์แวร์โอเพนซอร์ส 2) ข้อเสนอแนะในกระบวนการตรวจทานโค้ดและเครื่องมือที่ใช้ในการตรวจทานโค้ด และ 3) การบำรุงรักษาซอฟต์แวร์ โดยมีรายละเอียดดังต่อไปนี้

2.3.1 คุณภาพของซอฟต์แวร์โอเพนซอร์ส

งานวิจัยของ Bakar และ Arsat ได้ทำการศึกษาปัจจัยที่มีผลต่อคุณภาพของซอฟต์แวร์โอเพนซอร์ส โดยการวัดคุณภาพจากซอร์สโค้ด การทดสอบ และแบบจำลองในการออกแบบ โดยใช้ McCall’s Quality Factor Model ซึ่งเป็นโมเดลที่ใช้ในการวัดคุณภาพของซอฟต์แวร์ ได้แก่ การบำรุงรักษา ความถูกต้อง ความน่าเชื่อถือ ประสิทธิภาพ และใช้งานได้ง่าย (Bakar and Arsat, 2014) ซึ่งงานวิจัยนี้จะใช้ 6 object-oriented metrics ที่คิดค้นโดย Chidamber and Kemerer (CK) เป็นตัวชี้วัดคุณภาพของซอฟต์แวร์ ประกอบด้วย

1) Weighted Methods per Class (WMC)
2) Coupling between Object Classes (CBO)
3) Depth of Inheritance Tree (DIT)
4) Response for Class (RFC)
5) Number of Children (NOC)
6) Lack of Cohesion Metric (LCOM)

จากผลการศึกษาจะแสดงให้เห็นว่า ความสัมพันธ์และความซับซ้อนของซอร์สโค้ดส่งผลต่อขนาดของคลาสหรือไม่ โดยจากการวิเคราะห์ค่าสหสัมพันธ์ (Correlation Analysis) ผลลัพธ์ที่ได้พบว่า การทำนายความสัมพันธ์และความความซับซ้อนมีนัยสำคัญต่อขนาดของคลาส (บรรทัดของรหัส) ดังนั้นปัจจัยด้านคุณภาพที่มีอิทธิพลต่อขนาดของคลาสมากที่สุด คือ ความถูกต้อง (Correctness) การบำรุงรักษา (Maintainability) ประสิทธิภาพ (Efficiency) และการใช้งาน (Usability)

งานวิจัยที่เกี่ยวข้องกับการวัดค่าความซับซ้อนของซอร์สโค้ด มักจะนำตัวชี้วัด Cyclomatic Complexity (CC) มาใช้ ซึ่งเป็นตัวชี้วัดที่ได้รับความนิยมจากนักวิจัยทั่วโลก ถึงแม้ว่าปัจจุบันจะมีตัวชี้วัดใหม่ ๆ ก็ตาม ในทำนองเดียวกัน Walden และคณะ ได้ทำการวิเคราะห์ขนาดของซอร์สโค้ดและนำตัวชี้วัดความซับซ้อน (Complexity Metrics) มาใช้ในการคาดการณ์ความหนาแน่นที่จะเกิดช่องโหว่ใน 14 เว็บแอปพลิเคชัน (Walden, et al., 2009) จากผลการศึกษาพบว่านักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์สสามารถสร้างโครงสร้างของระบบที่เรียบง่าย และเป็นผลที่แสดงให้เห็นว่ามีการปรับปรุงซอฟต์แวร์ โดยเฉพาะทางด้านความมั่นคงเพราะมีจำนวนความหนาแน่นของช่องโหว่ลดน้อยลง และสำหรับงานวิจัยของ Norick และคณะ ได้ทำการศึกษาโครงการโอเพนซอร์สที่ถูกพัฒนาด้วยภาษาซีหรือซีพลัสพลัส (C/C++) ทั้งหมด 11 โครงการ ได้แก่ Filezilla, WinSCP, Miktex, Notepad++, UltraVNC, Audacity, Exult, SMPlayer, TCL, Wireshark, และ TortoiseSVN โดยทำการวิเคราะห์เพื่อตรวจสอบว่าจำนวนเวอร์ชันไฟล์ที่ได้รับการแก้ไขของนักพัฒนาซอฟต์แวร์มีผลกระทบต่อคุณภาพของซอร์สโค้ดหรือไม่ (Norick, et al., 2010) ในการตรวจสอบดังกล่าวจะใช้การวัดความซับซ้อน (Cyclomatic Complexity) (Ebert and Cain, 2016) ของบรรทัดซอร์สโค้ดต่อหนึ่งฟังก์ชัน และความหนาแน่นของข้อเสนอแนะเป็นตัวแทนในการวัดคุณภาพของซอร์สโค้ด แต่ผลของงานวิจัยนี้กลับไม่พบหลักฐานที่จะชี้ให้เห็นว่าจำนวนของนักพัฒนาที่ทำการอัปโหลดไฟล์ที่แก้ไข (Commit) มีความสัมพันธ์กับคุณภาพของซอฟต์แวร์ เพราะค่าเฉลี่ยในการวัดคุณภาพของซอฟต์แวร์ในแต่ละโครงการโอเพนซอร์สอยู่ในเกณฑ์คุณภาพดี ยกเว้นโครงการ TCL ที่มีแนวโน้มคุณภาพของซอฟต์แวร์ที่ไม่ดีเท่าที่ควร เมื่อเทียบกับโครงการอื่น ๆ ดังนั้นจึงสรุปได้ว่าหากนักพัฒนาในโครงการโอเพนซอร์สมีจำนวนมากหรือน้อยจนเกินไปก็จะส่งผลต่อคุณภาพของซอฟต์แวร์

ในขณะที่งานวิจัยของ Schmidt และ Porter ได้ทำการศึกษากระบวนการพัฒนาในโครงการซอฟต์แวร์โอเพนซอร์ส โดยตรวจสอบจากโครงการที่เรียกว่า “Skoll” ซึ่งเป็นการรวมเว็บไซต์หลาย ๆ เว็บที่เป็นโครงการโอเพนซอร์สเข้าด้วยกัน และทำการเปรียบเทียบระหว่างโครงการ Skoll ที่เปิดโอกาสให้เข้าถึงแหล่งเก็บซอร์สโค้ดกับโครงการพัฒนาที่ไม่มีการเผยแพร่ซอร์สโค้ด (Close Source) ซึ่งผลจากงานวิจัยนี้พบว่าชุมชนนักพัฒนาในโครงการโอเพนซอร์สสามารถควบคุมขั้นตอนการบำรุงรักษาในระยะยาวและค่าใช้จ่ายในการพัฒนา สร้างความมั่นใจหรือระดับการยอมรับในคุณภาพของซอฟต์แวร์

การทำให้ผู้ใช้มีความเชื่อมั่นที่จะใช้งานซอฟต์แวร์ รวมถึงการสร้างความมั่นใจในการเชื่อมโยงหรือความสัมพันธ์ของส่วนประกอบภายในซอฟต์แวร์และคุณสมบัติการใช้งานของระบบ โดยโครงการโอเพนซอร์สสามารถทำให้สิ่งเหล่านี้ให้เป็นเรื่องที่ง่าย เมื่อเทียบกับการพัฒนาโครงการที่ไม่มีการเผยแพร่ซอร์สโค้ด เพราะการเปิดเผยแหล่งเก็บซอร์สโค้ดจะทำให้นักพัฒนาที่สนใจโครงการโอเพนซอร์สได้เข้าร่วมปรับปรุงคุณภาพของซอฟต์แวร์โอเพนซอร์สด้วยความสมัครใจ (Schmidt and Porter, 2001)

งานวิจัยทั้งหมดข้างต้นนี้ เป็นงานวิจัยเกี่ยวกับการตรวจสอบคุณภาพของซอฟต์แวร์ในโครงการโอเพนซอร์ส ซึ่งผลจากงานวิจัยทำให้เห็นว่าโครงการโอเพนซอร์สมีการพัฒนาและการปรับปรุงคุณภาพของซอฟต์แวร์อย่างต่อเนื่อง เพื่อที่จะเพิ่มความมั่นใจและความน่าเชื่อในการใช้งานซอฟต์แวร์โอเพนซอร์ส

2.3.2 กระบวนการตรวจทานโค้ดและเครื่องมือที่ใช้ในการตรวจทานโค้ด

โครงการโอเพนซอร์สที่ประสบความสำเร็จมักจะมีการนำกระบวนการตรวจทานโค้ดมาประยุกต์ใช้ เพื่อเป็นสิ่งที่ช่วยในการประกันคุณภาพของซอฟต์แวร์ (Asundi and Jayant, 2007) ซึ่งกระบวนการตรวจทานโค้ดจะเริ่มต้นจากนักพัฒนาซอฟต์แวร์สร้างชุดข้อมูล โดยที่ชุดข้อมูลนั้นเป็นสิ่งที่นักพัฒนาซอฟต์แวร์ต้องการเพิ่มมูลค่าให้กับโครงการ ถึงแม้ว่าหลักปฏิบัติเกี่ยวกับกระบวนการตรวจทานโค้ดจะมีความแตกต่างกันไปตามการพัฒนาดังเดิมของโครงการโอเพนซอร์สนั้น ๆ เช่น โครงการอาปาเช่ (Apache) ได้มีการใช้การตรวจทานโค้ดที่เรียกว่า ตรวจทานก่อนอัปโหลด (Review Then Commit : RTC) ซึ่งเป็นกระบวนการที่นักพัฒนาซอฟต์แวร์ทำการร้องขอไปยังผู้ตรวจทานโค้ด เพื่อให้ทำการตรวจสอบซอร์สโค้ดและให้คำแนะนำเกี่ยวกับการปรับปรุงโค้ดกลับไปทางกลุ่มรายชื่ออีเมล์ (Mailing Lists) ซึ่งก็คือ เมื่อส่งอีเมล์ไปที่กลุ่มรายชื่อนั้น ๆ ระบบก็จะช่วยทำการกระจายไปยังอีเมล์ที่มีรายชื่ออยู่ในกลุ่มโดยอัตโนมัติ หากนักพัฒนาซอฟต์แวร์ไม่ต้องทำการแก้ไขโค้ดตามคำแนะนำ ผู้ตรวจทานโค้ดจะทำการอัปโหลดไฟล์ขึ้นไปยังพื้นที่เก็บข้อมูล แต่ก่อนที่จะเพิ่มไฟล์ใด ๆ ลงในที่เก็บข้อมูล ชุดของไฟล์จะต้องได้รับการตรวจสอบจากทีมนักพัฒนาหลักของโครงการอย่างน้อยสามคน และหากนักพัฒนาซอฟต์แวร์ต้องทำการแก้ไขโค้ดก็จะกลับเข้าสู่กระบวนการร้องขอไปยังผู้ตรวจทานโค้ดอีกครั้ง

ในทางตรงข้ามกับการตรวจทานก่อนอัปโหลด โครงการโอเพนซอร์สบางโครงการจะมีการเปิดให้ผู้มีส่วนได้ส่วนเสียมีส่วนร่วมในพื้นที่เก็บข้อมูลที่ใช้ร่วมกัน (Shared Code Repository) ก่อนที่จะทำการตรวจทานโค้ด โดยนักพัฒนาซอฟต์แวร์จะทำการอัปโหลดชุดของไฟล์ที่ทำการพัฒนาหรือแก้ไขลงในพื้นที่เก็บข้อมูลและส่งคำร้องขอทางอีเมล์เพื่อแจ้งให้ผู้ตรวจทานโค้ดรับทราบ จากนั้นผู้ตรวจทานโค้ดก็จะทำการตรวจสอบซอร์สโค้ดและให้คำแนะนำกลับไป ซึ่งทีมนักพัฒนาหลักของโครงการจะต้องทำการตรวจสอบไฟล์ที่ได้รับการตรวจทานโค้ดทั้งหมด โดยหากชุดของไฟล์ได้รับการยอมรับ (Accept) หรือไม่มีการแก้ไขใด ๆ ก็คือว่าเป็นจุดสิ้นสุดของกระบวนการตรวจทานโค้ด แต่ถ้าชุดของไฟล์

ได้รับการปฏิเสธ (Reject) ทีมนักพัฒนาหลักก็จะทำการลบชุดของไฟล์ออกจากพื้นที่เก็บข้อมูล ซึ่งรูปแบบการตรวจทานโค้ดนี้เรียกว่า อัปโหลดก่อนตรวจทาน (Commit Then Review : CTR)

ส่วนใหญ่แล้วโครงการโอเพนซอร์สมักจะใช้ “ตรวจทานก่อนอัปโหลด” แต่บางโครงการก็จะใช้ “อัปโหลดก่อนตรวจทาน” (Rigby and Bird, 2013) ซึ่งการเลือกใช้รูปแบบการตรวจทานโค้ดจะขึ้นอยู่กับลักษณะของชุดข้อมูล (Patch) และทีมนักพัฒนาหลักในโครงการที่จะต้องมีการควบคุมการพัฒนาซอฟต์แวร์ นั่นคือ กระบวนการพัฒนา หลักปฏิบัติงานที่ถูกต้อง และข้อบังคับต่าง ๆ ที่มีการกำหนดร่วมกันโดยทีมนักพัฒนาหลัก ผู้มีส่วนได้ส่วนเสีย และนักพัฒนาซอฟต์แวร์ภายในโครงการ

ถึงแม้ว่างานวิจัยของ Rigby ได้กล่าวว่า การใช้กลุ่มรายชื่ออีเมล์ในการกระจายข่าวสารหรือชุดของไฟล์ให้กับทุกคนในกลุ่มเป็นวิธีที่มีประสิทธิภาพและเหมาะสมกับทีมนักพัฒนา อย่างไรก็ตามหากไม่มีการตรวจสอบย้อนกลับและเครื่องมือที่ช่วยสนับสนุนการตรวจทานโค้ดก็จะทำให้ยากต่อการตรวจสอบความก้าวหน้าของโครงการโอเพนซอร์ส (Rigby, 2011) ด้วยเหตุนี้จึงทำให้มีการผลักดันให้ทำการพัฒนาซอฟต์แวร์ที่อำนวยความสะดวกในเรื่องการตรวจทานโค้ด จนถึงปัจจุบันมีการใช้งานเครื่องมือเกี่ยวกับการตรวจทานโค้ดอย่างแพร่หลาย เช่น ReviewBoard, Gerrit, Crucible

งานวิจัยของ Baysal และคณะ ได้ทำการศึกษาเกี่ยวกับปัจจัยที่ส่งผลต่อการแก้ไขข้อบกพร่องของโปรแกรม (Patch) โดยทำการสกัดข้อมูล (Extract) ในกระบวนการตรวจทานโค้ดของโครงการ WebKit จาก Bugzilla ซึ่งผลที่ได้จากการตรวจสอบพบว่า ผู้ตรวจทานโค้ดส่วนใหญ่มักจะทำการปฏิเสธชุดของการแก้ไขซอร์สโค้ดขนาดใหญ่ เพราะนักพัฒนาซอฟต์แวร์บางคนมักจะแก้ไขหรือพัฒนาซอร์สโค้ดที่มีความซับซ้อนและมีส่วนประกอบที่ไม่จำเป็นต่อการทำงานของระบบ อีกทั้งยังส่งผลกระทบต่อฟังก์ชันหรือโมดูลอื่น ๆ (Baysal, et al., 2013) ในขณะที่งานวิจัยของ Tao และคณะ ได้นำเสนอแนวทางที่จะช่วยให้นักพัฒนาเขียนโปรแกรมที่ใช้แก้ไขข้อบกพร่องของซอฟต์แวร์ได้รับการยอมรับโดยผู้ตรวจทานโค้ด ซึ่งงานวิจัยนี้จะใช้วิธีการสังเกต เพื่อตรวจสอบหาสาเหตุที่ทำให้โปรแกรมแก้ไขซอฟต์แวร์ถูกปฏิเสธ โดยรวบรวมข้อมูลการปฏิเสธโปรแกรมที่ได้รับการแก้ไขในโครงการซอฟต์แวร์โอเพนซอร์สจำนวน 2 โครงการ นั่นคือ โครงการ Eclipse และโครงการ Mozilla ซึ่งข้อมูลที่นำมาวิเคราะห์จะใช้วิธีเก็บข้อมูล โดยอาศัยแบบสำรวจออนไลน์จากนักพัฒนาซอฟต์แวร์จำนวน 246 คน (Tao, et al., 2014)

งานวิจัยที่ได้กล่าวไปข้างต้นจะเกี่ยวข้องกับชุดของการแก้ไขข้อบกพร่องของโปรแกรมที่เกิดขึ้นในกระบวนการตรวจทานโค้ด โดยทางผู้วิจัยจะต้องทราบถึงสาเหตุหรือปัญหาที่ทำให้โปรแกรมที่ได้รับการแก้ไขข้อบกพร่องนั้นได้รับการยอมรับหรือถูกปฏิเสธ เพื่อที่จะสามารถนำไปวิเคราะห์และค้นหาแนวทางในการเพิ่มประสิทธิภาพของกระบวนการตรวจทานโค้ด

ในปัจจุบันนักวิจัยและวิศวกรซอฟต์แวร์ได้ให้ความสนใจเกี่ยวกับการตรวจทานโค้ดเพิ่มมากขึ้น เนื่องจากการตรวจทานโค้ดสมัยใหม่มีการประยุกต์ใช้เทคโนโลยีเป็นสื่อกลางในการสื่อสาร จึงทำให้ผู้ใช้งานสามารถติดต่อสื่อสารกันได้อย่างสะดวก โดยที่ไม่ต้องอยู่ในสถานที่เดียวกันหรือในช่วงเวลาเดียวกัน ด้วยเหตุนี้จึงทำให้มีการศึกษาเกี่ยวกับการตรวจทานโค้ดในหลากหลายแง่มุมและมีการพัฒนาเครื่องมือที่ใช้ในการตรวจทานโค้ดเพิ่มมากขึ้น ตัวอย่างเช่น งานวิจัยของ Bosu และ Carver ได้ศึกษาข้อเสนอแนะภายใต้เครื่องมือที่ใช้ในการตรวจทานโค้ดมีชื่อว่า “ReviewBoard” สำหรับวัตถุประสงค์ของงานวิจัยนี้ คือ การวิเคราะห์จำนวนผู้มีส่วนร่วมในโครงการและจำนวนของผู้ส่งคำร้องขอการตรวจทานโค้ด รวมถึงเวลาในการตอบกลับหลังจากส่งคำร้องขอการตรวจทาน (Bosu and Carver, 2012) โดยทำการเปรียบเทียบระหว่างโครงการโอเพนซอร์สจำนวน 2 โครงการด้วยกัน นั่นคือ โครงการ MusicBrainz Server และโครงการ Asterisk

แต่สำหรับงานวิจัยของ Bacchelli และ Bird เป็นการศึกษาเกี่ยวกับแรงจูงใจและความท้าทายในการค้นหาข้อบกพร่องของโครงการต่าง ๆ ภายใต้บริษัทไมโครซอฟต์ โดยใช้เครื่องมือที่ชื่อว่า “CodeFlow” ซึ่งเป็นโปรแกรมในการบันทึกข้อมูลของการตรวจทานโค้ด โดยข้อมูลที่ถูกนำมาวิเคราะห์จะเกี่ยวกับข้อเสนอแนะที่เป็นแนวทางการการแก้ไขโค้ดของกระบวนการตรวจทานโค้ด ซึ่งผลจากการวิจัยเกี่ยวกับการใช้เครื่องมือในกระบวนการตรวจทานโค้ดสมัยใหม่ทั้ง 2 งานวิจัยพบว่า หากผู้ตรวจทานโค้ดมีความรู้ความเชี่ยวชาญเกี่ยวกับการพัฒนาระบบก็จะส่งผลต่อความสามารถในการอ่านซอร์สโค้ดและทำการตรวจทานโค้ดได้อย่างรวดเร็ว พร้อมทั้งให้ข้อเสนอแนะที่เป็นประโยชน์ต่อนักพัฒนาที่จะทำการแก้ไข ดังนั้นจึงสรุปผลได้ว่าความเชี่ยวชาญและประสบการณ์ของนักพัฒนาซอฟต์แวร์ที่ทำหน้าที่ผู้ตรวจทานโค้ดจะมีผลโดยตรงต่อคุณภาพของซอร์สโค้ดในโครงการโอเพนซอร์ส (Bacchelli and Bird, 2013)

จากการศึกษาเกี่ยวกับเครื่องมือที่ใช้ในการตรวจทานโค้ดพบว่า ถึงแม้จะใช้เครื่องมือการตรวจทานโค้ดเพื่อหาคำตอบของงานวิจัยที่แตกต่างกัน แต่มีวัตถุประสงค์ในการใช้งานอย่างเดียวกัน คือ เป็นเครื่องมือที่อำนวยความสะดวกในการตรวจทานโค้ดและเป็นสื่อกลางในการติดต่อสื่อสารระหว่างนักพัฒนาซอฟต์แวร์ด้วยกัน นอกจากเครื่องมือ ReviewBoard และ CodeFlow ที่ได้กล่าวไว้ข้างต้น ปัจจุบันยังมีเครื่องมือหรือระบบที่ให้บริการเกี่ยวกับการตรวจทานโค้ดมากกว่า 10 เครื่องมือให้เลือกใช้งาน

งานวิจัยของ Bosu และคณะ ได้ทำการศึกษาเกี่ยวกับการตรวจทานโค้ดและวิธีในการเพิ่มประสิทธิภาพของการตรวจทานโค้ดที่มีประโยชน์ต่อการพัฒนาระบบ รวมถึงช่วยประหยัดค่าใช้จ่าย โดยงานวิจัยนี้จะทำการวิเคราะห์ข้อเสนอแนะจำนวน 1.5 ล้านที่ได้จากโครงการของไมโครซอฟต์ 5 โครงการ ซึ่งผลจากการวิจัยจะเป็นการระบุลักษณะของข้อเสนอแนะที่มีประโยชน์และไม่มีประโยชน์

โดยทั่วไปข้อเสนอแนะที่นักพัฒนาได้รับอาจมีข้อมูลที่ไม่ถูกต้องหรืออาจให้ข้อเสนอแนะอื่น ๆ ที่ไม่เกี่ยวข้องกับการพัฒนาระบบ จึงทำให้นักพัฒนาซอฟต์แวร์ต้องใช้เวลาในการตอบคำถามและระบุสาเหตุที่ไม่ปรับปรุงซอร์สโค้ด นอกจากนี้ยังได้กล่าวถึงปัจจัยที่มีอิทธิพลต่อประโยชน์ของข้อเสนอแนะ (Bosu, et al., 2015) ได้แก่

(1) ลักษณะของผู้ตรวจทานโค้ดและทีม ได้แก่ ประสบการณ์เกี่ยวกับการพัฒนาซอฟต์แวร์ในการตรวจทาน ประสบการณ์ในองค์กร และการอยู่ในทีมเดียวกับผู้เขียนโค้ดที่ทำการเปลี่ยนแปลง

(2) จากการสำรวจการแก้ไขโค้ดภายใต้กระบวนการตรวจทานโค้ดแสดงให้เห็นว่ามีจำนวนข้อเสนอแนะที่มีประโยชน์มากกว่า 60% ต่อจำนวนของข้อเสนอแนะทั้งหมด

งานวิจัยของ Czerwonka และ Greiler ได้ทำการศึกษาข้อเสนอแนะที่ได้รับการตอบกลับจากผู้ตรวจทานโค้ด จากผลการศึกษาพบว่า มีจำนวนข้อเสนอแนะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษามากถึง 50% ของข้อเสนอแนะทั้งหมด และการตรวจทานโค้ดจะมีความสัมพันธ์เชิงลบกับขนาดของไฟล์ที่ตรวจทาน นั่นคือ การที่มีไฟล์ขนาดใหญ่มาก ๆ จะส่งผลให้การตรวจทานหนึ่งครั้งผู้ตรวจทานโค้ดจะให้ข้อเสนอแนะที่มีประโยชน์ลดลงและใช้เวลาเฉลี่ย 6 ชั่วโมงต่อสัปดาห์ สำหรับตรวจทานไฟล์ที่มีการเปลี่ยนแปลง รวมถึงผู้ตรวจทานโค้ดยังต้องอาศัยความพยายามเป็นอย่างมากในแง่ของการให้ข้อเสนอแนะที่มีประโยชน์ ด้วยเหตุนี้จึงทำให้ผู้ตรวจทานโค้ดมักจะใช้เวลาในการตรวจทานโค้ดค่อนข้างนาน (Czerwonka and Greiler, 2015)

จากการศึกษางานวิจัยข้างต้นจะเห็นได้ว่าข้อเสนอแนะของผู้ตรวจทานโค้ดมีประโยชน์และสะท้อนให้เห็นถึงมุมมองการพัฒนาซอฟต์แวร์ รวมทั้งยังแสดงให้เห็นถึงความร่วมมือในการแก้ไขปัญหาของนักพัฒนาซอฟต์แวร์ที่จะช่วยให้สามารถพัฒนาหรือแก้ไขปรับปรุงซอฟต์แวร์ได้อย่างรวดเร็ว โดยการศึกษาข้อเสนอแนะในงานวิจัยดังกล่าวถือได้ว่าเป็นแนวทางที่ช่วยให้ผู้วิจัยสามารถเข้าใจภาพรวมของการตรวจทานโค้ดได้ง่ายยิ่งขึ้น

2.3.3 ความสามารถในการบำรุงรักษา

การบำรุงรักษาเป็นขั้นตอนที่สำคัญมาก เพราะเป็นขั้นตอนที่ใช้เวลามากถึง 66% ของวงจรชีวิตในการพัฒนาซอฟต์แวร์ และเสียค่าใช้จ่ายในการบำรุงรักษาประมาณ 40-80% (เฉลี่ย 60%) ซึ่งหนึ่งในปัญหาที่ต้องได้รับการแก้ไขอย่างเร่งด่วน คือ ร่องรอยโค้ดที่ไม่ดี (Code Smell) ซึ่งเป็นโค้ดที่เกิดจากการออกแบบโครงสร้างไม่ดีและนำไปสู่การดำเนินการพัฒนาอย่างไม่เป็นระบบ โดยสาเหตุเหล่านี้ทำให้คุณภาพของความสามารถในการบำรุงรักษาลดลง

งานวิจัยของ Wagey และคณะ ได้นำเสนอรูปแบบของความสามารถในการบำรุงรักษาซอฟต์แวร์ โดยใช้แบบแผนหรือแนวทางที่ใช้ในการแก้ไขปัญหาที่เกิดขึ้นจากการออกแบบ มาประยุกต์ใช้ในโครงการโอเพนซอร์สจำนวน 6 โครงการ (Wagey, et al., 2015) โดยผลจากการวิจัยพบว่า รูปแบบของความสามารถในการบำรุงรักษาดังกล่าวสามารถช่วยให้โครงการโอเพนซอร์สมีโครงสร้างของระบบที่ดีขึ้นและสำหรับเทคนิคที่นิยมใช้ในการปรับปรุงโครงสร้างภายในของระบบ คือ การทำรีแฟคทอริง (Refactoring) (Kádár, et al., 2016) เป็นการแก้ไขปรับปรุงฟังก์ชันการทำงานของซอฟต์แวร์ที่เคยเขียนไปแล้ว โดยทำให้โค้ดมีบรรทัดที่สั้นลง หรือทำให้รวมฟังก์ชันการทำงานที่เหมือนหรือคล้ายกันให้อยู่ในฟังก์ชันเดียวกัน เพื่อลดความซับซ้อนของโค้ดและช่วยให้นักพัฒนาสามารถอ่านและทำความเข้าใจกับโค้ดได้ง่ายมากขึ้น (Stroggylos and Spinellis, 2007)

เนื่องจากต้นทุนของการพัฒนาซอฟต์แวร์อย่างน้อยสองในสามนั้นมักเกิดขึ้นจากขั้นตอนการบำรุงรักษา ซึ่งหากปราศจากการออกแบบที่ดีและการวัดเพื่อประเมินคุณภาพของซอฟต์แวร์แล้ว ต้นทุนของการบำรุงรักษานั้นอาจจะสูงขึ้นอีกได้ ดังนั้นการวัดคุณภาพซอฟต์แวร์เชิงวัตถุจึงได้รับการสนับสนุนจากงานวิจัยหลายงานที่ทำการวัดคุณภาพภายใต้แนวคิดปัจจัย เกณฑ์ และมาตรวัดเดียวกัน แต่ก็มีงานวิจัยบางส่วนที่นำเสนอการวัดคุณภาพของซอฟต์แวร์เชิงแง่มุมโดยใช้แนวคิดปัจจัยและกลยุทธ์ โดยจากการศึกษาเกี่ยวกับคุณลักษณะของคุณภาพซอฟต์แวร์พบว่า รูปแบบของความสามารถในการบำรุงรักษา (Maintainaiblity Model) จะขึ้นอยู่กับค่าพารามิเตอร์หลาย ๆ ค่าและคุณลักษณะย่อยของความสามารถในการบำรุงรักษา (Ghosh, et al., 2011) เช่น งานวิจัยของ Peercy เป็นการพัฒนาโครงสร้าง (Framework) ของความสามารถในการบำรุงรักษาซอฟต์แวร์ที่แสดงให้เห็นว่าแต่ละโปรแกรมของซอฟต์แวร์จะประกอบไปด้วยชุดขององค์ประกอบที่เรียกว่า “โมดูล” (Peercy, 1981) ส่วนงานวิจัยของ Abreu และ Melo ได้ใช้รูปแบบของคุณภาพซอฟต์แวร์ (Software Quality Model) ในการประเมินผลกระทบของการออกแบบเชิงวัตถุ (Object-Oriented Design) ที่มีคุณลักษณะของความสามารถในการบำรุงรักษาและความน่าเชื่อถือ (Reliability) โดยที่ตัวชี้วัดที่นำมาใช้วัดคุณลักษณะของการออกแบบเชิงวัตถุ คือ MOOD (Metrics for Object-Oriented Design) (Abreu and Melo, 1996) และงานวิจัยของ Harrison ได้ระบุว่า ความซับซ้อนของซอร์สโค้ดเป็นปัจจัยที่ส่งผลต่อคุณลักษณะย่อยของความสามารถในการบำรุงรักษา ได้แก่ ความสามารถในการทำความเข้าใจ (Understandability) ความสามารถในการแก้ไข (Modifiability) และความสามารถในการทดสอบ (Testability) ซึ่งความซับซ้อนของซอฟต์แวร์นี้จะเกี่ยวข้องโครงสร้างโมดูล (Modularity) การเชื่อมโยง (Coupling) และการทำงานร่วมกัน (Cohesion) ของส่วนประกอบภายในระบบ นอกจากนี้รูปแบบของความสามารถในการบำรุงรักษายังเป็นตัวชี้วัดความสามารถในการอ่านซอร์สโค้ด (Readability) และการจัดการเอกสารให้มีคุณภาพยิ่งขึ้นอีกด้วย (Harrison, et al., 1982)

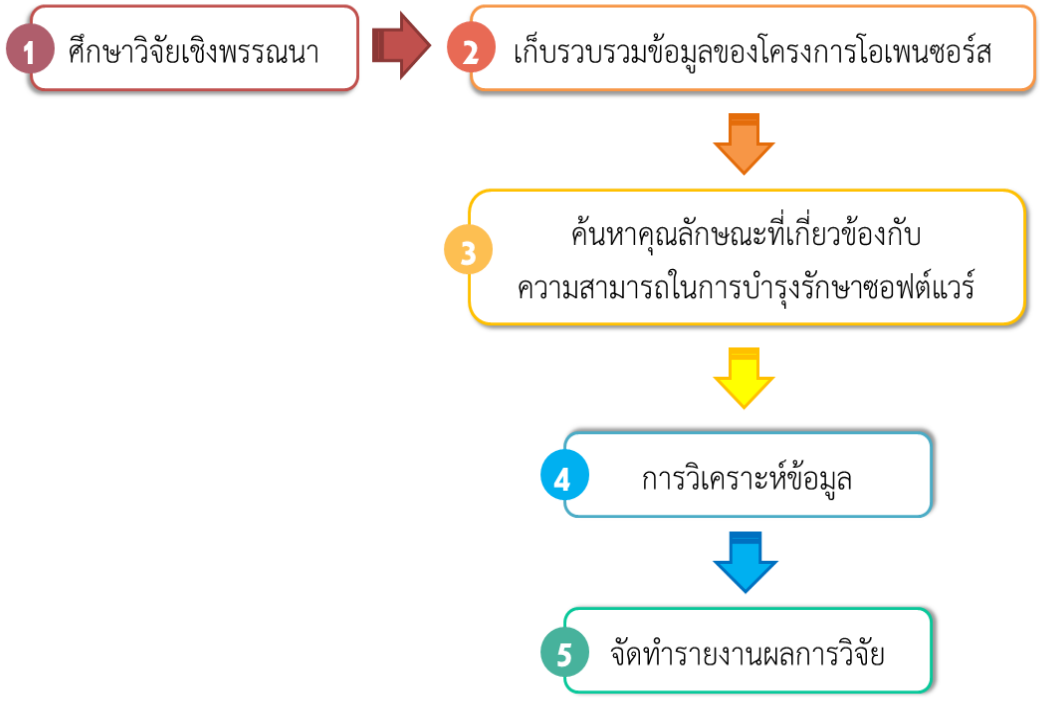
งานวิจัยของ Rizvi และ Khan ได้ใช้วิธีทางสถิติ Regression Analysis ในการสร้างโมเดลที่ใช้ในการพยากรณ์ความสามารถในการบำรุงรักษาซอฟต์แวร์ (Maintainability Estimation Model) ในระดับคลาส ซึ่งมีชื่อว่า MEMOOD เพื่อเป็นการอำนวยความสะดวกและลดเวลาในการปรับปรุงหรือการแก้ไขงานก่อนการส่งมอบ โดยรูปแบบหรือโมเดลนี้จะใช้คุณลักษณะของความสามารถในการเข้าใจโค้ด (Understandability) และความสามารถในการปรับเปลี่ยน (Modifiability) มาคำนวณด้วยมาตรวัดเชิงวัตถุ (Object-Oriented Metrics) เพื่อหาค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ โดยใช้มาตรวัดเกี่ยวกับขนาด (Size) และความซับซ้อนของโครงสร้างระบบ (Strutural Complexity) รวมถึงยังทำการคำนวณจากแผนภาพคลาส (Class Diagrams) โดยการประเมินโมเดลนี้จะใช้วิธีการนำค่าที่ได้จากการคำนวณโมเดลมาเปรียบเทียบกับค่าจริง (Actual) (Rizvi and Khan, 2010) และงานวิจัยของ Khan และ Khan ได้ทำการสร้างโมเดลที่ใช้ในการวัดคุณภาพทางด้านความสามารถในการวิเคราะห์ข้อผิดพลาด (Analyzability) ด้วยการวิเคราะห์การถดถอยเชิงเส้นแบบพหุคูณ (Multiple Linear Regression) ผลที่ได้จากงานวิจัยพบว่า การออกแบบโครงสร้างการทำงานของระบบที่ซับซ้อน (Complexity) เป็นปัจจัยที่อาจส่งผลต่อคุณภาพของความสามารถในการบำรุงรักษาซอฟต์แวร์ โดยเฉพาะคุณลักษณะทางด้านการวิเคราะห์ข้อผิดพลาด แต่มีข้อสังเกตว่า ในงานวิจัยยังแสดงให้เห็นว่ามีปัจจัยอื่น ๆ ที่อาจส่งผลให้การวิเคราะห์ข้อผิดพลาดทำได้ยากขึ้น เช่น ขนาดของโปรแกรม (Khan and Khan, 2012)

งานวิจัยทางด้านความสามารถในการบำรุงรักษาซอฟต์แวร์ส่วนใหญ่มักจะเป็นการสร้างโมเดลหรือการใช้เครื่องมือเพื่อตรวจสอบคุณภาพของซอร์สโค้ดที่นักพัฒนาซอฟต์แวร์ได้ทำการพัฒนาขึ้น ในขณะที่งานวิจัยที่เกี่ยวข้องกับการศึกษาความสามารถในการบำรุงรักษาซอฟต์แวร์ในด้านอื่น ๆ ยังมีค่อนข้างน้อย โดยเฉพาะการศึกษาทางด้านการวิเคราะห์ข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ ดังนั้นผู้วิจัยจึงมีความสนใจที่จะศึกษาข้อเสนอแนะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษามากกว่าการศึกษาซอร์สโค้ดในระบบที่มีนักวิจัยทั่วโลกได้ทำการศึกษาไปแล้ว ส่วนสาเหตุที่ผู้วิจัยศึกษาเฉพาะทางด้านความสามารถในการบำรุงรักษาซอฟต์แวร์ เพราะเป็นคุณลักษณะที่นักพัฒนาซอฟต์แวร์ทุกคนควรที่จะใส่ใจ อีกทั้งยังช่วยในเรื่องของการแก้ไขข้อบกพร่อง การเพิ่มฟังก์ชันหรือคุณสมบัติที่จะเป็นเวอร์ชันใหม่ของระบบ รวมไปถึงอำนวยความสะดวกในการบำรุงรักษาในอนาคตที่จะช่วยลดเวลาและค่าใช้จ่ายในกระบวนการพัฒนาซอฟต์แวร์ ตั้งแต่ขั้นตอนการวางแผนจนกระทั่งส่งมอบซอฟต์แวร์ให้แก่ลูกค้าหรือการนำไปใช้งานจริง

บทที่ 3

วิธีดำเนินการวิจัย

การศึกษาวิจัยนี้เป็นการวิเคราะห์ข้อเสนอแนะเกี่ยวกับความสามารถในการบำรุงรักษาซอฟต์แวร์ภายใต้กระบวนการตรวจทานโค้ดของโครงการซอฟต์แวร์โอเพนซอร์ส ด้วยเทคนิคการทำเหมืองข้อความ (Text Mining) สำหรับวิธีการดำเนินงานวิจัยแสดงดังภาพที่ 3.1 โดยมีขั้นตอนในการดำเนินงานวิจัยจำนวน 5 ขั้นตอน ดังนี้ 1) ศึกษาวิจัยเชิงพรรณนา 2) เก็บรวบรวมข้อมูลของโครงการโอเพนซอร์ส 3) ค้นหาคุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษา 4) การวิเคราะห์ข้อมูล และ 5) จัดทำรายงานผลการวิจัย โดยรายละเอียดในแต่ละขั้นตอนมีดังต่อไปนี้



รูปที่ 3.1 ขั้นตอนในการดำเนินงานวิจัย

3.1 การศึกษาวิจัยเชิงพรรณนา

สำหรับขั้นตอนแรกในการดำเนินงานวิจัยนี้ ทางผู้วิจัยได้ทำการศึกษาแนวคิด หลักการและเทคโนโลยีที่เกี่ยวข้องกับการตรวจทานโค้ด และทำการสืบค้นเอกสารวิชาการ บทความ วรรณกรรมต่าง ๆ ที่มีการศึกษาเกี่ยวกับความสามารถในการบำรุงรักษา รวมไปถึงงานวิจัยที่เกี่ยวข้องกับคุณภาพของซอฟต์แวร์โอเพนซอร์ส เพื่อนำมาเป็นแนวทางในการศึกษางานวิจัยนี้

3.2 การเก็บรวบรวมข้อมูลของโครงการโอเพนซอร์ส

ขั้นตอนนี้จะเริ่มจากการกำหนดข้อมูลที่จะนำไปใช้วิเคราะห์และเก็บรวบรวมข้อมูลที่ต้องการจากระบบเกอริต โดยการพัฒนาซอฟต์แวร์ที่ใช้ในการสืบค้นและจัดเก็บข้อมูล หลังจากนั้นผู้วิจัยทำการตรวจสอบข้อมูลที่ได้สืบค้น เพื่อพิจารณาว่าข้อมูลที่ได้มาเป็นข้อมูลที่ถูกต้องและตรงกับความต้องการที่จะใช้ในการวิเคราะห์เพื่อหาคำตอบของคำถามวิจัย โดยรายละเอียดของขั้นตอนมีดังนี้

3.2.1 การกำหนดโครงการโอเพนซอร์สที่ใช้ในการศึกษา

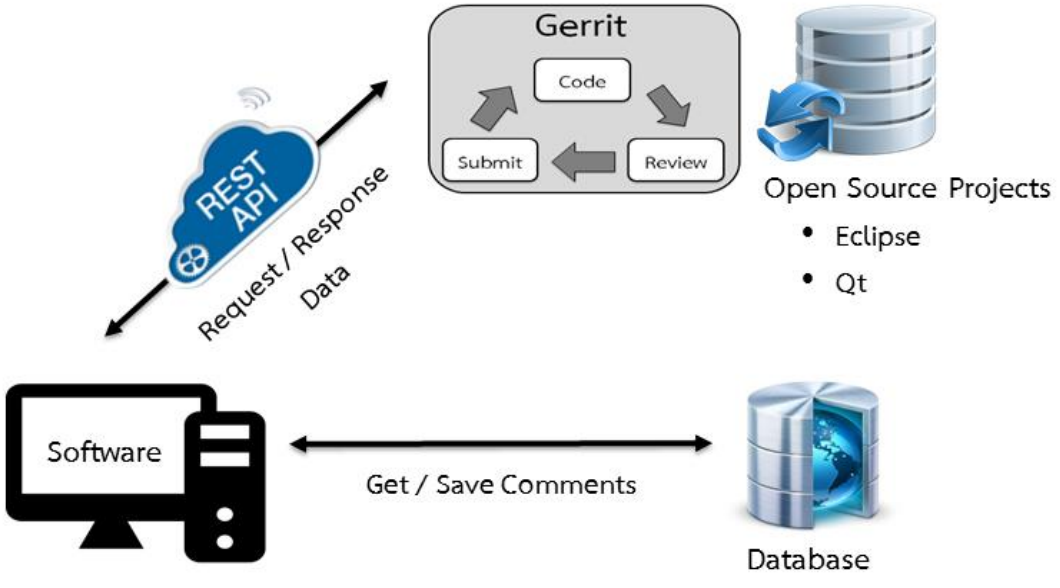
การศึกษางานวิจัยนี้ผู้วิจัยได้ทำการศึกษาโครงสร้างการทำงานของระบบเกอริตและสำรวจโครงการโอเพนซอร์สที่มีข้อมูลจัดเก็บอยู่ในระบบเกอริต จากการศึกษาพบว่าระบบเกอริตมีข้อมูลของโครงการโอเพนซอร์สถูกจัดเก็บไว้จำนวนมาก และในแต่ละโครงการจะมีวิธีการเข้าถึงแหล่งข้อมูลที่แตกต่างกัน ดังนั้นในการเลือกโครงการโอเพนซอร์สที่นำมาศึกษาวิจัยในครั้งนี้ ผู้วิจัยได้กำหนดเกณฑ์ในการพิจารณาดังนี้

1) เป็นโครงการที่มีการใช้งานและการพัฒนาอย่างต่อเนื่องจนถึงปัจจุบัน
2) ได้รับความนิยมในการนำมาศึกษาในงานวิจัยทางด้านวิศวกรรมซอฟต์แวร์
3) เป็นโครงการที่มีความเป็นไปได้ในการเข้าถึงแหล่งข้อมูลที่ต้องการนำมาศึกษา

เมื่อผู้วิจัยทำการพิจารณาโครงการโอเพนซอร์สที่อยู่ภายในระบบเกอริตแล้ว ผู้วิจัยจึงได้เลือกโครงการซอฟต์แวร์โอนเพนซอร์สที่ต้องการศึกษาจำนวน 2 โครงการ คือ โครงการ Eclipse และโครงการ Qt

3.2.2 การพัฒนาซอฟต์แวร์ที่ช่วยในการสืบค้นและจัดเก็บข้อมูลจากระบบเกอริต

การศึกษางานวิจัยนี้ทางผู้วิจัยได้ทำการพัฒนาซอฟต์แวร์ที่ช่วยในการสืบค้นและจัดเก็บข้อมูลเกี่ยวกับข้อเสนอแนะของโครงการซอฟต์แวร์โอนเพนซอร์สจำนวน 2 โครงการ คือ Eclipse และโครงการ Qt ที่เก็บรวบรวมจากระบบเกอริตในช่วงปี ค.ศ. 2012 - 2016 สำหรับการรวบรวมข้อมูลดังกล่าว ผู้วิจัยทำการศึกษาโครงสร้างการเก็บข้อมูล (Database Schema) (Bosu and Carver, 2014) และวิธีการดึงข้อมูลเกี่ยวกับการตรวจทานโค้ดของโครงการ Eclipse และโครงการ Qt ในระบบเกอริต เพื่อนำมาใช้ในการออกแบบและพัฒนาซอฟต์แวร์ จากการศึกษาพบว่า ระบบเกอริตจะมีเว็บเซอร์วิส (Web Services) ที่ให้บริการในการเข้าถึงข้อมูลที่ถูกจัดเก็บอยู่ในฐานข้อมูลของระบบเกอริตมาใช้วิเคราะห์ได้ โดยการทำงานผ่านเชลล์ (Shell) ซึ่งเป็นโปรแกรมที่ทำหน้าที่ติดต่อระหว่างผู้ใช้งานและเว็บเซอร์วิสของระบบเกอริตที่ทำงานร่วมกับ Gerrit REST API (Mukadam, et al., 2013) ดังรูปที่ 3.2 แสดงการทำงานของซอฟต์แวร์ที่ใช้ในการสืบค้นข้อมูล



รูปที่ 3.2 ภาพรวมการทำงานของซอฟต์แวร์ที่ใช้ในการสืบค้นข้อมูล

จากรูปที่ 3.2 ซอฟต์แวร์ที่ทางผู้วิจัยพัฒนามีขั้นตอนการทำงานดังนี้

1) สืบค้นข้อมูลจากระบบเกอริต โดยเขียนเชลล์สคริปต์ (Shell Script) ที่กำหนดเงื่อนไขในการดึงข้อมูลที่ต้องการ

2) พัฒนาโปรแกรมที่ใช้แปลงข้อมูลที่ดึงมาจากระบบเกอริตในรูปแบบของ Java Object Notation (JSON) ด้วยภาษาจาวา โดยแปลงข้อมูลให้อยู่ในรูปแบบที่สามารถจัดเก็บในระบบฐานข้อมูลเชิงสัมพันธ์ (Relational)

3) นำข้อมูลที่ได้จัดเก็บลงฐานข้อมูล MySQL ซึ่งเป็นระบบจัดการฐานข้อมูลแบบสัมพันธ์ โดยทำการเก็บข้อมูลทั้งหมดในรูปแบบของตาราง เพื่อช่วยให้สามารถทำงานได้รวดเร็วและมีความยืดหยุ่น เพราะในแต่ละตารางที่เก็บข้อมูลสามารถเชื่อมโยงเข้าหากันหรือสามารถจัดการข้อมูลในลักษณะต่าง ๆ ได้ โดยอาศัยภาษา Structure Query Language (SQL) ที่เป็นภาษามาตรฐานในการเข้าถึงฐานข้อมูล นอกจากนี้การจัดเก็บข้อมูลในระบบฐานข้อมูล ยังช่วยอำนวยความสะดวกแก่ผู้วิจัยในการจัดการและสืบค้นข้อมูล รวมไปถึงการแปลงข้อมูลให้อยู่ในรูปแบบต่าง ๆ เช่น .excel หรือ .csv เพื่อนำมาใช้ในการวิเคราะห์เพื่อหาคำตอบของงานวิจัยได้ตามที่ผู้วิจัยต้องการ

ก่อนที่จะทำการจัดเก็บข้อมูลที่สืบค้นมาจากระบบเกอริต ผู้วิจัยได้ทำการออกแบบตารางในฐานข้อมูลเบื้องต้น เพื่อรองรับข้อมูลที่จำเป็นต้องใช้ในงานวิจัยครั้งนี้ โดยผู้วิจัยได้กำหนดให้มีข้อมูลที่จะจัดเก็บดังนี้

1) หมายเลขและ URL ของไฟล์ที่ทำการเปลี่ยนแปลง (Id and Link_id)

2) หมายเลขของการแก้ไขโค้ดในโปรแกรม (Patchset_number)

3) ผู้อัปโหลดไฟล์ (Uploader)

4) วันที่และเวลาที่ผู้ตรวจทานโค้ดให้คำแนะนำ (Created_on)

5) ผู้เขียนโค้ดหรือผู้ร้องขอการตรวจทานโค้ด (Author)

6) ผู้ตรวจทานโค้ด (Reviewer)

7) ไฟล์ที่มีการร้องขอให้ตรวจทาน (File)

8) จำนวนข้อเสนอแนะทั้งหมดของไฟล์ที่มีการร้องขอให้ตรวจทาน (SizeInsertion)

9) จำนวนข้อเสนอแนะที่ถูกลบภายในไฟล์ที่มีการร้องขอให้ตรวจทาน (SizeDeletions)

10) หมายเลขบรรทัดโค้ด (Line)

11) ข้อความของข้อเสนอแนะ (Message)

12) ประเภทของการแก้ไขโค้ด (Kind)

3.2.3 การตรวจสอบข้อมูลที่ได้ทำการสืบค้นจากระบบเกอริต

ผู้วิจัยทำการตรวจสอบความสมบูรณ์ของข้อมูลก่อนที่จะนำไปใช้ในการวิเคราะห์ด้วยเทคนิคการทำเหมืองข้อความ ซึ่งกระบวนการนี้จะเป็นการตรวจสอบข้อมูลที่ถูกจัดเก็บอยู่ในฐานข้อมูลเพื่อเตรียมข้อมูลให้พร้อมสำหรับการนำไปใช้ในงานวิจัย สำหรับกระบวนการตรวจสอบข้อมูลดังกล่าวผู้วิจัยจะใช้โปรแกรม R ในการตรวจสอบโดยมี 2 ขั้นตอนดังนี้

1) การตรวจสอบความซ้ำซ้อนของข้อมูล เป็นการตรวจสอบข้อมูลในตารางของโครงการ Eclipse และโครงการ Qt ว่ามีข้อมูลที่เกิดความซ้ำซ้อนหรือไม่ โดยใช้โปรแกรม R ในการตรวจสอบด้วยคำสั่งดังนี้

```
1 !duplicated(Project[c("id","link_id","patchset_number",
2     "uploader","created_on","author","reviewer","file","line",
3     "message", "sizeInsertion","sizeDeletions","kind")])
```

!duplicated() เป็นคำสั่งที่ใช้ในการกรองข้อมูลที่มีความซ้ำซ้อนและทำการลบข้อมูลที่เกิดความซ้ำซ้อนนั้นออกจากฐานข้อมูล โดยมีเงื่อนไขในการระบุให้ตรวจสอบคอลัมน์ทั้งหมดในรูปแบบของตารางที่ถูกจัดเก็บในระบบฐานข้อมูล

2) การตรวจสอบความครบถ้วนของข้อมูล เป็นขั้นตอนที่ผู้วิจัยนำข้อมูลมาทำการตรวจสอบว่ามีข้อมูลที่จำเป็นต่อการนำไปทำการวิเคราะห์ส่วนใดขาดหายไปบ้าง โดยการตรวจสอบค่าว่าง (Null) ของข้อมูลในตารางที่ถูกจัดเก็บในระบบฐานข้อมูล โดยมีคำสั่งในโปรแกรม R ดังนี้

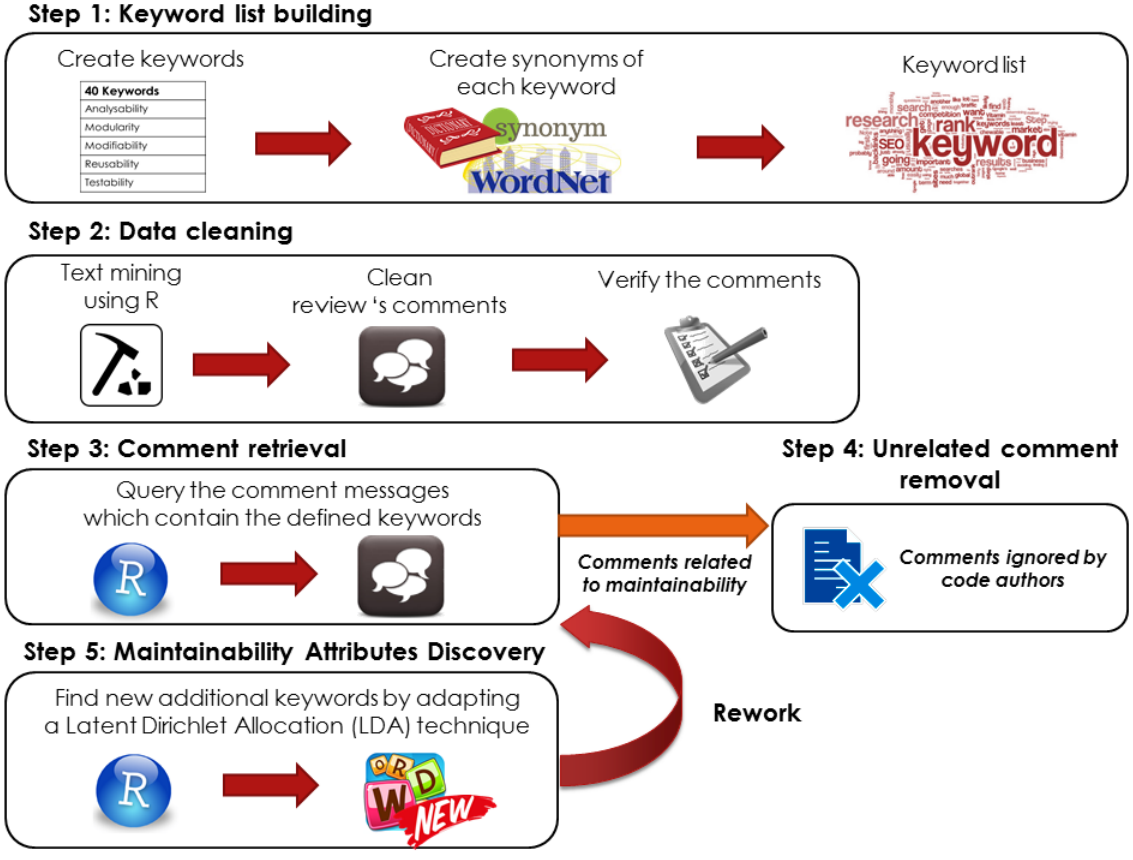
```
1 sqldf("select * from DataEclipse where revision NOT LIKE ''
2       AND uploader NOT LIKE ''   AND created_on NOT LIKE ''
3       AND author NOT LIKE ''     AND reviewer NOT LIKE ''
4       AND file NOT LIKE ''       AND line NOT LIKE ''
5       AND message NOT LIKE ''    AND kind NOT LIKE ''")
```

การเรียกใช้ฟังก์ชัน “sqldf” เพื่อทำการสืบค้นข้อมูลในระบบฐานข้อมูล โดยกำหนด “DataEclipse” เป็นข้อมูลในรูปแบบตารางที่ถูกจัดเก็บอยู่ในฐานข้อมูล และใช้คำสั่ง SQL NOT LIKE เป็นคำสั่งที่ใช้สำหรับการระบุเงื่อนไขการเลือกข้อมูลในตาราง โดยทำการค้นหาค่าว่างที่ปรากฏในคอลัมน์ที่กำหนดและข้อมูลที่ได้จะไม่แสดงแถวที่ค้นพบค่าว่างนั้น

ในงานวิจัยนี้ ผู้วิจัยทำการวิเคราะห์เฉพาะข้อมูลที่ผ่านกระบวนการตรวจสอบความสมบูรณ์ของข้อมูลแล้วเท่านั้น สาเหตุที่ต้องทำการตรวจสอบความสมบูรณ์ของข้อมูลเพราะจากการศึกษาพบว่า นักวิทยาศาสตร์ข้อมูล (Data Scientist) ส่วนใหญ่จะใช้เวลาในการทำความสะอาดข้อมูลประมาณ 70–80% ของการทำงาน เมื่อผู้วิจัยทำการกำจัดข้อมูลที่มีความซ้ำซ้อนหรือข้อมูลที่มีบางส่วนขาดหายไปก็จะทำให้ข้อมูลที่ผู้วิจัยต้องนำไปทำการวิเคราะห์หรือใช้งานมีความน่าเชื่อถือในแง่ของการประมวลผลข้อมูล นอกจากนี้การลบข้อมูลบางส่วนที่ไม่จำเป็นต่อการนำไปวิเคราะห์ยังช่วยลดเวลาในการทำความสะอาดข้อมูล

3.3 การค้นหาคุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์

ในขั้นตอนนี้ผู้วิจัยได้ทำการค้นหาคุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ที่ปรากฏอยู่ในประโยคของข้อเสนอแนะภายใต้กระบวนการตรวจทานโค้ดในโครงการโอเพนซอร์ส โดยอาศัยเทคนิคการทำเหมืองข้อความ พร้อมทั้งทำการตรวจสอบว่าข้อเสนอแนะดังกล่าว ได้รับการแก้ไขโค้ดตามคำแนะนำเกี่ยวกับการบำรุงรักษาของผู้ตรวจทานโค้ดหรือไม่ สำหรับขั้นตอนในการค้นหาคุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์แสดงดังรูปที่ 3.3 ประกอบด้วย 5 ขั้นตอน คือ 1) สร้างคำหลัก 2) ทำความสะอาดข้อมูล 3) ค้นหาคำหลักที่ปรากฏอยู่ในข้อเสนอแนะ 4) ลบข้อเสนอแนะที่ไม่ได้รับการแก้ไขโค้ด และ 5) ค้นหาประเภทของความสามารถในการบำรุงรักษาเพิ่มเติม โดยมีรายละเอียดแต่ละขั้นตอนดังนี้



รูปที่ 3.3 ขั้นตอนในการค้นหาคุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์

1) สร้างคำหลัก (Keyword List Building)

การสร้างคำหลักที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ เช่น Modularity Analyzability Testability โดยคำหลักนี้ผู้วิจัยได้ทำการรวบรวมมาจากคุณลักษณะย่อยที่เกี่ยวข้องกับคำนิยามของ “ความสามารถในการบำรุงรักษาซอฟต์แวร์” จากวารสารวิชาการเรื่อง “Comparative Study of the Factors that Affect Maintainability” ของ Ghosh และคณะ (2011) ในงานวิจัยนี้ ผู้วิจัยจะนำคำหลักที่ได้รวบรวมมาไปใช้ในการค้นหาข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์

แต่จากการศึกษาเกี่ยวกับข้อเสนอแนะของโครงการโอเพนซอร์สพบว่า โดยปกติแล้วประโยคในข้อเสนอแนะของผู้ตรวจทานโค้ดจะไม่มีการใช้คำที่ตรงตามคำหลักที่ผู้วิจัยได้ระบุไว้ ดังนั้นผู้วิจัยจึงนำคุณลักษณะย่อยที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์มาแปลงให้อยู่ในรูปแบบรากศัพท์หรือแก่นคำ (Stemmed Word) เป็นส่วนที่แสดงถึงความหมายพื้นฐานหรือความหมายหลักของคำศัพท์ ยกตัวอย่างเช่น “modifiability” เมื่อตัด “ability” จะได้รากศัพท์คือ “modify” หลังจากนั้นจึงนำคำหลักที่อยู่ในรูปแบบรากศัพท์ดังกล่าวมาทำการจัดกลุ่มคำที่มีความหมายเหมือนหรือคล้าย (Synonyms) กับคำหลัก แต่เป็นคำที่สะกดต่างกัน ตัวอย่างเช่น “edit”, “improve”, “solve”, “amend” ซึ่งเป็นคำที่มีความหมายคล้ายกับคำหลัก “modify” โดยผู้วิจัยได้ทำการค้นหาคำที่มีความหมายเหมือนกับคำหลักนี้จากซอฟต์แวร์เวิร์ดเน็ต เว็บไซต์แหล่งข้อมูลเกี่ยวกับกลุ่มคำเหมือน[28] และพจนานุกรมภาษาอังกฤษที่เป็นฐานข้อมูลสำหรับเก็บรวบรวมคำศัพท์ภาษาอังกฤษเป็นตัวช่วยในการค้นหา เพื่อให้ได้คำศัพท์ที่มีความหมายสื่อถึงคำหลักมากที่สุดเท่าที่จะเป็นไปได้

2) ทำความสะอาดข้อมูล (Data Cleaning)

ในขั้นตอนนี้เป็นการทำความสะอาดข้อมูล เพื่อช่วยลดเวลาในการประมวลผลสำหรับการค้นหาข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์ส โดยใช้โปรแกรม R ซึ่งเป็นโปรแกรมที่ใช้ในการประมวลผลข้อมูลเชิงสถิติที่มี text mining (tm) package ในการทำความสะอาดข้อมูล ซึ่งผู้วิจัยได้ทำการลบสัญลักษณ์ที่ไม่มีความเกี่ยวข้องออกจากข้อเสนอแนะ ได้แก่ ช่องว่าง เครื่องหมายวรรคตอน ตัวเลข และอักขระที่ไม่ใช่ตัวอักษร เช่น สัญลักษณ์พิเศษ (), {, *, #, !, เพื่อเป็นการลดดัชนีของข้อมูลและลดเวลาในการประมวลผล นอกจากนี้ผู้วิจัยได้ทำการลบข้อความบางส่วนที่ไม่เป็นประโยชน์ต่อการค้นหาข้อเสนอแนะเกี่ยวกับการบำรุงรักษาและทำการแปลงคำในประโยคหรือข้อเสนอแนะของโครงการโอเพนซอร์สให้อยู่ในรูปแบบรากศัพท์ โดยรายละเอียดสามารถอธิบายได้ดังนี้

[28] http://www.synonym.com/

2.1) กำจัดคำหยุด (Stop-Word List Removal)

ข้อความในแต่ละข้อเสนอแนะจะมีคำที่ไม่จำเป็นต่อการค้นหาคำหลัก หรือมีข้อความที่ไม่มีความเกี่ยวข้องกับคุณลักษณะของความสามารถในการบำรุงรักษาซอฟต์แวร์ ผู้วิจัยจึงได้ทำการกำจัดคำหยุด (Stop-Word) โดยการตัดคำที่ไม่มีนัยสำคัญออกโดยที่ไม่ทำให้ข้อความหรือความหมายของประโยคเกิดการเปลี่ยนแปลง เช่น คำสรรพนาม คำสันธาน หรือคำบุพบทที่มักจะปรากฏอยู่ในข้อเสนอแนะทุกข้อความ ตัวอย่างเช่น a, an, the โดยคำเหล่านี้เป็นคำที่ไม่มีประโยชน์ต่อการค้นหาคุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษา หลังจากนั้นผู้วิจัยได้ทำการแปลงตัวอักษรภาษาอังกฤษที่เป็นตัวพิมพ์ใหญ่ให้เป็นตัวพิมพ์เล็ก (Upper case to Lower case) เพื่อให้สะดวกแก่การวิเคราะห์ข้อมูลที่อยู่ในรูปแบบข้อความตัวอักษร

2.2) การหารากศัพท์ (Stemming)

การลดรูปแบบคำศัพท์ในประโยคของข้อเสนอแนะให้อยู่ในรูปแก่นคำ โดยตัดคำหน้าหรืออุปสรรค (Prefix) และคำท้าย (Suffix) ของภาษาอังกฤษออก ซึ่งคำศัพท์ที่มีแก่นคำเดียวกันจะมีความหมายที่คล้ายกัน เช่น “compatible” “compatibility” เมื่อตัด “ible” กับ “ibility” จะได้รากศัพท์คือ “compat” หรือ “connection” “unconnected” “connective” และ “connecting” เมื่อทำการหารากศัพท์จะได้ว่าคำว่า “connect”

สำหรับรายละเอียดของคำสั่งในโปรแกรม R ที่ใช้ในการทำความสะอาดข้อมูลเกี่ยวกับข้อเสนอแนะของผู้ตรวจทานโค้ดทั้ง 2 โครงการ แสดงดังรูปที่ 3.4

```
1   library(tm)
2   mycorpus <- Corpus(VectorSource(Data))
3   removeURL <- function(x) gsub("http[^[:space:]]*", "", x)
4   mycorpus <- tm_map(mycorpus, removeURL)
5   mycorpus <- tm_map(mycorpus, removeNumbers)
6   mycorpus <- tm_map(mycorpus, removePunctuation)
7   mycorpus <- tm_map(mycorpus, removeWords,stopwords("english"))
8   mycorpus <- tm_map(mycorpus, tolower)
9   mycorpus <- tm_map(mycorpus, stemDocument)
10  mycorpus <- tm_map(mycorpus, stripWhitespace)

11  dataframe <- data.frame(text=sapply(mycorpus, identity),
                            stringsAsFactors=F)

12  write.csv(dataframe, file = "D:/Data_Clean.csv")
```

รูปที่ 3.4 ชุดคำสั่ง R ที่ได้ทำการพัฒนาสำหรับใช้ในการทำความสะอาดข้อมูล

จากรูปที่ 3.4 ประกอบด้วยคำสั่งดังรายละเอียดต่อไปนี้

บรรทัดที่ 1 : เป็นการเรียกใช้งาน package tm

บรรทัดที่ 2 : เป็นการสร้างคลังเอกสารสำหรับใช้ในกระบวนการทำความสะอาดข้อมูล โดย “`VectorSource(Data)`” เป็นฟังก์ชันที่กำหนดให้ข้อมูลอยู่ในรูปแบบอักขระ (Character Vectors) ซึ่งเป็นหน่วยของคำในข้อมูลที่มีความหมายเข้าใจได้ง่ายและเป็นลักษณะอักขระที่ถูกเขียนในรูปแบบของภาษาธรรมชาติ ได้แก่ ตัวเลข (Numeric) ตัวอักษร (Text) และเครื่องหมายวรรคตอน รวมถึงช่องว่างในข้อความ จากนั้นก็จะทำการจัดเก็บชุดข้อความดังกล่าวในฟังก์ชัน “`Corpus`” ซึ่งเป็นฟังก์ชันที่ใช้ในการจัดเก็บคลังเอกสารอยู่ในตัวแปรที่มีชื่อว่า “`mycorpus`” โดยเอกสารที่ใช้ในกระบวนการนี้จะมีข้อมูลเกี่ยวกับข้อเสนอแนะของผู้ตรวจทานโค้ด

บรรทัดที่ 3-4 : เป็นคำสั่งที่ใช้ในกระบวนการทำความสะอาดข้อมูล โดยประกาศใช้ “`tm_map`” ซึ่งเป็นฟังก์ชันที่ใช้ในการรับและส่งคลังเอกสาร โดยมีเงื่อนไขในการเรียกใช้ “`removeURL`” เป็นตัวแปรที่จัดเก็บฟังก์ชันที่ใช้ในการลบ Uniform Resource Locator (URL) และสัญลักษณ์พิเศษต่าง ๆ ที่ปรากฏในคลังเอกสาร จากนั้นจะเป็นการจัดเก็บผลลัพธ์ของคลังเอกสารไปยังตัวแปรที่มีชื่อว่า “`mycorpus`”

บรรทัดที่ 5-10 : เป็นคำสั่งที่ใช้ในการเรียกฟังก์ชัน “`tm_map`” ที่มีเงื่อนไขในการทำความสะอาดคลังเอกสาร โดยการเรียกใช้ตัวแปรที่มีอยู่ใน package tm ซึ่งตัวแปรที่สามารถอ้างอิงถึงฟังก์ชันที่ใช้ในการทำความสะอาดข้อมูลจะมีด้วยกันดังนี้

- `removeNumbers` เป็นฟังก์ชันที่ใช้ในการลบตัวเลข
- `removePunctuation` เป็นฟังก์ชันที่ใช้ในการลบเครื่องหมายวรรคตอน
- `removeWords` และ `stopwords("english")` เป็นฟังก์ชันที่ใช้ในการลบคำที่ไม่มีนัยสำคัญออกจากข้อความหรือที่เรียกว่า “คำหยุด” โดยมีเงื่อนไขให้ลบคำหยุดที่เป็นมาตรฐานของภาษาอังกฤษ
- `tolower` เป็นฟังก์ชันที่ใช้ในการแปลงตัวอักษรให้เป็นตัวพิมพ์เล็ก
- `stemDocument` เป็นฟังก์ชันที่ใช้ในการแปลงคำศัพท์ในคลังเอกสารให้อยู่ในรูปแบบรากศัพท์
- `stripWhitespace` เป็นฟังก์ชันที่ใช้ในการลบช่องว่าง

เมื่อผ่านกระบวนการทำความสะอาด โดยมีเงื่อนไขในการใช้ฟังก์ชันที่ได้ระบุไว้แล้ว โปรแกรมก็จะทำการจัดเก็บผลลัพธ์ของคลังเอกสารในตัวแปรที่มีชื่อว่า “`mycorpus`”

บรรทัดที่ 11 : เป็นการใช้คำสั่ง “`data.frame`” ซึ่งเป็นฟังก์ชันที่ใช้ในการแปลงผลลัพธ์ให้อยู่ในรูปแบบเดต้าเฟรม (Data Frame) หรือก็คือเป็นการเก็บข้อมูลที่มีลักษณะโครงสร้างเป็นตาราง โดยมีแถวและคอลัมน์ โดยการแปลงข้อมูลให้เก็บอยู่ในรูปแบบของเดต้าเฟรมจะช่วยให้สามารถ

ส่งออกไฟล์ข้อมูลเพื่อนำไปใช้ในการวิเคราะห์ต่อได้ง่ายมากขึ้น

บรรทัดที่ 12 : เป็นการเรียกใช้ฟังก์ชัน “write.csv” ในการส่งออกไฟล์ข้อมูลให้อยู่ในรูปแบบ Comma Separated Values (CSV) โดยสาเหตุที่ผู้วิจัยทำการส่งออกไฟล์ CSV เพราะเป็นไฟล์ข้อความประเภทหนึ่งที่ใช้สำหรับเก็บข้อมูลในรูปแบบตารางและการบันทึกไฟล์ที่ได้มีขนาดเล็กมาก นอกจากนี้ยังใช้สำหรับการเชื่อมโยงข้อมูลต่าง ๆ หรือนำเข้าระบบฐานข้อมูล เพื่อทำการประมวลผลข้อมูล

3) ค้นหาคำหลักที่ปรากฏอยู่ในข้อเสนอแนะ (Comment Retrieval)

สำหรับขั้นตอนนี้จะเป็นการพัฒนาชุดคำสั่ง เพื่อทำการค้นหาข้อเสนอแนะที่มีการกล่าวถึงคำหลักหรือคำที่มีความหมายสื่อถึงคำหลักเกี่ยวกับความสามารถในการบำรุงรักษาซอฟต์แวร์อย่างน้อย 1 คำ โดยใช้คำที่ได้จากการสืบค้นในขั้นตอนที่ 1) การสร้างคำหลัก พร้อมทั้งทำการแปลงผลลัพธ์ของการค้นหาให้อยู่ในรูปแบบของไฟล์เอกสารเพื่อให้ง่ายต่อการนำไปวิเคราะห์ต่อ

ในการค้นหาข้อมูลดังกล่าว ผู้วิจัยได้ทำการพัฒนาคำสั่งในโปรแกรม R โดยเชื่อมต่อกับฐานข้อมูล MySQL และพัฒนาคำสั่งในการค้นหาออกเป็นชุดคำสั่งย่อย ตัวอย่างเช่น ผู้วิจัยได้ทำการสร้างชุดคำสั่งเพื่อค้นหาคำว่า “modify” และคำที่มีความหมายเหมือนกับ “modify” ที่ปรากฏในข้อความของข้อเสนอแนะที่ถูกจัดเก็บในระบบฐานข้อมูล แสดงดังรูปที่ 3.5 ชุดคำสั่งในโปรแกรม R สำหรับการค้นหาคำหลัก “modify” ที่ปรากฏอยู่ในข้อเสนอแนะ

```
1  library(RMySQL)
2  library(sqldf)
3  comment <- sqldf("select * from Data_Clean - Eclipse
4                  where message LIKE '%modify%'
5                     OR massage LIKE '%edit%'
6                     OR massage LIKE '%improve%'
7                     OR massage LIKE '%solve%' ")

8  if(!nrow(comment) == 0){
9     comment["Keywords"] <- "Modifiability"
10    write.csv(comment, file = "D:/Eclipse - modifiability.csv")

11 }else { print("Not comment") }
```

รูปที่ 3.5 ชุดคำสั่ง R ที่ได้ทำการพัฒนาสำหรับใช้ในการค้นหาคำหลักที่ปรากฏในข้อเสนอแนะ

สำหรับหลักการทำงานของชุดคำสั่งในรูปที่ 3.5 สามารถอธิบายรายละเอียดของคำสั่งที่ใช้ในแต่ละบรรทัดดังนี้

บรรทัดที่ 1-2 : เป็นการเรียกใช้งาน package RMySQL และ package sqldf ที่ใช้ในการเชื่อมต่อกับฐานข้อมูล MySQL และสามารถใช้คำสั่ง SQL ในการประมวลผลข้อมูล

บรรทัดที่ 3-7 : เป็นการเรียกใช้ฟังก์ชัน “`sqldf`” เพื่อทำการค้นหาคำหลักหรือคำที่สื่อถึงคำหลักที่ปรากฏในประโยคของข้อเสนอแนะจากฐานข้อมูล โดยการค้นหาด้วยคำสั่ง SQL ที่มี “`Data_Clean - Eclipse`” เป็นข้อมูลในรูปแบบตารางที่ได้จากการรวบรวมคำแนะนำที่ได้จากระบบเกอริตและผ่านกระบวนการทำความสะอาดข้อมูล โดยมีเงื่อนไขในการระบุคำสำหรับค้นหาใน “`message`” ซึ่งเป็นคอลัมน์ที่อยู่ภายในตารางข้อมูลที่ใช้ในการค้นหา

บรรทัดที่ 8 : เป็นการระบุเงื่อนไข if-else ที่เป็นคำสั่งแบบเลือกทำ 2 ทาง โดยในที่นี้ได้ระบุ if ในการตรวจสอบข้อมูลที่ได้จากการค้นหา

บรรทัดที่ 9 : เป็นคำสั่งที่ใช้ในการเพิ่มคอลัมน์ในตารางของผลลัพธ์ที่ได้จากการค้นหาโดยมีการระบุคำหลักที่ใช้สำหรับการค้นหา

บรรทัดที่ 10 : เป็นการเรียกใช้ฟังก์ชัน “`write.csv`” เพื่อส่งออกไฟล์ข้อมูลในรูปแบบ CSV

บรรทัดที่ 11 : เป็นการระบุเงื่อนไข else ที่ใช้ในการบ่งบอกว่าไม่มีผลลัพธ์จากการค้นหาข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษา

ในการค้นหาข้อเสนอแนะที่มีคำหลักปรากฏอยู่ ผู้วิจัยได้ทำการสร้างชุดคำสั่ง 1 คำสั่งต่อการค้นหาคำหลัก 1 คำ เพื่อให้สะดวกแก่การสืบค้นและบันทึกผล รวมถึงช่วยลดเวลาการประมวลผลของการสืบค้นข้อมูลในแต่ละครั้ง เนื่องจากโครงการโอเพนซอร์สทั้ง 2 โครงการ มีข้อมูลเกี่ยวกับข้อเสนอแนะเป็นจำนวนมาก และเมื่อผู้วิจัยทำการสืบค้นข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์เรียบร้อยแล้ว ผู้วิจัยได้ทำการตรวจสอบข้อเสนอแนะที่ได้จากการค้นหา เพื่อให้ข้อมูลที่จะนำไปทำการวิเคราะห์ต่อนั้นมีความน่าเชื่อถือเพิ่มมากขึ้น โดยใช้วิธีการตรวจสอบด้วยตัวผู้วิจัยเอง (Manual) หรือก็คือการอ่านข้อเสนอแนะเพื่อดูว่า ข้อเสนอแนะนั้นมีการระบุคำหลักหรือคำที่มีความหมายสื่อถึงคำหลักที่ได้กำหนดไว้จริงหรือไม่ ซึ่งหากพบว่าข้อความในประโยคของข้อเสนอแนะมีคำที่ไม่ตรงตามที่ได้กำหนดไว้ ข้อความนั้นก็จะถูกลบออกจากระบบฐานข้อมูล

หลังจากที่ผู้วิจัยทำการตรวจสอบผลลัพธ์ของการค้นหาข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์เรียบร้อยแล้ว ผู้วิจัยจึงนำผลลัพธ์ที่ได้จากการค้นหามาจัดเก็บข้อมูลในรูปแบบที่เป็นตาราง หรือเรียกว่า รีเลชัน (Relation) ซึ่งมีลักษณะเป็นตารางที่มีแถวและสดมภ์แสดงดังตัวอย่างในตารางที่ 3.1 โดยสดมภ์จะมีการระบุคำหลัก เช่น คำว่า Module Reuse Anlayze Modify และ Test

เป็นคำหลักที่อยู่ในรูปแบบของแก่นคำหรือรากศัพท์ และแถวจะมี Eclipse เป็นชื่อของโครงการซอฟต์แวร์โอเพนซอร์ส พร้อมทั้งแสดงช่วงระยะเวลา 5 ปีที่ได้ทำการวิเคราะห์ข้อมูล สำหรับข้อมูลในตารางจะเป็นการแสดงจำนวนความถี่ของการปรากฏคำหลัก (Term Frequency) หรือคำที่สื่อถึงคำหลักที่พบในประโยคข้อความของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษา

ตารางที่ 3.1 ตัวอย่างตารางการเก็บข้อมูลที่ได้จากการค้นหาข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาของโครงการ Eclipse ในช่วงระยะเวลา 5 ปี

Keyword	Eclipse				
	2012	2013	2014	2015	2016
Module	4	0	2	1	3
Reuse	1	3	0	0	1
Analyze	3	1	1	3	1
Modify	2	1	3	4	2
Test	1	0	0	2	0

4) ลบข้อเสนอแนะที่ไม่ได้รับการแก้ไขโค้ด (Unrelated Comment Removal)

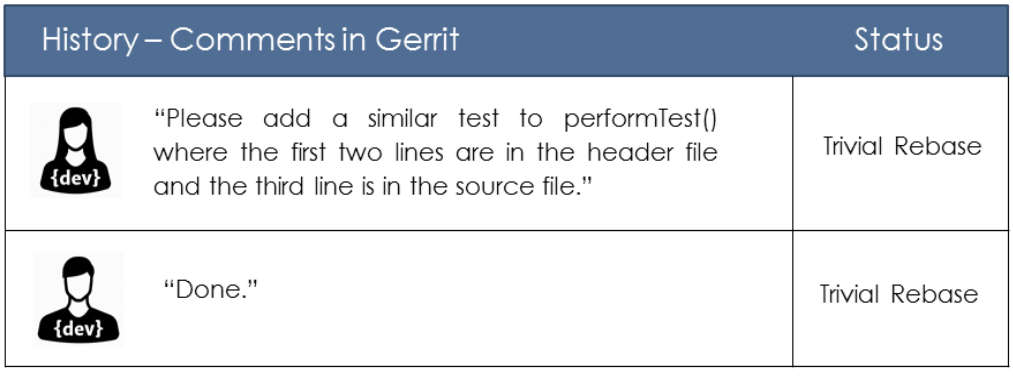
สำหรับขั้นตอนนี้ ผู้วิจัยจะนำผลลัพธ์ที่ได้ทำการค้นหาจากขั้นตอนที่ 3 ทำการวิเคราะห์ว่า ในช่วงปี ค.ศ. 2012 – 2016 โครงการ Eclipse และโครงการ Qt มีจำนวนข้อเสนอแนะที่ได้รับการแก้ไขโค้ดเกี่ยวกับความสามารถในการบำรุงรักษามากน้อยแค่ไหน โดยใช้วิธีการตรวจสอบด้วยการอ่านและวิเคราะห์ด้วยตัวผู้วิจัยเอง ด้วยวิธีการนี้จะทำให้ผู้วิจัยทราบว่า นักพัฒนาในโครงการโอเพนซอร์สให้ความสนใจกับการบำรุงรักษาซอฟต์แวร์ภายใต้กระบวนการตรวจทานโค้ดหรือไม่

การตรวจสอบข้อเสนอแนะที่ได้รับการแก้ไขโค้ดนั้น ผู้วิจัยได้ทำการพัฒนาชุดคำสั่งในโปรแกรม R โดยทำการจับคู่ข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ที่ได้จากการค้นหาในขั้นตอนที่ 3 กับข้อเสนอแนะที่ได้รวบรวมจากระบบเกอริต เพื่อทำการตรวจสอบการสนทนาโต้ตอบระหว่างผู้ตรวจทานโค้ดและนักพัฒนา เนื่องจากข้อเสนอแนะที่ได้รวบรวมมาจากระบบเกอริตนั้นไม่ได้มีเพียงแค่คำแนะนำจากผู้ตรวจทานโค้ด แต่ยังมีข้อเสนอแนะของนักพัฒนาที่แสดงความคิดเห็นเพิ่มเติมหรือการตอบรับการแก้ไขโค้ดตามคำแนะนำที่ได้รับ ซึ่งชุดคำสั่งที่ใช้ในการจับคู่ข้อเสนอแนะที่ได้จากการค้นหาและการตอบกลับของนักพัฒนาจะใช้คำสั่งดังนี้

```
result <- merge(DataEclipse, CommentEclipse,
            by.x = c("link_id", "line", "revision", "file"),
            by.y = c("link_id", "line", "revision", "file"),
            all.y = TRUE)
```

ฟังก์ชัน “merge” เป็นการรวมข้อมูลที่ต้องการจากฐานข้อมูล MySQL ในงานวิจัยนี้ ผู้วิจัยได้ทำการรวมตารางข้อมูลที่เก็บรวบรวมข้อเสนอแนะของผู้ตรวจทานโค้ดกับตารางข้อมูลที่เก็บรวบรวมผลลัพธ์ที่ได้จากการค้นหาข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ โดยมีเงื่อนไขที่ได้ระบุไว้ คือ ข้อมูลของทั้ง 2 ตารางที่จับคู่กันจะต้องเป็นข้อมูลที่มี “link_id”, “line”, “revision”, “file” เหมือนกันเท่านั้น

เมื่อทำการจับคู่ข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์และการตอบกลับของนักพัฒนาแล้ว ผู้วิจัยต้องทำการพิจารณาว่า มีข้อเสนอแนะใดบ้างที่ได้รับการแก้ไขโค้ดจากนักพัฒนา โดยข้อเสนอแนะที่ได้รับการแก้ไขโค้ดที่ได้ระบุไว้ คือ ข้อเสนอแนะที่ได้รับจากการตรวจทานโค้ดจากนักพัฒนาท่านอื่น และนักพัฒนาที่เป็นผู้เขียนโค้ดได้ทำการแก้ไขโค้ดตามคำแนะนำที่ได้รับจากผู้ตรวจทานโค้ด ส่วนข้อเสนอแนะที่ไม่ได้รับการแก้ไขโค้ด คือ ข้อเสนอแนะที่ได้รับจากการตรวจทานโค้ดแล้วเช่นกัน แต่นักพัฒนาไม่ได้ทำการแก้ไขโค้ดตามข้อเสนอแนะดังกล่าว สำหรับการตรวจสอบข้อเสนอแนะที่ได้รับการแก้ไขโค้ด เป็นพิจารณาจากสถานะของการแก้ไขโค้ด เช่น การแก้ไขเพียงเล็กน้อย (Trivial Rebase) หรือ การแก้ไขปรับปรุงใหม่ (Rework) พร้อมทั้งพิจารณาคำศัพท์หรือประโยคในการตอบรับการแก้ไขโค้ดของนักพัฒนา เช่น “Done“ ”Finished“ ”Edited” โดยตัวอย่างข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์และการตอบรับการแก้ไขโค้ดของนักพัฒนาแสดงดังรูปที่ 3.6



รูปที่ 3.6 ตัวอย่างข้อเสนอแนะที่ได้รับการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาซอฟต์แวร์

หลังจากทำการตรวจสอบข้อเสนอแนะตามเกณฑ์การพิจารณาข้างต้นแล้ว ผู้วิจัยจึงทำการลบข้อเสนอแนะที่ไม่ได้รับการตอบรับจากนักพัฒนาซอฟต์แวร์หรือก็คือข้อเสนอแนะที่ไม่ได้รับการแก้ไขโค้ดออกจากระบบฐานข้อมูล ส่วนข้อเสนอแนะที่ได้รับการตรวสอบหรือผลลัพธ์ที่ได้จากขั้นตอนนี้จะถูกจัดเก็บในรูปแบบตารางเช่นเดียวกับตัวอย่างตารางที่ 3.1 แต่ผลลัพธ์ที่แสดงในตารางจะเป็นผลของจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาที่ได้รับการแก้ไขโค้ดจากนักพัฒนาในช่วงระยะเวลา 5 ปี โดยข้อมูลที่ได้จากการวิเคราะห์แสดงให้เห็นว่านักพัฒนาของโครงการ Eclipse และโครงการ Qt ให้ความสำคัญกับการบำรุงรักษาซอฟต์แวร์มากน้อยแค่ไหน ในช่วงปี ค.ศ. 2012 – 2016

5) ค้นหาประเภทของความสามารถในการบำรุงรักษาเพิ่มเติม (Maintainability Attributes Discovery)

เมื่อผู้วิจัยทำการค้นหาประเภทของความสามารถในการบำรุงรักษาทั้ง 33 ประเภทแล้ว ผู้วิจัยจึงได้ทำการค้นหาคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาประเภทอื่น ๆ ที่ซ่อนอยู่ในข้อเสนอแนะของโครงการ Eclipse และโครงการ Qt โดยอาศัยอัลกอรึทึม LDA ซึ่งเป็นวิธีในการค้นหาคำที่มีส่วนเกี่ยวข้องกับคำหลักที่ผู้วิจัยได้ระบุไว้ ซึ่งผลที่ได้จากกระบวนการใช้อัลกอริทึม LDA ในโปรแกรม R จะแสดงให้เห็นถึงความถี่ของคำที่มักจะพบร่วมกับคำหลัก และทางผู้วิจัยจะต้องทำการพิจารณาว่า คำที่ค้นพบนั้นควรจะนำมาเป็นคำหลักหรือไม่ หากพบว่าควรนำคำเหล่านี้มาเพิ่มเป็นคำหลักก็จะดำเนินการในขั้นตอนที่ 3 และ 4 ซ้ำอีกครั้งจนกว่าจะไม่ค้นพบคำหลักประเภทใหม่เพิ่มเติมอีก

สำหรับผลลัพธ์ที่ได้จากขั้นตอนที่ 3 ถึง 5 ผู้วิจัยได้ให้นักพัฒนาซอฟต์แวร์ที่ผู้วิจัยเคยร่วมงานด้วย ซึ่งมีประสบการณ์เกี่ยวกับการตรวจทานโค้ดทำการตรวจสอบผลลัพธ์ โดยการอ่านและวิเคราะห์ข้อเสนอแนะประมาณ 30% ของจำนวนทั้งหมด เพื่อตรวจสอบว่าผลลัพธ์ที่ได้มีความถูกต้องจริง

3.4 การวิเคราะห์ข้อมูล

ในการวิเคราะห์ข้อมูลเพื่อศึกษาการตรวจทานโค้ดที่มีต่อความสามารถในการบำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์ส ผู้วิจัยได้ทำการจัดกลุ่มประเภทของบำรุงรักษาซอฟต์แวร์และค้นหาคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาประเภทใหม่ที่ปรากฏอยู่ในข้อเสนอแนะของผู้ตรวจทานโค้ดโดยใช้อัลกอริทึม LDA จากนั้นผู้วิจัยได้นำคุณลักษณะที่ได้รวบรวมมาทำการค้นหาข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาด้วยวิธีการทำเหมืองข้อความ และตรวจสอบข้อเสนอแนะที่นักพัฒนาได้ทำการแก้ไขโค้ด

เกี่ยวกับการบำรุงรักษาซอฟต์แวร์ในช่วงปี ค.ศ. 2012 – 2016 ของโครงการโอเพนซอร์ส โดยผลลัพธ์ที่ได้จากดำเนินการนี้จะเป็นการตอบคำถามวิจัย 3 ข้อ ดังนี้

1) คุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์สแบ่งออกเป็นกี่คุณลักษณะ
2) นักพัฒนาซอฟต์แวร์ให้ความสนใจกับความสามารถในการบำรุงรักษาซอฟต์แวร์ภายใต้การตรวจทานโค้ดในโครงการโอเพนซอร์สหรือไม่
3) คุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ประเภทใดที่นักพัฒนาให้ความสนใจเป็นพิเศษ

ผลลัพธ์ที่ได้จากการวิเคราะห์แสดงให้เห็นถึงอัตราร้อยละของจำนวนข้อเสนอแนะที่เกี่ยวกับทางด้านความสามารถในการบำรุงรักษาซอฟต์แวร์ของโครงการ Eclipse และโครงการ Qt ในช่วงระยะเวลา 5 ปีว่า มีปริมาณข้อเสนอแนะที่ได้รับการแก้ไขโค้ดมาก-น้อยแค่ไหน เมื่อเทียบกับจำนวนข้อเสนอแนะทั้งหมด นอกจากนี้ผู้วิจัยยังใช้การวิเคราะห์การถดถอย (Regression Analysis) มาช่วยในการวิเคราะห์ข้อมูล เพื่อพยากรณ์แนวโน้มของการการแก้ไขโค้ดตามข้อเสนอแนะเกี่ยวกับความสามารถในการบำรุงรักษาซอฟต์แวร์ที่มีอัตราที่เพิ่มสูงขึ้น-ลดลงในแต่ละปี พร้อมทั้งทำการตรวจสอบว่า นักพัฒนาในโครงการโอเพนซอร์สทำการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ประเภทใดมากเป็นพิเศษ โดยผลจากการศึกษานี้จะแสดงให้เห็นถึงความสนใจของนักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์ที่จะช่วยลดปัญหาด้านคุณภาพของซอฟต์แวร์ที่มีอยู่และช่วยในการปรับปรุงคุณภาพซอฟต์แวร์ให้ดียิ่งขึ้น

นอกจากการตอบคำถามวิจัยหลักข้างต้นแล้ว ทางผู้วิจัยได้นำผลของข้อมูลที่ได้จากการวิเคราะห์มาทำการศึกษาเพื่อเพิ่มหลักฐานเชิงประจักษ์เกี่ยวกับปัจจัยที่มีอิทธิพลต่อการตรวจทานโค้ดทางด้านการบำรุงรักษาซอฟต์แวร์ในระบบเกอริต โดยผู้วิจัยได้ทำการระบุปัจจัยด้วยกัน 2 ประเภท ได้แก่ 1) บทบาทและขนาดกลุ่มของนักพัฒนาซอฟต์แวร์ และ 2) เวลาในการตรวจทานโค้ดของนักพัฒนาซอฟต์แวร์ การเลือกศึกษาปัจจัยเหล่านี้เพราะได้รับแรงจูงใจจากการศึกษาก่อนหน้าเกี่ยวกับการตรวจทานโค้ดหรือการตรวจทานเชิงปฏิบัติ (Peer Review Practices) (Rigby, et al., 2008) ในโครงการโอเพนซอร์ส โดยเฉพาะวิธีการตรวจสอบซอฟต์แวร์แบบละเอียด (Software Inspection) (Rigby and Bird, 2013) ซึ่งเป็นวิธีการตรวจทานโค้ดดั้งเดิมของโครงการขนาดใหญ่ในโครงการโอเพนซอร์ส นั่นคือ โครงการเซิร์ฟเวอร์อาปาเช่

เพื่อตรวจสอบอิทธิพลของแต่ละปัจจัยเกี่ยวกับการตรวจทานโค้ดในโครงการ Eclipse และโครงการ Qt ผู้วิจัยจึงได้ทำการกำหนดคำถามวิจัยที่จะทำการศึกษาเพิ่มเติมดังนี้

1) ขนาดกลุ่มของผู้ตรวจทานโค้ดส่งผลต่อจำนวนข้อเสนอแนะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์หรือไม่
2) ผู้ตรวจทานโค้ดทำงานเกี่ยวกับการตรวจทานโค้ดในโครงการโอเพนซอร์สมากน้อยเพียงใดต่อสัปดาห์

เพื่อตอบคำถามวิจัยที่ได้ระบุไว้ข้างต้น ผู้วิจัยจึงได้ทำการตรวจสอบขนาดกลุ่มของนักพัฒนาซอฟต์แวร์เป็นรายเดือน เนื่องจากข้อเสนอแนะที่ได้รับจากผู้ตรวจทานโค้ดในแต่ละเดือน มักจะเป็นคำแนะนำที่ได้รับมาจากผู้ตรวจทานโค้ดคนเดิมหรือก็คือนักพัฒนาซอฟต์แวร์หลักที่ปฏิบัติงานประจำอยู่ในโครงการเป็นระยะเวลานาน นอกจากนี้ผู้วิจัยยังใช้สถิติสหสัมพันธ์ (Correlation) ในการวิเคราะห์ความสัมพันธ์ระหว่างขนาดกลุ่มของนักพัฒนาซอฟต์แวร์ ซึ่งมีหน้าที่ในการตรวจทานโค้ดกับข้อเสนอแนะเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ที่ได้จากการตรวจทานโค้ด โดยผลลัพธ์จากการศึกษาจะเป็นการตรวจสอบว่า ขนาดกลุ่มของนักพัฒนาซอฟต์แวร์เป็นปัจจัยหนึ่งที่ส่งผลต่อจำนวนข้อเสนอแนะเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ในช่วงระยะเวลา 5 ปีหรือไม่ สำหรับการตรวจสอบเกี่ยวกับการทำงานของนักพัฒนาซอฟต์แวร์ภายใต้กระบวนการตรวจทานโค้ด ผู้วิจัยได้ใช้สถิติการวิเคราะห์ความแปรปรวน (ANOVA หรือ Analysis of Variance) และ Post-hoc Comparisons แบบ Tukey's Honestly Significant Difference (HSD) ในการวิเคราะห์ว่า ค่าเฉลี่ยของแต่ละวันในช่วงสัปดาห์ที่ผู้ตรวจทานโค้ดให้คำแนะนำเกี่ยวกับการบำรุงรักษามีความแตกต่างกันหรือไม่ เพื่อเป็นหลักฐานที่แสดงให้เห็นว่านักพัฒนาซอฟต์แวร์ที่ทำงานด้วยความสมัครใจในโครงการโอเพนซอร์สได้ให้ความสำคัญกับการตรวจทานโค้ด โดยเฉพาะทางด้านการบำรุงรักษาซอฟต์แวร์มากน้อยแค่ไหนในช่วงสัปดาห์

3.5 จัดทำรายงานผลการวิจัย

การวิจัยในครั้งนี้ผู้วิจัยได้ทำการวิเคราะห์ข้อมูลเชิงปริมาณ (Quantitative) โดยใช้เทคนิคการทำเหมืองข้อความในการค้นหาและจัดกลุ่มประเภทของความสามารถในการบำรุงรักษาจากโครงการซอฟต์แวร์โอเพนซอร์สจำนวน 2 โครงการ หลังจากที่ได้ทำการวิเคราะห์ผลลัพธ์ของข้อมูล

เรียบร้อยแล้ว ผู้วิจัยได้นำผลลัพธ์จากการวิเคราะห์มาทำสรุปผลของงานวิจัยในรูปแบบวิทยานิพนธ์ บทความวิชาการและวารสารวิชาการ เพื่อเป็นหลักฐานเชิงประจักษ์เกี่ยวกับคุณภาพของซอฟต์แวร์ในโครงการโอเพนซอร์ส และเป็นแนวทางในการปรับปรุงคุณภาพของซอฟต์แวร์ให้แก่นักพัฒนาซอฟต์แวร์รุ่นใหม่ที่ต้องการเข้ามามีส่วนร่วมในการพัฒนาโครงการ

บทที่ 4

ผลการวิจัย

ในงานวิจัยนี้ ผู้วิจัยได้ทำการวิเคราะห์ข้อมูลในโครงการซอฟต์แวร์โอเพนซอร์สจำนวน 2 โครงการ โดยผู้วิจัยทำการศึกษาข้อมูลเกี่ยวกับข้อเสนอแนะภายใต้กระบวนการตรวจทานโค้ดของโครงการ Eclipse และโครงการ Qt ที่ได้รวบรวมจากระบบเกอริตในช่วงระยะเวลา 5 ปี นั่นคือ ตั้งแต่ปี ค.ศ. 2012 - 2016 เพื่อมีวัตถุประสงค์ที่จะตอบคำถามวิจัยดังนี้

1) คุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์สแบ่งออกเป็นกี่คุณลักษณะ
2) นักพัฒนาซอฟต์แวร์ให้ความสนใจกับความสามารถในการบำรุงรักษาซอฟต์แวร์ภายใต้การตรวจทานโค้ดในโครงการโอเพนซอร์สหรือไม่
3) คุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ประเภทใดที่นักพัฒนาให้ความสนใจเป็นพิเศษ
4) ขนาดกลุ่มของผู้ตรวจทานโค้ดส่งผลต่อจำนวนข้อเสนอแนะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์หรือไม่
5) ผู้ตรวจทานโค้ดทำงานเกี่ยวกับการตรวจทานโค้ดในโครงการโอเพนซอร์สมากน้อยเพียงใดต่อสัปดาห์

เพื่อตอบคำถามของงานวิจัยที่ได้ระบุไว้ข้างต้น ผู้วิจัยจะนำเสนอผลการวิเคราะห์ข้อมูลด้วยเทคนิคการทำเหมืองข้อความ ซึ่งผู้วิจัยได้ทำการรวบรวมคุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ เพื่อนำมาใช้ในการค้นหาข้อเสนอแนะของผู้ตรวจทานโค้ดในโครงการ Eclipse และโครงการ Qt ที่มีการแนะนำให้นักพัฒนาซอฟต์แวร์ทำการแก้ไขปรับปรุงโค้ดเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ พร้อมทั้งทำการตรวจสอบว่า นักพัฒนาซอฟต์แวร์ได้ทำการแก้ไขซอร์สโค้ดตามคำแนะนำดังกล่าวหรือไม่ นอกจากนี้ผู้วิจัยยังนำผลที่ได้จากการดำเนินงานข้างต้นมาทำการศึกษาต่อ โดยตรวจสอบเกี่ยวกับปัจจัยที่มีอิทธิพลต่อการตรวจทานโค้ดทางด้านการบำรุงรักษาซอฟต์แวร์ ผู้วิจัยจะนำเสนอผลการวิจัยตามคำถามวิจัยในแต่ละข้อตามลำดับ โดยมีรายละเอียดดังนี้

4.1 คุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์สแบ่งออกเป็นกี่คุณลักษณะ

ในงานวิจัยนี้ผู้วิจัยได้นำคุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ทั้ง 33 คุณลักษณะ มาเป็นคำหลักเพื่อใช้ในการค้นหาข้อเสนอแนะของผู้ตรวจทานโค้ดภายในโครงการโอเพนซอร์ส ซึ่งคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ทั้ง 33 คุณลักษณะนี้เป็นลักษณะของการแก้ไขปรับปรุงโค้ดที่จะช่วยลดความเสี่ยงของการเกิดข้อบกพร่อง ข้อผิดพลาดหรือปัญหาต่าง ๆ ที่อาจส่งผลกระทบในอนาคตได้ รวมถึงยังช่วยให้แนวโน้มของคุณภาพทางด้านซอฟต์แวร์เพิ่มสูงขึ้น นอกจากนี้ผู้วิจัยยังได้ทำการสืบค้นและรวบรวมคำที่มีความหมายสื่อถึงคำหลัก เพื่อให้ครอบคลุมถึงคำนิยามของความสามารถในการบำรุงรักษาซอฟต์แวร์มากที่สุดเท่าที่จะเป็นไปได้ ก่อนที่จะนำคำเหล่านี้ไปทำการค้นหาข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษา

4.1.1 การกำหนดคำหลัก

จากการศึกษาวรรณกรรมที่เกี่ยวข้องกับความสามารถในการบำรุงรักษา พบว่า ในงานวิจัยทางด้านวิศวกรรมซอฟต์แวร์ เรื่อง “Comparative Study of the Factors that Affect Maintainability” ซึ่งเป็นวารสารวิชาการของ Ghosh และคณะ (2011) ได้มีการศึกษาและรวบรวมคุณลักษณะย่อยที่มีความเกี่ยวข้องกับความสามารถในการบำรุงรักษาไว้ทั้งหมด 40 คุณลักษณะ โดยคุณลักษณะย่อยทั้งหมดเป็นคุณลักษณะที่นำมาจากมาตรฐานสากลทางด้านคุณภาพซอฟต์แวร์และวรรณกรรมต่าง ๆ ที่เกี่ยวข้องกับคำนิยามของความสามารถในการบำรุงรักษา ซึ่งคุณลักษณะย่อยดังกล่าวมีด้วยกันดังนี้

1) Accuracy
2) Adaptability
3) Analyzability
4) Augmentability
5) Availability
6) Changeability
7) Completeness
8) Complexity
9) Compliance
10) Comprehensibility
11) Conciseness
12) Consistency
13) Correctability
14) Delivery
15) Documentation
16) Durability

17) Efficiency
18) Effort
19) Expandability
20) Extensibility
21) Flexibility
22) Impact analysis
23) Implementation
24) Instrumentation
25) Integrability
26) Localization
27) Modifiability
28) Modularity
29) Perfectiveness
30) Portability
31) Programming Language
32) Readability
33) Reusability
34) Self-descriptiveness
35) Simplicity
36) Stability
37) Standardization
38) Testability
39) Traceability
40) Understandability

จากวารสารวิชาการของ Ghosh และคณะ (2011) ผู้วิจัยได้ทำการตรวจสอบคำนิยามของแต่ละคุณลักษณะย่อยทั้ง 40 ประเภท และกำหนดเกณฑ์การพิจารณาในการเลือกคุณลักษณะ 2 ประการ คือ 1) คำนิยามของคุณลักษณะย่อยจะต้องมีความเกี่ยวข้องกับคำนิยามของ “ความสามารถในการบำรุงรักษาซอฟต์แวร์” และ 2) ไม่เป็นคำศัพท์สามัญหรือคำศัพท์กลาง ๆ ที่พบได้ทั่วไป โดยคุณลักษณะที่ไม่ผ่านเกณฑ์ที่ได้พิจารณามีด้วยกัน 7 คำ ดังนี้

1) Compliance
2) Conciseness
3) Delivery
4) Documentation
5) Impact Analysis
6) Programming Language
7) Self-descriptiveness

เมื่อผู้วิจัยได้คุณลักษณะย่อยที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ที่ผ่านการพิจารณาแล้ว ผู้วิจัยจึงทำการแปลงคุณลักษณะย่อยดังกล่าวให้อยู่ในรูปกิริยาหรือคำนาม โดยการนำคุณลักษณะทั้ง 33 ประเภท ที่เป็นคำคุณศัพท์มาทำการตัดคำท้าย (Suffix) ออก เพื่อให้ได้คำหลักที่ง่ายต่อการนำไปใช้สำหรับการค้นหาข้อเสนอแนะของผู้ตรวจทานโค้ดในโครงการโอเพนซอร์ส ซึ่งผลลัพธ์จากการกำหนดคำหลักนี้แสดงดังตารางที่ 4.1

ตารางที่ 4.1 คำหลักที่ใช้ในการค้นหาข้อเสนอแนะของผู้ตรวจทานโค้ด

ลำดับ	คุณลักษณะของความสามารถในการบำรุงรักษา	คำหลัก
1	Accuracy	Accuracy
2	Adaptability	Adapt
3	Analyzability	Analyze
4	Augmentability	Augment
5	Availability	Available
6	Changeability	Change
7	Completeness	Complete
8	Complexity	Complex
9	Comprehensibility	Comprehension
10	Consistency	Consistency
11	Correctability	Correct
12	Durability	Durable
13	Efficiency	Efficient
14	Effort	Effort
15	Expandability	Expand
16	Extensibility	Extension
17	Flexibility	Flexible
18	Implementation	Implement
19	Instrumentation	Instrument
20	Integrability	Integrate
21	Localization	Localization
22	Modifiability	Modify
23	Modularity	Modular
24	Perfectiveness	Perfect
25	Portability	Portable

ตารางที่ 4.1 คำหลักที่ใช้ในการค้นหาข้อเสนอแนะของผู้ตรวจทานโค้ด (ต่อ)

ลำดับ	คุณลักษณะของความสามารถในการบำรุงรักษา	คำหลัก
26	Readability	Read
27	Reusability	Reuse
28	Simplicity	Simple
29	Stability	Stable
30	Standardization	Standard
31	Testability	Test
32	Traceability	Trace
33	Understandability	Understand

4.1.2 ผลของการสืบค้นและรวบรวมคำที่มีความหมายสื่อถึงคำหลัก

จากการศึกษาข้อเสนอแนะของโครงการโอเพนซอร์สพบว่า ประโยคในข้อเสนอแนะของผู้ตรวจทานโค้ดส่วนใหญ่มักจะไม่ใช้คำที่ตรงกับคำหลักที่ผู้วิจัยได้ระบุไว้ในตารางที่ 4.1 เพื่อให้ได้ข้อมูลเกี่ยวกับการค้นหาข้อเสนอแนะทางด้านการบำรุงรักษาอย่างถูกต้อง ผู้วิจัยจึงต้องทำการจัดกลุ่มคำที่มีความหมายสื่อถึงคำหลักทั้ง 33 คำ โดยทำการค้นหาและรวบรวมจากฐานข้อมูลต่าง ๆ ได้แก่ โปรแกรมเวิร์ดเน็ต เว็บไซต์แหล่งข้อมูลเกี่ยวกับกลุ่มคำเหมือน และพจนานุกรมภาษาอังกฤษ ตัวอย่างผลลัพธ์ที่ได้จากการค้นหาคำที่มีความหมายสื่อถึงคำหลัก “Modular” แสดงดังตารางที่ 4.2

ตารางที่ 4.2 ตัวอย่างผลลัพธ์ที่ได้จากการค้นหาคำที่มีความหมายสื่อถึงคำหลัก “Modular”

เครื่องมือและฐานข้อมูล	คำที่สื่อถึงคำหลัก
โปรแกรมเวิร์ดเน็ต	modular, component, constituent, element, ~~system~~, ~~computer circuit~~
พจนานุกรม	modular, component, constituent, ingredient, composition
เว็บไซต์แหล่งข้อมูลเกี่ยวกับกลุ่มคำเหมือน	modular, component, element, segment, configuration, ~~method~~

ผู้วิจัยได้ทำการตรวจสอบผลลัพธ์ที่ได้จากการค้นหาคำที่สื่อถึงคำหลักจากโปรแกรมและแหล่งข้อมูลที่ใช้ในการค้นหาพบว่า มีคำศัพท์บางคำที่ไม่ควรนำมาจัดเป็นกลุ่มคำที่มีความหมายสื่อถึงคำหลัก เพื่อให้ข้อมูลของกลุ่มคำเหมือนที่จะนำไปใช้ในการค้นหาข้อเสนอแนะมีความสมบูรณ์มากยิ่งขึ้น และช่วยลดการเกิดข้อมูลที่ไม่มีส่วนเกี่ยวข้องกับการบำรุงรักษา ผู้วิจัยจึงได้มีการกำหนดหลักเกณฑ์ที่ใช้ในการพิจารณาการเลือกคำที่สื่อถึงคำหลักดังนี้

1) ไม่นำคำศัพท์ที่เป็นคำหลักของภาษาโปรแกรม เช่น class, function, feature, method, object, parameter เป็นต้น และไม่นำคำศัพท์เกี่ยวกับคอมพิวเตอร์โดยทั่วไปมาเป็นคำที่สื่อถึงคำหลัก เช่น system, computer, software, file, command เป็นต้น (จากตัวอย่างที่แสดงในตารางที่ 4.2 โดยใช้สัญลักษณ์ขีดฆ่าคำศัพท์ที่เกี่ยวกับภาษาโปรแกรมและคอมพิวเตอร์โดยทั่วไป)

2) คำที่มีความหมายสื่อถึงคำหลักที่ซ้ำกัน ผู้วิจัยได้เลือกคำที่มีความหมายสื่อถึงคำหลักที่ซ้ำกันจากแหล่งข้อมูลทั้ง 3 แหล่งมาเพียง 1 คำ เช่น คำว่า “component” (จากตัวอย่างที่แสดงในตารางที่ 4.2 จะแสดงคำที่ซ้ำกัน โดยใช้สัญลักษณ์ขีดเส้นใต้)

3) คำที่มีความหมายสื่อถึงคำหลักที่แตกต่างกัน ผู้วิจัยได้นำคำศัพท์ทั้งหมดที่ได้จากผลลัพธ์ทั้ง 3 แหล่งข้อมูล เพื่อจัดกลุ่มเป็นคำที่มีความหมายเหมือนกับคำหลัก

จากตารางที่ 4.2 เมื่อได้ทำการตรวจสอบคำที่มีความหมายสื่อถึงคำหลัก “Modular” แล้ว ผู้วิจัยจึงได้รวบรวมคำที่ผ่านเกณฑ์การพิจารณา ซึ่งมีด้วยกันดังนี้ modular, configuration, component, constituent, ingredient, composition, complement, element, segment

เมื่อทำการค้นหาคำที่มีความหมายสื่อถึงคำหลักและเลือกคำเหมือนที่ผ่านเกณฑ์การพิจารณาเรียบร้อยแล้ว ผู้วิจัยจึงได้ทำการรวบรวมกลุ่มคำที่มีความหมายสื่อถึงคำหลักทั้ง 33 คำ แสดงดังตารางที่ 4.3

ตารางที่ 4.3 ผลลัพธ์ที่ได้จากการค้นหาคำที่มีความหมายสื่อถึงคำหลักทั้ง 33 คำ

ลำดับ	คำหลัก	คำที่มีความหมายสื่อถึงคำหลัก
1	Accuracy	accuracy, exact, truth, certainty, precision, propriety, rectitude, validity, sure, definite, inevitable, rigorous, evident, categorical, explicit, just, lawful
2	Adapt	adapt, adjust, modulate, alter, fine, shape, regulate
3	Analyze	analyze, analysis, diagnose, assay, delineate, muse, anatomize
4	Augment	augment, amplify, spread, inflate, escalate, dilate, enhance, accumulate, increase, suffuse, raise, mount, aggrandize, splay, accrete
5	Available	available, accessible, handiness, obtainable, satisfactory, convenient, usable, benefit
6	Change	change, vary, remodel, permute, convert, transform, revamp, purge, reform
7	Complete	complete, full, absolute, plenary, finish, utter, flawless
8	Complex	complex, complicate, sophisticated, elaborate, manifold, labyrinthine, multiple, confuse, entangle, mix, muddle, discursive, tangle, intricate, bewildered, imbroglio, intricacy, jumble, obscurity
9	Comprehension	comprehension, finality, inference, conclusion, notion, realization, savvy
10	Consistency	consistency, coherence, pertinacity, adhesion, invariability, tenacity
11	Correct	correct, rectify, fit, favorable, appropriate, worthy, deserve, suitable, due, rightful, infallible, redress, regularize

ตารางที่ 4.3 ผลลัพธ์ที่ได้จากการค้นหาคำที่มีความหมายสื่อถึงคำหลัก (ต่อ)

ลำดับ	คำหลัก	คำที่มีความหมายสื่อถึงคำหลัก
12	Durable	durable, lasting, hardy, imperishable, substantial, permanent, long, immune, indissoluble, enduring, strong
13	Efficient	efficient, able, capable, competent, proficient
14	Effort	effort, endeavor, fighting, might, stamina, energy, strength, activity
15	Expand	expand, add, accretion, accrue, develop, flatten, grow, prosper, thrive
16	Extension	extension, flare, connect, continue, broaden, enlarge, widen, magnify, prolong, elongate, protract, proliferate
17	Flexible	flexible, elastic, dexterous, limber, resilient, springy, pliable, stretch
18	Implement	implement, execute, accomplish, achieve, resource, utilize, apply
19	Instrument	instrument, equipment, accessory, apparatus, machinery, appliance, device, gadget
20	Integrate	integrate, gather, collect, compile, assemble, embody, coordinate, cooperate, harmonize, consolidate, sticking, combination, joining, amassing, hoard, compound
21	Localization	localization, limitation, narrowing, restriction, stint, definition, circumscription
22	Modify	modify, customize, edit, improve, solve, repair, amend, qualify

ตารางที่ 4.3 ผลลัพธ์ที่ได้จากการค้นหาคำที่มีความหมายสื่อถึงคำหลัก (ต่อ)

ลำดับ	คำหลัก	คำที่มีความหมายสื่อถึงคำหลัก
23	Modular	modular, configuration, component, constituent, ingredient, composition, complement, element, procedure, segment
24	Perfect	perfect, excellent, ideal, immense, keen, superb, wonderful, terrific, fantastic, splendid, magnificent, superior, entirety, consummate, faultless
25	Portable	portable, mild, soft, slight, lightweight, feathery, weak, mushy, flimsy
26	Read	read, peruse, pronounce, extrapolate, imagine, speculate, surmise, see, view, interpret, transliterate, look
27	Reuse	reuse, recycle, reiterate, rehash, reclaim, exploit, revise
28	Simple	simple, ease, expedient, facile, easy
29	Stable	stable, still, steady, invariable, constancy, endurance, firmness, fastness, indissolubility, sturdily
30	Standard	standard, formula, archetypal, representative, typical, characteristic, degree, tier, quality, criterion, measurement, imperative, touchstone, property
31	Test	test, verify, check, prove, evaluate, examine, assess, attempt, experiment, inspect, trial, proof, tryout, investigate
32	Trace	trace, pursue, tag, trail, detect, search, seek, probe, meaning, significance, hint, consequence, point, follow, behave, track, extract, transcribe
33	Understand	understand, explain, fathom, grasp, knowledge, perceive

4.1.3 ผลที่ได้จากการค้นหาคุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์เพิ่มเติมจากที่มีอยู่

เพื่อเพิ่มหลักฐานเชิงประจักษ์ว่า นักพัฒนาซอฟต์แวร์ในโครงการ Eclipse และโครงการ Qt ได้มีแก้ไขปรับปรุงซอร์สโค้ดได้อย่างครอบคลุมทุกคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ ผู้วิจัยได้ทำการค้นหาคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์เพิ่มเติมนอกเหนือจากที่ได้ทำการรวบรวมไว้ทั้ง 33 คุณลักษณะ ที่ได้ระบุไว้ในหัวข้อที่ 4.1.1 โดยใช้อัลกอริทึม LDA ในการค้นหาคำที่มักจะปรากฏในข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษา โดยรูปที่ 4.1 จะแสดงคำสั่งการใช้ package LDA ในโปรแกรม R

```
1 > require(lda)
2 > corpusLDA <- lexicalize(docLines)
3 > ldaModel=lda.collapsed.gibbs.sampler(corpusLDA$documents,K=5,v
4   ocab=corpusLDA$vocab,burnin=9999,num.iterations=10000,alpha=1
5   ,beta=0.1)
6 > top.words <- top.topic.words(ldaModel$topics,6,by.score=TRUE)
7 > print(top.words)
```

รูปที่ 4.1 คำสั่งการใช้อัลกอริทึม LDA ในโปรแกรม R

ชุดคำสั่งในรูปที่ 4.1 ประกอบด้วยคำสั่งดังรายละเอียดต่อไปนี้

บรรทัดที่ 1 : เป็นการเรียกใช้งาน package lda

บรรทัดที่ 2 : เป็นการสร้างคลังเอกสารสำหรับใช้ในกระบวนการของอัลกอริทึม LDA โดย “`lexicalize(docLines)`” เป็นฟังก์ชันที่ใช้ในการอ่านข้อความในเอกสาร และจัดเก็บคลังเอกสารอยู่ในตัวแปรที่มีชื่อว่า “`corpusLDA`” ซึ่งเอกสารที่ใช้ในกระบวนการนี้จะมีข้อมูลเกี่ยวกับข้อเสนอแนะของผู้ตรวจทานโค้ดที่ให้คำแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ของโครงการ Eclipse และโครงการ Qt

บรรทัดที่ 3-5 : เป็นการกำหนดค่าของพารามิเตอร์ที่ใช้ในการค้นหาคำหลักหรือคุณลักษณะเกี่ยวกับการบำรุงรักษาที่ซ่อนอยู่ในข้อเสนอแนะ เพื่อให้ข้อมูลที่แสดงผลมีความถูกต้องและแม่นยำยิ่งขึ้น ผู้วิจัยจึงได้กำหนดให้มีการค้นหาอย่างซ้ำ ๆ ในชุดคำสั่งนี้มีการระบุจำนวนกลุ่มของหัวข้อ “`K`” ที่มีความเกี่ยวข้องกันให้มาอยู่ในกลุ่มเดียวกัน (ในกรณีนี้ ผู้วิจัยได้เลือกไว้ 5 หัวข้อ) มีการสุ่มตัวอย่างโดยกำหนดค่าของ “`burnin`” เป็น 9,999 ตัวอย่าง และระบุการค้นหาคำหลักที่ผ่านขั้นตอนการค้นหาเอกสารซ้ำ ๆ สำหรับแต่ละหัวข้อ โดยกำหนด “`num.iterations`” มีการค้นหาเป็น 10,000 ครั้ง (ค่าของ `burnin` จะไม่ซ้ำกับค่าของ `num.iterations`) ซึ่งการกำหนดค่าเหล่านี้จะใช้เพื่อทำการสุ่มเก็บ

ข้อมูลทั้งหมดในคลังเอกสารของ “corpusLDA” โดยใช้วิธีการสุ่มตัวอย่างหรือการจำลองข้อมูลแบบที่เรียกว่าการสุ่มตัวอย่างแบบกิบส์ (Gibbs Sampling) นอกจากนี้ยังมีการกำหนดค่าที่จะใช้ในการกระจายตัวของเอกสารและหัวข้อ ซึ่งค่าของ “alpha” เป็นค่าของเอกสารที่จะกระจายตัวไปยังหัวข้อผ่านทางเอกสารต่าง ๆ โดยมีค่าเริ่มต้นเป็น 1 และค่าของ “beta” เป็นค่าการกระจายตัวไปยังคำต่าง ๆ ที่ปรากฏอยู่ในข้อเสนอแนะของแต่ละหัวข้อ โดยกำหนดให้มีค่าเริ่มต้นเป็น 0.1

บรรทัดที่ 6 : เป็นการค้นหาคำหลักที่ซ่อนอยู่ในเอกสาร โดยผ่านกระบวนการอัลกอริทึม LDA คือ การคำนวณ K x V ซึ่ง “K” เป็นจำนวนของหัวข้อ และ “v” เป็นจำนวนของคำหลักในแต่ละหัวข้อ โดยรูปที่ 4.1 ได้กำหนดให้มีการแสดง 6 คำหลักของทั้ง 5 หัวข้อที่มีการเรียงลำดับคะแนนสูงสุด-ต่ำสุดที่เกิดขึ้นในอัลกอริทึม LDA

บรรทัดที่ 7 : เป็นคำสั่งที่ใช้ในการแสดงผลลัพธ์ที่เกิดขึ้นในอัลกอริทึม LDA ซึ่งในที่นี้จะแสดงผลลัพธ์ที่ได้จากการค้นหาในตารางที่ 4.4

ตารางที่ 4.4 ผลลัพธ์ที่ได้จากการใช้อัลกอริทึม LDA ในโปรแกรม R

	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
[1]	change	use	add	work	method
[2]	convert	sure	adjust	example	tool
[3]	parameter	suggest	file	make	class
[4]	patch	implement	test	code	review
[5]	index	cohesion	improve	model	package
[6]	name	resource	worth	duplicate	commit

ผลลัพธ์ที่ได้จากการใช้อัลกอริทึม LDA ในตารางที่ 4.4 ประกอบด้วยคำหลักและคำที่สื่อความหมายเหมือนกับคำหลักปรากฏอยู่ในผลลัพธ์ด้วย ตัวอย่างเช่น change, adjust, test, improve ดังนั้นการเลือกคุณลักษณะทางด้านการบำรุงรักษาประเภทใหม่ที่ได้จากอัลกอริทึม LDA ผู้วิจัยจึงพิจารณาจากคำที่มีความถี่ในการค้นพบสูงที่สุดของแต่ละกลุ่ม โดยที่คำนั้นจะต้องไม่ใช่คำหลักและคำที่สื่อถึงคำหลักที่มีอยู่ก่อนแล้ว นอกจากนี้ยังรวมไปถึงคำศัพท์ที่เป็นคำหลักของภาษาโปรแกรม คำศัพท์เกี่ยวกับคอมพิวเตอร์ และคำศัพท์สามัญที่พบได้ทั่วไป เช่น parameter, method, package, work, make เป็นต้น

ในตารางที่ 4.5 เป็นการแสดงคุณลักษณะประเภทใหม่ของการบำรุงรักษาซอฟต์แวร์ที่ผ่านการพิจารณาหลักเกณฑ์ในการเลือกคำหลักข้างต้น

ตารางที่ 4.5 คุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาที่ค้นพบจากการใช้อัลกอริทึม LDA

	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5
[1]	~~change~~	~~use~~	~~add~~	~~work~~	~~method~~
[2]	~~convert~~	~~sure~~	~~adjust~~	~~example~~	~~tool~~
[3]	~~parameter~~	~~suggest~~	~~file~~	~~make~~	~~class~~
[4]	~~patch~~	~~implement~~	~~test~~	~~code~~	~~review~~
[5]	~~index~~	cohesion	~~improve~~	~~model~~	~~package~~
[6]	~~name~~	~~resource~~	~~worth~~	duplicate	~~commit~~

เมื่อทำการพิจารณาคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาประเภทใหม่ที่ค้นพบจากการใช้อัลกอริทึม LDA พบว่าบางกลุ่มไม่มีคำที่เหมาะสม ดังนั้นในโครงการ Eclipse และโครงการ Qt จึงมีคุณลักษณะเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ที่ค้นพบใหม่ด้วยกัน 2 ประเภท ได้แก่ cohesion และ duplicate เพื่อที่จะทำการกำหนดคำนิยามของคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาประเภทใหม่ที่ค้นพบ ผู้วิจัยจึงได้นำข้อเสนอแนะที่ผู้ตรวจทานโค้ดให้คำแนะนำเกี่ยวกับการบำรุงรักษามาทำการค้นหาคำที่มักจะเกิดขึ้นร่วมกับคุณลักษณะทางด้านการบำรุงรักษาประเภทใหม่ทั้ง 2 ประเภท โดยใช้คำสั่งในโปรแกรม R ดังนี้

```
> findAssocs(docs.dtm, terms = "cohesion", corlimit = 0.6)
> findAssocs(docs.dtm, terms = "duplicate", corlimit = 0.6)
```

ฟังก์ชัน “`findAssocs`” เป็นการค้นหาความสัมพันธ์ระหว่างคำที่เกิดขึ้นร่วมกับคำหลักภายใต้คลังข้อมูล โดยในงานวิจัยนี้ผู้วิจัยได้กำหนดให้มีการค้นหาคำที่มักจะเกิดขึ้นร่วมกับ “`cohesion`” และ “`duplicate`” ที่มีการกำหนดค่าความสัมพันธ์ 0.6 นั่นคือ ผลลัพธ์ที่มีการค้นพบร่วมกันกับคำหลักอย่างน้อย 60% ซึ่งผลลัพธ์ของความสัมพันธ์ระหว่างคำที่มักจะเกิดขึ้นร่วมกับคุณลักษณะประเภทใหม่แสดงดังตารางที่ 4.6

ตารางที่ 4.6 ผลลัพธ์ของความสัมพันธ์ระหว่างคำที่มักจะเกิดขึ้นร่วมกับคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ประเภทใหม่

คุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษา	คำที่มักจะเกิดขึ้นร่วมกับคุณลักษณะ	ค่าความสัมพันธ์ที่เกิดขึ้นระหว่างคำ
Cohesion	Module	0.82
	Class	0.78
	Elements	0.76
	Function	0.67
	Case	0.67
	Work	0.65
	Test	0.63
Duplicate	Function	0.89
	Code	0.83
	Component	0.78
	Method	0.75
	Object	0.68
	Use	0.62

จากผลลัพธ์ตารางที่ 4.6 ผู้วิจัยจะนำคำที่มักจะเกิดขึ้นร่วมกับคุณลักษณะทางด้านการบำรุงรักษาประเภทใหม่ที่ค้นพบมาพิจารณาร่วมกับข้อเสนอแนะที่มีการระบุคำหลักทั้ง 2 ประเภท เพื่อให้ได้คำจำกัดความที่ชัดเจนของคุณลักษณะทางด้านการบำรุงรักษาประเภทใหม่

เมื่อได้ทำการพิจารณาข้อเสนอแนะที่มีคุณลักษณะทางด้านการบำรุงรักษาประเภทใหม่และคำร่วมเรียบร้อยแล้ว ผู้วิจัยจึงได้ให้คำนิยามหรือคำจำกัดความของคุณลักษณะทางด้านการบำรุงรักษาทั้ง 2 คุณลักษณะดังนี้

1) Cohesion module เป็นกิจกรรมในการระบุโมดูลที่มีการทำงานร่วมกัน โดยมีการเชื่อมความสัมพันธ์ระหว่างส่วนประกอบที่มีอยู่ในโมดูล เช่น องค์ประกอบที่มีการทำงานพร้อม ๆ กันหรือองค์ประกอบที่มีการนำข้อมูลเข้าตัวเดียวกัน แต่มีการทำงานไม่เหมือนกัน
2) Duplicate function เป็นกิจกรรมในการรวมฟังก์ชันที่มีคุณสมบัติหรือขั้นตอนการทำงานที่คล้ายกันให้อยู่ในฟังก์ชันเดียว เพื่อลดการทำงานซ้ำซ้อนของฟังก์ชันที่อาจก่อให้เกิดความสับสนในการเรียกใช้งาน

เมื่อได้กำหนดคำนิยามของคุณลักษณะทางด้านการบำรุงรักษาประเภทใหม่ทั้ง 2 ประเภทเรียบร้อยแล้ว ผู้วิจัยจึงทำการจัดกลุ่มคำที่มีความหมายเหมือนกับคำหลักเช่นเดียวกับหัวข้อที่ 4.1.2 โดยสืบค้นจากโปรแกรมเวิร์ดเน็ต เว็บไซต์แหล่งข้อมูลเกี่ยวกับกลุ่มคำเหมือน และพจนานุกรมภาษาอังกฤษ ซึ่งผลลัพธ์จากการสืบค้นคำเหมือนของคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาประเภทใหม่แสดงดังตารางที่ 4.7

ตารางที่ 4.7 ผลลัพธ์ของการค้นหาคำที่สื่อถึงคำหลักของคุณลักษณะทางด้านการบำรุงรักษาประเภทใหม่ที่ค้นพบ

ลำดับ	คำหลัก	คำที่มีความหมายสื่อถึงคำหลัก
1	Cohesion	cohesion, sticky, tough, gummy, leathery, tenacious
2	Duplicate	duplicate, repetitive, double, copy, transcript, counterpart, facsimile, reproduce, mimeograph, repeat, replicate

4.2 นักพัฒนาซอฟต์แวร์ให้ความสนใจกับความสามารถในการบำรุงรักษาซอฟต์แวร์ภายใต้การตรวจทานโค้ดในโครงการโอเพนซอร์สหรือไม่

ผู้วิจัยได้ใช้เทคนิคการทำเหมืองข้อความ เพื่อค้นหาข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาและตรวจสอบว่า ชุมชนนักพัฒนาในโครงการ Eclipse และโครงการ Qt ได้ทำการแก้ไขซอร์สโค้ดตามคำแนะนำทางด้านการบำรุงรักษามากน้อยเพียงใด พร้อมทั้งนำเสนอแนวโน้มที่พัฒนาซอฟต์แวร์จะทำการปรับปรุงแก้ไขซอร์สโค้ดตามข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ โดยมีการอธิบายรายละเอียดของผลลัพธ์ดังนี้

ผู้วิจัยได้นำคำหลักทั้ง 35 ประเภทที่รวบรวมจากวารสารที่ระบุถึงคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาทั้ง 33 ประเภท และคุณลักษณะเกี่ยวกับการบำรุงรักษาประเภทใหม่ที่ค้นพบโดยใช้อัลกอริทึม LDA อีก 2 ประเภท มาทำการค้นหาข้อเสนอแนะที่ผู้ตรวจทานโค้ดให้คำแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ ซึ่งผู้วิจัยได้ทำการพัฒนาชุดคำสั่ง (Script) ในโปรแกรม R ซึ่งเป็นโปรแกรมที่มีแพ็กเกจในการทำเหมืองข้อความ (tm package)

ตัวอย่างข้อเสนอแนะของโครงการ Eclipse ที่ได้จากการค้นหา เช่น

(1) “Please add a method to test for possible error condition. isValid() maybe.”

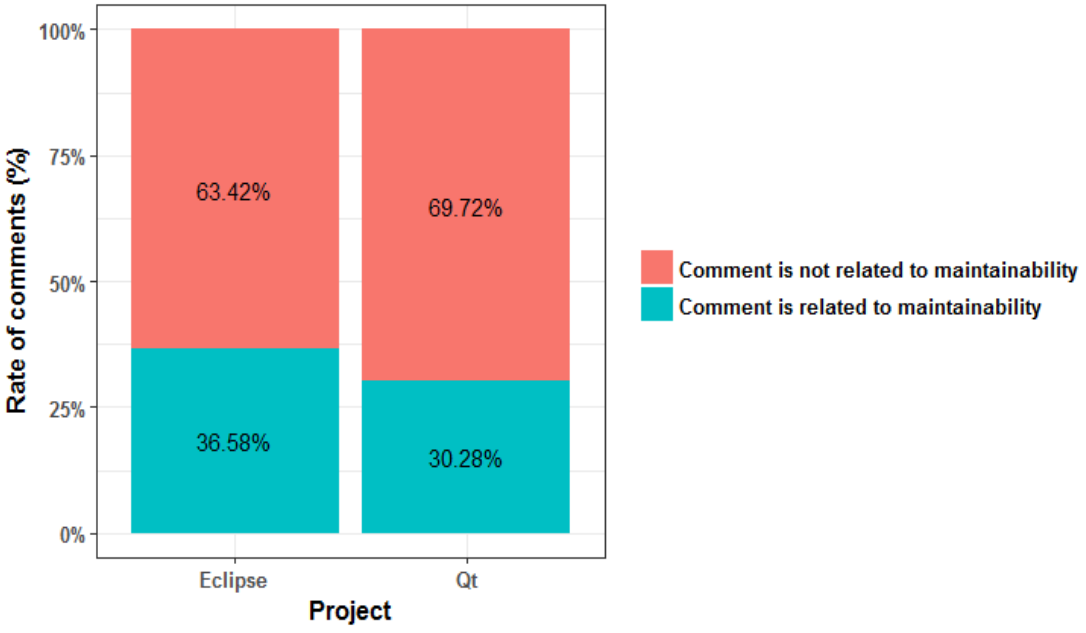
(2) “Another possibility is to get isValidThread to call isCurrentThread(), then the isValidThread can be updated to another implementation if desired without code duplication.”

สำหรับตัวอย่างข้อเสนอแนะดังกล่าวเป็นข้อเสนอแนะที่ได้จากการค้นหาคำหลัก “Test” และการค้นหาคำหลัก “Duplicate” ซึ่งเป็นคุณลักษณะประเภทใหม่ที่ค้นพบ สำหรับผลลัพธ์ของข้อมูลที่ได้จากการค้นหาข้อเสนอแนะเกี่ยวกับความสามารถในการบำรุงรักษาทั้ง 35 คุณลักษณะของโครงการ Eclipse และโครงการ Qt โดยผ่านกระบวนการทำเหมืองข้อความแสดงดังตารางที่ 4.8

ตารางที่ 4.8 รายละเอียดข้อมูลของโครงการโอเพนซอร์สทั้ง 2 โครงการที่มีการแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ในช่วงปี ค.ศ. 2012 - 2016

โครงการโอเพนซอร์ส	Eclipse	Qt
ภาษาที่ใช้ในการพัฒนา	Java	C++
จำนวนข้อเสนอแนะทั้งหมด	108,357	309,165
จำนวนผู้ตรวจทานโค้ดทั้งหมด	748	1,059
จำนวนข้อเสนอแนะทางด้านการบำรุงรักษาซอฟต์แวร์	39,638	93,629
จำนวนผู้ตรวจทานโค้ดที่ให้คำแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์	668	885

ตารางที่ 4.8 แสดงให้เห็นว่าจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาที่ค้นพบในโครงการ Eclipse และ Qt เป็นระยะเวลา 5 ปี นั่นคือ ในช่วงปี ค.ศ. 2012 - 2016 คิดเป็นอัตราส่วนได้เท่ากับ 36.58% และ 30.28% ตามลำดับ แสดงดังรูปที่ 4.2



รูปที่ 4.2 อัตราส่วนของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์

รูปที่ 4.2 เป็นการแสดงอัตราส่วนของข้อเสนอแนะทางด้านความสามารถในการบำรุงรักษาซอฟต์แวร์ทั้ง 35 ประเภท โดยผลลัพธ์ที่แสดงดังรูปที่ 4.2 แสดงให้เห็นว่าทั้ง 2 โครงการมีจำนวนข้อเสนอแนะที่ผู้ตรวจทานโค้ดให้คำแนะนำทางด้านการบำรุงรักษาซอฟต์แวร์ค่อนข้างน้อย เมื่อเทียบกับจำนวนข้อเสนอแนะที่ได้รับจากผู้ตรวจทานโค้ดทั้งหมด

เมื่อได้พิจารณาข้อมูลของจำนวนผู้ตรวจทานโค้ดทั้ง 2 โครงการ พบว่าแม้จำนวนของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ที่ได้ทำการค้นหาจะคิดเป็นอัตราส่วนประมาณ 1 ใน 3 ของจำนวนข้อเสนอแนะทั้งหมด แต่จำนวนของผู้ตรวจทานโค้ดในโครงการ Eclipse และโครงการ Qt ที่ให้คำแนะนำทางด้านการบำรุงรักษามีจำนวน 668 และ 885 ตามลำดับ ซึ่งเมื่อเทียบกับจำนวนผู้ตรวจทานโค้ดทั้งหมดถือว่า โครงการ Eclipse และโครงการ Qt มีผู้ตรวจทานโค้ดจำนวน 89.31% และ 83.57% ที่ให้คำแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ ดังนั้นจึงสามารถสรุปได้ว่าผู้ตรวจทานโค้ด 1 คน จะมีการให้ข้อเสนอแนะเกี่ยวกับการบำรุงรักษาซอฟต์แวร์มากกว่า 1 ข้อเสนอแนะ ซึ่งข้อมูลดังกล่าวเหล่านี้สามารถเป็นหลักฐานเบื้องต้นที่บ่งบอกได้ว่า ผู้ตรวจทานโค้ดส่วนใหญ่ในโครงการโอเพนซอร์สให้ความสำคัญกับการบำรุงรักษาซอฟต์แวร์และคำแนะนำทางด้านการบำรุงรักษาซอฟต์แวร์ที่เกิดขึ้นมักจะได้รับจากผู้ตรวจทานโค้ดคนเดิม

4.2.1 ผลของการค้นหาข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษา

เมื่อได้ทำการค้นหาข้อเสนอแนะทางด้านความสามารถในการบำรุงรักษาซอฟต์แวร์ทั้ง 35 ประเภท ผู้วิจัยพบว่าในหนึ่งข้อเสนอแนะจะมีการให้คำแนะนำเกี่ยวกับการบำรุงรักษาหลายคุณลักษณะ เนื่องจากผู้ตรวจทานโค้ดมักพบปัญหาหรือข้อบกพร่องหลาย ๆ กรณีที่ควรจะแก้ไขปรับปรุงโค้ด เช่น ผู้ตรวจทานโค้ดคนหนึ่งในโครงการ Eclipse ได้ให้คำแนะนำว่า

"I think this method performTest() is useless and makes reading the test code more complex than required. Please remove it"

คำแนะนำดังกล่าวแสดงให้เห็นว่า ผู้ตรวจทานโค้ดแนะนำให้นักพัฒนาซอฟต์แวร์ทำการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ด้วยกัน 3 ประเภท นั่นคือ Testability Readability และ Complexity ดังนั้นในตารางที่ 4.9 และตารางที่ 4.10 จะแสดงรายละเอียดจำนวนความถี่ของคำหลักที่ปรากฏในข้อเสนอแนะของโครงการ Eclipse และโครงการ Qt ที่เกิดขึ้นในช่วงปี ค.ศ. 2012 – 2016

ตารางที่ 4.9 จำนวนความถี่ของคำหลักที่ปรากฏทั้งหมดในข้อเสนอแนะของโครงการ Eclipse

ลำดับ	คำหลัก	ปี				
		2012	2013	2014	2015	2016
1	Accuracy	640	1,164	1,732	1,986	1,730
2	Adapt	74	136	185	241	270
3	Analyze	4	101	186	122	264
4	Augment	15	41	87	78	86
5	Available	70	165	231	196	188
6	Change	654	1,442	2,179	2,390	2,623
7	Cohesion	75	140	263	277	240
8	Complete	78	205	259	328	361
9	Complex	172	285	375	359	353
10	Comprehension	7	11	23	33	38
11	Consistency	41	120	139	153	88
12	Correct	142	254	413	474	449

ตารางที่ 4.9 จำนวนความถี่ของคำหลักที่ปรากฏทั้งหมดในข้อเสนอแนะของโครงการ Eclipse (ต่อ)

ลำดับ	คำหลัก	ปี				
		2012	2013	2014	2015	2016
13	Duplicate	89	251	361	383	294
14	Durable	52	137	194	210	197
15	Efficient	12	27	44	41	30
16	Effort	110	267	338	348	440
17	Expand	354	932	1,262	1,342	1,347
18	Extension	84	146	240	190	179
19	Flexible	7	10	15	11	18
20	Implement	129	257	453	388	566
21	Instrument	3	15	30	27	25
22	Integrate	7	13	24	20	15
23	Localization	50	90	126	139	126
24	Modify	72	127	245	227	234
25	Module	61	166	323	381	347
26	Perfect	5	17	39	33	45
27	Portable	8	5	13	20	43
28	Read	465	928	1,419	1,433	1,551
29	Reuse	23	36	47	70	71
30	Simple	46	95	148	116	86
31	Stable	127	225	336	372	384
32	Standard	55	145	215	215	194
33	Test	590	1,292	2,132	2,331	1,951
34	Trace	481	1,073	1,436	1,417	1,393
35	Understand	104	187	342	372	402

ตารางที่ 4.10 จำนวนความถี่ของคำหลักที่ปรากฏทั้งหมดในข้อเสนอแนะของโครงการ Qt

ลำดับ	คำหลัก	ปี				
		2012	2013	2014	2015	2016
1	Accuracy	289	3,327	3,556	3,090	4,285
2	Adapt	17	452	531	459	596
3	Analyze	1	12	21	11	25
4	Augment	16	91	207	224	539
5	Available	311	2,582	2,837	2,587	3,567
6	Change	389	4,516	5,702	10,932	12,262
7	Cohesion	56	565	592	564	741
8	Complete	42	337	447	402	507
9	Complex	61	620	685	562	724
10	Comprehension	6	133	152	139	215
11	Consistency	27	319	267	246	334
12	Correct	70	870	963	877	1,272
13	Duplicate	78	729	792	758	1,227
14	Durable	28	286	314	321	472
15	Efficient	25	210	241	207	280
16	Effort	75	798	894	718	979
17	Expand	114	1,694	1,733	1,577	1,842
18	Extension	39	389	453	465	514
19	Flexible	2	23	30	26	41
20	Implement	98	881	1,048	962	1,172
21	Instrument	16	389	500	340	344
22	Integrate	6	61	56	35	63
23	Localization	28	272	291	284	413
24	Modify	40	455	467	381	535
25	Module	48	835	857	701	1,014
26	Perfect	8	62	102	87	121

ตารางที่ 4.10 จำนวนความถี่ของคำหลักที่ปรากฏทั้งหมดในข้อเสนอแนะของโครงการ Qt (ต่อ)

ลำดับ	คำหลัก	ปี				
		2012	2013	2014	2015	2016
27	Portable	15	115	102	100	146
28	Read	268	2,684	2,934	2,600	3,330
29	Reuse	6	52	56	42	71
30	Simple	29	244	232	218	310
31	Stable	80	944	998	943	1,279
32	Standard	72	772	715	664	857
33	Test	227	2,431	2,622	2,338	3,035
34	Trace	159	1,705	1,928	1,718	2,360
35	Understand	50	560	668	589	767

4.2.2 ผลของจำนวนการแก้ไขโค้ดที่ได้รับคำแนะนำทางด้านการบำรุงรักษา

หลังจากที่ได้ค้นหาข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ทั้ง 35 คุณลักษณะเรียบร้อยแล้ว ผู้วิจัยได้นำข้อมูลดังกล่าวไปทำการวิเคราะห์ต่อ โดยใช้โปรแกรม R ในการจับคู่ระหว่างข้อเสนอแนะเกี่ยวกับการบำรุงรักษาและข้อเสนอแนะทั้งหมดที่ถูกจัดเก็บอยู่ในระบบฐานข้อมูล โดยมีวัตถุประสงค์เพื่อที่จะตรวจสอบว่า นักพัฒนาได้ทำการแก้ไขโค้ดตามข้อเสนอแนะที่เกี่ยวกับการบำรุงรักษามากน้อยเพียงใด สำหรับวิธีการตรวจสอบนี้ ผู้วิจัยได้ใช้วิธีการตรวจสอบด้วยตัวผู้วิจัยเอง โดยการอ่านข้อเสนอแนะที่ได้จากการค้นหาและการตอบกลับของนักพัฒนา เช่น Done ซึ่งเป็นข้อความตอบกลับที่แสดงให้เห็นว่า นักพัฒนาได้ทำการแก้ไขโค้ดตามคำแนะนำของผู้ตรวจทานโค้ดแล้วนั่นเอง

สำหรับผลลัพธ์ที่ได้จากการตรวจสอบด้วยวิธีการดังกล่าวข้างต้น คือ จำนวนข้อเสนอแนะที่ได้รับการแก้ไขโค้ดตามคำแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ของโครงการ Eclipse และโครงการ Qt แสดงดังตารางที่ 4.11 และตารางที่ 4.12 โดยผลลัพธ์จากตารางจะแสดงให้เห็นถึงจำนวนข้อเสนอแนะที่ได้รับการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาที่เพิ่มขึ้น-ลดลงในแต่ละปี (ค.ศ. 2012-2016) และเมื่อทำการเปรียบเทียบกับจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาที่แสดงดังตารางที่ 4.9 และตารางที่ 4.10 ในหัวข้อ 4.2.1 จะสามารถตอบคำถามวิจัยได้ว่า นักพัฒนาซอฟต์แวร์ให้ความสนใจต่อการบำรุงรักษาซอฟต์แวร์ทั้ง 35 คุณลักษณะของโครงการ Eclipse และโครงการ Qt ในช่วงระยะเวลา 5 ปีหรือไม่

ตารางที่ 4.11 จำนวนการแก้ไขโค้ดทางด้านการบำรุงรักษาของโครงการ Eclipse

ลำดับ	คำหลัก	ปี				
		2012	2013	2014	2015	2016
1	Accuracy	367	590	981	1,098	947
2	Adapt	53	84	126	164	182
3	Analyze	1	47	96	73	137
4	Augment	14	23	59	44	52
5	Available	52	98	157	120	118
6	Change	442	843	1,334	1,460	1,603
7	Cohesion	44	70	163	173	137
8	Complete	58	125	172	189	218
9	Complex	117	169	216	230	229
10	Comprehension	6	7	15	22	33
11	Consistency	27	74	72	97	54
12	Correct	99	164	252	286	276
13	Duplicate	54	133	214	225	158
14	Durable	27	83	117	142	127
15	Efficient	10	13	31	29	18
16	Effort	58	140	217	206	246
17	Expand	224	386	680	695	684
18	Extension	48	88	129	114	105
19	Flexible	4	4	9	8	14
20	Implement	91	145	278	246	337
21	Instrument	2	6	19	17	16
22	Integrate	5	10	15	14	11
23	Localization	33	56	79	79	76
24	Modify	49	75	140	132	142
25	Module	33	80	191	209	203

ตารางที่ 4.11 จำนวนการแก้ไขโค้ดทางด้านการบำรุงรักษาของโครงการ Eclipse (ต่อ)

ลำดับ	คำหลัก	ปี				
		2012	2013	2014	2015	2016
26	Perfect	2	13	29	18	30
27	Portable	6	2	4	15	28
28	Read	320	528	879	870	964
29	Reuse	17	21	34	38	37
30	Simple	31	63	88	70	61
31	Stable	93	134	219	229	262
32	Standard	32	74	115	110	111
33	Test	332	536	932	1,059	894
34	Trace	294	531	823	851	850
35	Understand	78	127	219	237	277

ตารางที่ 4.12 จำนวนการแก้ไขโค้ดทางด้านการบำรุงรักษาของโครงการ Qt

ลำดับ	คำหลัก	ปี				
		2012	2013	2014	2015	2016
1	Accuracy	196	2,320	2,562	2,226	3,122
2	Adapt	12	344	403	346	460
3	Analyze	0	9	15	9	22
4	Augment	13	64	115	90	135
5	Available	137	1,441	1,629	1,519	2,125
6	Change	209	2,751	3,168	3,491	4,873
7	Cohesion	39	387	369	401	514
8	Complete	23	240	324	273	374
9	Complex	40	458	484	433	544
10	Comprehension	4	106	118	110	177
11	Consistency	15	224	178	179	255
12	Correct	50	650	704	674	955

ตารางที่ 4.12 จำนวนการแก้ไขโค้ดทางด้านการบำรุงรักษาของโครงการ Qt (ต่อ)

ลำดับ	คำหลัก	ปี				
		2012	2013	2014	2015	2016
13	Duplicate	46	479	550	496	869
14	Durable	21	221	234	233	347
15	Efficient	18	157	193	156	213
16	Effort	40	571	647	507	740
17	Expand	64	1,075	1,131	1,044	1,272
18	Extension	22	233	273	299	356
19	Flexible	2	15	20	20	36
20	Implement	61	616	726	700	820
21	Instrument	11	242	309	241	232
22	Integrate	5	39	45	24	43
23	Localization	16	186	218	203	309
24	Modify	28	334	346	281	398
25	Module	28	483	519	444	651
26	Perfect	8	53	87	68	102
27	Portable	12	85	79	77	113
28	Read	161	1,886	2,084	1,912	2,481
29	Reuse	4	44	46	43	52
30	Simple	20	184	174	188	253
31	Stable	56	682	729	669	943
32	Standard	42	507	472	460	593
33	Test	146	1,545	1,726	1,500	2,067
34	Trace	104	1,239	1,405	1,256	1,773
35	Understand	38	407	499	463	605

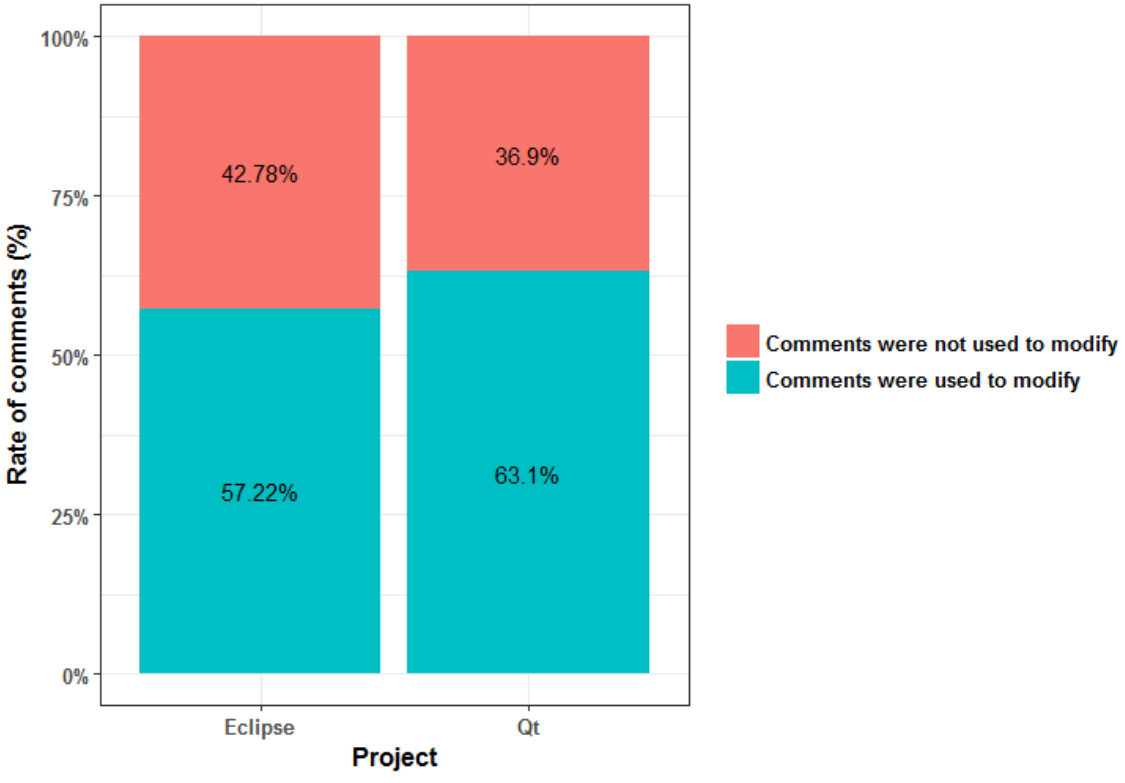
4.2.3 การสรุปผลลัพธ์ของการตรวจสอบความสนใจของชุมชนนักพัฒนาในโครงการโอเพนซอร์สที่มีต่อการบำรุงรักษาซอฟต์แวร์

เพื่อที่จะตรวจสอบว่า นักพัฒนาซอฟต์แวร์ในโครงการ Eclipse และโครงการ Qt ให้ความสำคัญกับการบำรุงรักษาซอฟต์แวร์มากน้อยแค่ไหน ผู้วิจัยได้ทำการสรุปผลลัพธ์จากการตรวจสอบจำนวนการแก้ไขปรับปรุงซอร์สโค้ดที่นักพัฒนาซอฟต์แวร์ทำตามคำแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ของผู้ตรวจทานโค้ดในโครงการโอเพนซอร์ส ซึ่งรายละเอียดข้อมูลเกี่ยวกับจำนวนข้อเสนอแนะที่ได้จากการค้นหาทั้ง 35 คุณลักษณะและจำนวนข้อเสนอแนะที่ได้รับการแก้ไขโค้ดทางด้านการบำรุงรักษาในโครงการโอเพนซอร์ส Eclipse และโครงการ Qt ในช่วงปี ค.ศ. 2012 - 2016 แสดงดังตารางที่ 4.13

ตารางที่ 4.13 รายละเอียดผลลัพธ์ที่ได้จากการวิเคราะห์ข้อเสนอแนะเกี่ยวกับการบำรุงรักษาที่ได้รับการแก้ไขโค้ดในช่วงปี ค.ศ. 2012-2016

โครงการโอเพนซอร์ส	Eclipse	Qt
จำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์	64,616	149,610
จำนวนข้อเสนอแนะเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ที่ได้รับการแก้ไขโค้ด	36,975	94,408

ตารางที่ 4.13 แสดงให้เห็นว่าจำนวนการแก้ไขปรับปรุงซอร์สโค้ดที่เกี่ยวข้องกับคุณลักษณะย่อยของการบำรุงรักษาทั้ง 35 ประเภทของโครงการ Eclipse คือ 36,975 ข้อเสนอแนะและโครงการ Qt มีการแก้ไขโค้ดปรับปรุงซอร์สโค้ดทางด้านการบำรุงรักษาจำนวน 94,408 ข้อเสนอแนะ เมื่อเทียบกับจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ทั้งหมด ซึ่งจำนวนดังกล่าวสามารถคิดเป็น 57.22% และ 63.10% ตามลำดับ แสดงดังรูปที่ 4.3



รูปที่ 4.3 อัตราส่วนของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษา

จากรูปที่ 4.3 เป็นการแสดงอัตราส่วนของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาทั้ง 35 ประเภท ซึ่งผลลัพธ์ดังกล่าวก็ทำให้สามารถสรุปได้ว่า ชุมชนนักพัฒนาซอฟต์แวร์ในโครงการ Eclipse และโครงการ Qt ให้ความสำคัญกับการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาซอฟต์แวร์มากกว่า 50% เมื่อเทียบกับจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาทั้งหมดของผู้ตรวจทานโค้ด และรูปที่ 4.4 เป็นการแสดงอัตราส่วนของข้อเสนอแนะที่มีการแก้ไขปรับปรุงซอร์สโค้ดตามคำแนะนำทางด้านการบำรุงรักษาในช่วงระยะเวลา 5 ปี (ค.ศ. 2012 - 2016) โดยผลลัพธ์ดังกล่าวแสดงให้เห็นว่า ในแต่ละปีมีการแก้ไขโค้ดทางด้านการบำรุงรักษาประมาณ 50-60% เมื่อเทียบกับจำนวนข้อเสนอแนะที่ผู้ตรวจทานโค้ดได้ให้คำแนะนำเกี่ยวกับบำรุงรักษาซอฟต์แวร์ทั้งหมด

รูปที่ 4.4 อัตราส่วนของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดทางด้านการบำรุงรักษาในช่วง 5 ปี

เพื่อตอบคำถามวิจัยว่า *"นักพัฒนาซอฟต์แวร์ให้ความสนใจกับความสามารถในการบำรุงรักษาซอฟต์แวร์ภายใต้การตรวจทานโค้ดในโครงการโอเพนซอร์สหรือไม่"* ผู้วิจัยจึงได้ทำการวิเคราะห์ข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์และตรวจสอบการแก้ไขโค้ดของนักพัฒนาซอฟต์แวร์ในช่วงปี ค.ศ. 2012 – 2016 ทำให้สามารถสรุปได้ว่า ชุมชนนักพัฒนาในโครงการ Eclipse และโครงการ Qt ได้ทำการแก้ไขโค้ดตามคำแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ประมาณ 50-60% ดังนั้นผลลัพธ์ข้างต้นจึงเป็นหลักฐานเชิงประจักษ์ที่สามารถตอบคำถามวิจัยได้ว่า นักพัฒนาซอฟต์แวร์ในโครงการ Eclipse และโครงการ Qt ให้ความสนใจกับการบำรุงรักษาซอฟต์แวร์มากกว่า 50%

4.2.4 ความสัมพันธ์ระหว่างจำนวนข้อเสนอแนะที่ได้จากการค้นหาและจำนวนการแก้ไขโค้ดตามคำแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ของทั้ง 2 โครงการ

ผู้วิจัยได้ใช้สถิติสหสัมพันธ์ (Correlation) เพื่อวิเคราะห์ความสัมพันธ์ระหว่างข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์และการปรับปรุงซอร์สโค้ดทางด้านการบำรุงรักษา ผู้วิจัยจึงได้ทำการสรุปข้อมูลของจำนวนข้อเสนอแนะของผู้ตรวจทานโค้ดที่ให้คำแนะนำเกี่ยวกับการบำรุงรักษาและข้อมูลของจำนวนข้อเสนอแนะที่ได้รับการปรับปรุงแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาในช่วงระยะเวลา 5 ปีของโครงการ Eclipse แสดงดังตารางที่ 4.14 และข้อมูลของโครงการ Qt แสดงดังตารางที่ 4.15

ตารางที่ 4.14 ผลลัพธ์จากการวิเคราะห์ที่เกิดขึ้นในโครงการ Eclipse ในช่วงระยะเวลา 5 ปี

ปี	โครงการ Eclipse	
	จำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์	จำนวนข้อเสนอแนะที่ได้รับการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษา
2012	4,906	3,123
2013	10,505	5,542
2014	15,854	9,104
2015	16,723	9,569
2016	16,628	9,637

ตารางที่ 4.15 ผลลัพธ์จากการวิเคราะห์ที่เกิดขึ้นในโครงการ Qt ในช่วงระยะเวลา 5 ปี

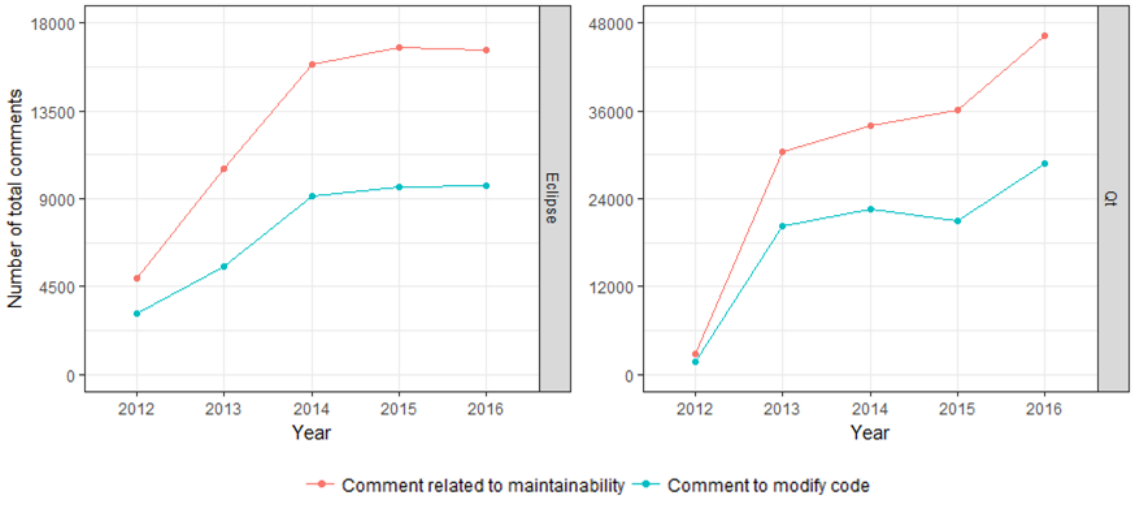
ปี	โครงการ Qt	
	จำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์	จำนวนข้อเสนอแนะที่ได้รับการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษา
2012	2,796	1,691
2013	30,415	20,277
2014	33,993	22,581
2015	36,167	21,035
2016	46,239	28,824

สัมประสิทธิ์สหสัมพันธ์ (Coefficient of Correlation) แบบ Pearson's product moment เป็นสถิติที่ใช้ในการหาค่าระดับความสัมพันธ์เชิงเส้น โดยจะมีค่าอยู่ระหว่าง -1.0 ถึง +1.0 โดยที่ค่าที่อยู่ใกล้ -1.0 หรือ +1.0 ถือว่ามีความสัมพันธ์กันมากที่สุด ส่วน 0 หมายความว่า ตัวแปรทั้งสองไม่มีความสัมพันธ์กันแม้แต่น้อย ส่วนเครื่องหมาย + หรือ – เป็นการบ่งบอกว่าความสัมพันธ์นั้นเป็นไปในทิศทางเดียวกันหรือตรงกันข้าม หรือก็คือ ตัวแปรหนึ่งมีค่าสูงอีกตัวแปรหนึ่งก็จะมีค่าสูง แต่ในกรณีที่ตัวแปรหนึ่งมีค่าลดลงอีกตัวแปรหนึ่งก็จะมีค่าลดลงตามไปด้วย สำหรับเกณฑ์การพิจารณาค่าที่บ่งบอกระดับของความสัมพันธ์จะอาศัยตามเกณฑ์ของ Hinkle และคณะ (Hinkle, et al., 2003) ดังนี้

ค่า r	**ระดับของความสัมพันธ์**
.90-1.00	มีความสัมพันธ์กันสูงมาก
.70-.90	มีความสัมพันธ์กันในระดับสูง
.50-.70	มีความสัมพันธ์กันในระดับปานกลาง
.30-.50	มีความสัมพันธ์กันในระดับต่ำ
.00-.30	มีความสัมพันธ์กันในระดับต่ำมาก

ผู้วิจัยได้นำผลลัพธ์จากการศึกษาข้างต้นมาทำการวิเคราะห์ค่าสัมประสิทธิ์สหสัมพันธ์เพียร์สัน โดยกำหนดให้ระดับความเชื่อมั่นอยู่ที่ 95% เพื่อหาความสัมพันธ์ระหว่างข้อเสนอแนะเกี่ยวกับการบำรุงรักษาและข้อเสนอแนะที่ได้รับการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาของโครงการโอเพนซอร์สทั้ง 2 โครงการ

ซึ่งผลลัพธ์ที่ได้จากการวิเคราะห์โดยใช้โปรแกรม R พบว่าค่าสัมประสิทธิ์สหสัมพันธ์เพียร์สันของโครงการ Eclipse และโครงการ Qt เท่ากับ 0.995 และ 0.993 ตามลำดับ โดยค่าสัมประสิทธิ์สหสัมพันธ์ดังกล่าวแสดงให้เห็นว่าจำนวนของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษากับจำนวนของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาซอฟต์แวร์มีความสัมพันธ์ระดับสูงมากไปในทิศทางเดียวกัน จึงอาจจะกล่าวได้ว่า เมื่อผู้ตรวจทานโค้ดของโครงการ Eclipse และโครงการ Qt ให้คำแนะนำเกี่ยวกับการบำรุงรักษาเพิ่มมากขึ้น นักพัฒนาซอฟต์แวร์ก็จะทำการแก้ไขปรับปรุงซอร์สโค้ดตามข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาเพิ่มมากขึ้นด้วยเช่นกัน ดังที่ปรากฏในรูปที่ 4.5 แสดงความสัมพันธ์ระหว่างข้อเสนอแนะเกี่ยวกับการบำรุงรักษาซอฟต์แวร์และการแก้ไขโค้ดตามคำแนะนำทางด้านการบำรุงรักษาซอฟต์แวร์ของโครงการ Eclipse และโครงการ Qt



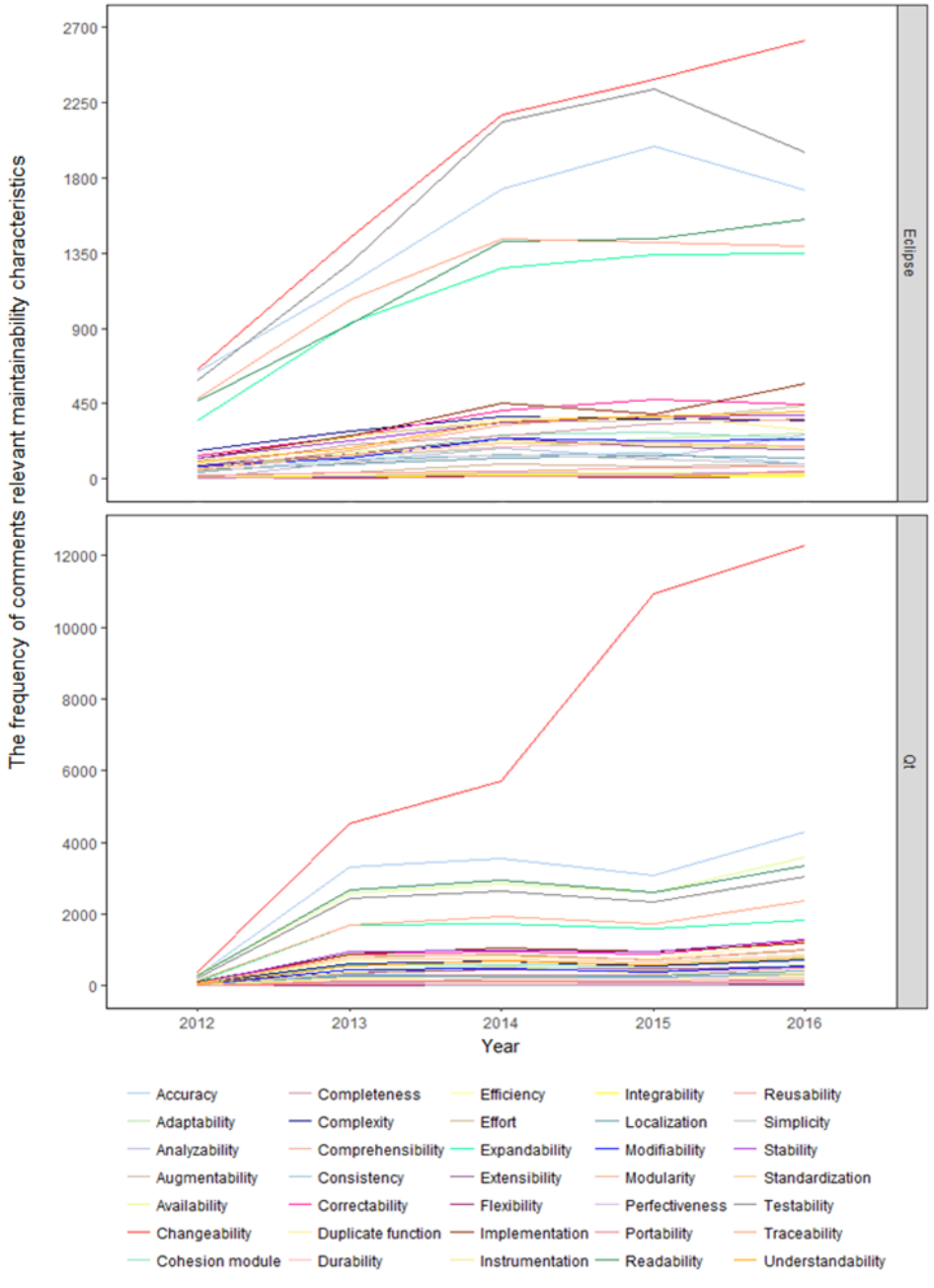
รูปที่ 4.5 กราฟแสดงความสัมพันธ์ระหว่างข้อเสนอแนะเกี่ยวกับการบำรุงรักษาและการแก้ไขโค้ดที่ได้รับคำแนะนำทางด้านการบำรุงรักษาโครงการ Eclipse และโครงการ Qt

4.2.5 แนวโน้มของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาและแนวโน้มของการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ของทั้ง 2 โครงการ

สำหรับหัวข้อนี้จะเป็นการแสดงให้เห็นถึงแนวโน้มของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์และแนวโน้มการแก้ไขโค้ดตามข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ในช่วงระยะเวลา 5 ปี เพื่อเป็นหลักฐานว่า ชุมชนนักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์สให้ความสำคัญกับการบำรุงรักษาเพิ่มขึ้นหรือลดลงในแต่ละปี นอกจากนี้ผู้วิจัยได้ทำการวิเคราะห์ความสัมพันธ์ระหว่างจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษากับปีที่มีการให้คำแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ และวิเคราะห์ความสัมพันธ์ระหว่างจำนวนการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษากับปีที่มีการแนะนำให้นักพัฒนาปรับปรุงแก้ไขโค้ดของทั้ง 2 โครงการ ในช่วงปี ค.ศ. 2012 – 2016 อีกทั้งยังใช้การวิเคราะห์การถดถอยเพื่อพยากรณ์จำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาและจำนวนการแก้ไขโค้ดทางด้านการบำรุงรักษาที่จะเกิดขึ้นในปี ค.ศ. 2018

1) แนวโน้มของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์

การศึกษาแนวโน้มการให้คำแนะนำของผู้ตรวจทานโค้ดที่มีต่อการบำรุงรักษาซอฟต์แวร์ แสดงให้เห็นว่า ผู้ตรวจทานโค้ดในชุมชนโอเพนซอร์สโดยส่วนใหญ่มีการให้คำแนะนำให้ปรับปรุงแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ในแต่ละคุณลักษณะเพิ่มมากขึ้นเรื่อย ๆ ตั้งแต่ปี ค.ศ. 2012 และมีแนวโน้มที่จะเพิ่มขึ้นอีกในอนาคตแสดงดังรูปที่ 4.6



รูปที่ 4.6 แนวโน้มของข้อเสนอแนะเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ทั้ง 35 คุณลักษณะ

จากรูปที่ 4.6 สามารถแสดงให้เห็นว่า จำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์จะเพิ่มสูงขึ้นในอนาคต ดังนั้นผู้วิจัยจึงใช้การวิเคราะห์การถดถอย (Regression Analysis) ซึ่งเป็นการวิเคราะห์ความสัมพันธ์ทางสถิติที่ใช้หาความสัมพันธ์ระหว่างตัวแปรตั้งแต่ 2 ตัวขึ้นไป เพื่อใช้ทดสอบความมีนัยสำคัญทางสถิติของความสัมพันธ์ โดยการศึกษาความสัมพันธ์นี้มีวัตถุประสงค์เพื่อตรวจสอบว่า ตัวแปรอิสระ (Independent Variable หรือ Predictor) และตัวแปรตาม (Dependent Variable) มีความสัมพันธ์กันมากน้อยเท่าใด และนำไปสู่การสร้างสมการถดถอยที่ใช้เพื่อพยากรณ์ตัวแปรที่ผู้วิจัยสนใจ

ในการวิเคราะห์ความสัมพันธ์ระหว่างจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาและปีที่มีการให้คำแนะนำเกี่ยวกับการปรับปรุงซอร์สโค้ดทางด้านการบำรุงรักษา ผู้วิจัยได้ทำการสรุปข้อมูลจากการรวบรวมข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาที่เกิดขึ้นในช่วงระยะเวลา 5 ปีของโครงการ Eclipse และโครงการ Qt ปรากฏดังตารางที่ 4.16 และตารางที่ 4.17 ตามลำดับ

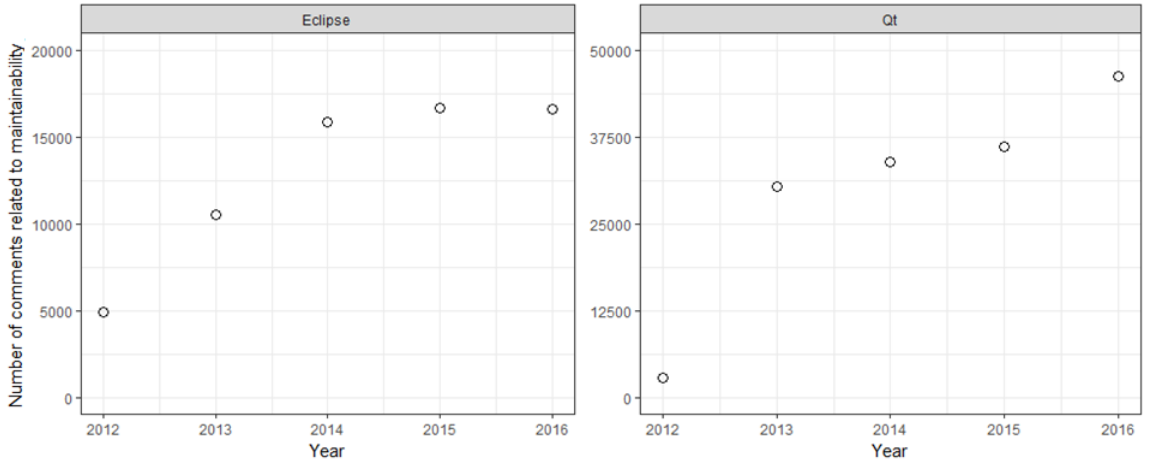
ตารางที่ 4.16 การเก็บรวบรวมข้อมูลจากข้อเสนอแนะที่เกิดขึ้นในโครงการ Eclipse ระยะเวลา 5 ปี

ปี	ปีที่มีการให้คำแนะนำ (x)	จำนวนของข้อเสนอแนะที่ได้รับคำแนะนำทางด้านการบำรุงรักษา (y)
1	2012	4,906
2	2013	10,505
3	2014	15,854
4	2015	16,723
5	2016	16,628

ตารางที่ 4.17 การเก็บรวบรวมข้อมูลจากข้อเสนอแนะที่เกิดขึ้นในโครงการ Qt ระยะเวลา 5 ปี

ปี	ปีที่มีการให้คำแนะนำ (x)	จำนวนของข้อเสนอแนะที่ได้รับคำแนะนำทางด้านการบำรุงรักษา (y)
1	2012	2,796
2	2013	30,415
3	2014	33,993
4	2015	36,167
5	2016	46,239

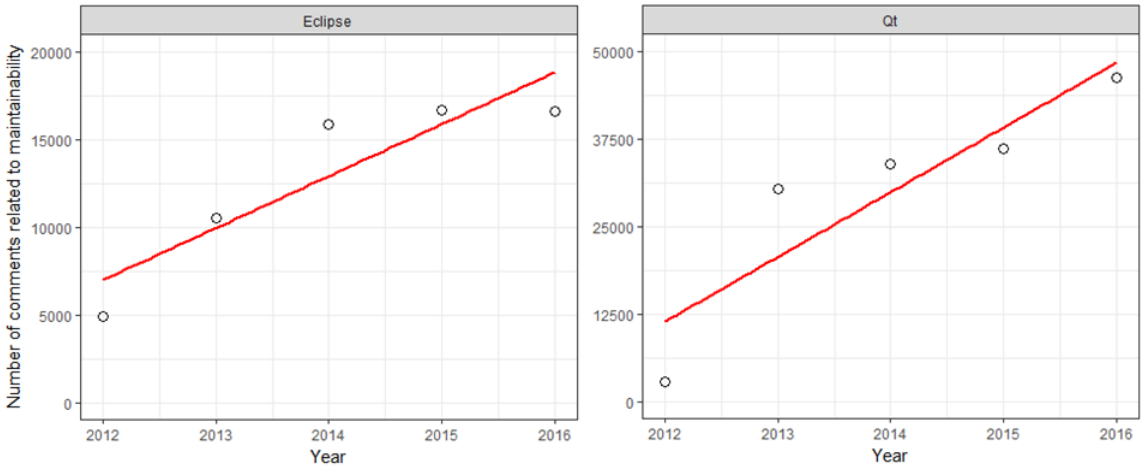
จากตารางที่ 4.16 และตารางที่ 4.17 แสดงข้อมูลของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาในแต่ละโครงการจะมีอยู่ด้วยกัน 5 ชุดข้อมูล เมื่อผู้วิจัยได้นำมาเขียนแผนภาพการกระจายแสดงความสัมพันธ์ของทั้ง 2 ตัวแปร โดยกำหนดให้แกนแนวตั้งแสดงจำนวนของข้อเสนอแนะเกี่ยวกับการบำรุงรักษา (y) และแกนแนวนอนแสดงปีที่ให้คำแนะนำทางด้านการบำรุงรักษาของผู้ตรวจทานโค้ด (x) แสดงดังรูปที่ 4.7



รูปที่ 4.7 แผนภาพกระจายแสดงความสัมพันธ์ระหว่างจำนวนของข้อเสนอแนะเกี่ยวกับการบำรุงรักษากับปีที่ให้คำแนะนำทางด้านการบำรุงรักษาของผู้ตรวจทานโค้ด

จากรูปที่ 4.7 เป็นการแสดงความสัมพันธ์ระหว่างจำนวนของข้อเสนอแนะเกี่ยวกับการบำรุงรักษากับปีที่ให้คำแนะนำทางด้านการบำรุงรักษาของผู้ตรวจทานโค้ด จากข้อมูลดังกล่าวทำให้สามารถวิเคราะห์ได้ว่า ปีที่มีการให้คำแนะนำในการแก้ไขปรับปรุงโค้ดเกี่ยวกับการบำรุงรักษาและจำนวนของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาที่ได้รับจากผู้ตรวจทานโค้ดมีความสัมพันธ์ระดับสูงไปในทิศทางเดียวกัน จึงอาจจะกล่าวได้ว่าเมื่อจำนวนปีเพิ่มมากขึ้น จำนวนข้อเสนอแนะเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ของทั้ง 2 โครงการก็จะมีจำนวนเพิ่มมากขึ้นด้วยเช่นกัน

ในงานวิจัยนี้ ผู้วิจัยต้องการพยากรณ์แนวโน้มของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาในอนาคต ดังนั้นผู้วิจัยจึงได้ทำการวิเคราะห์การถดถอย โดยกำหนดให้ปีที่ให้คำแนะนำทางด้านการบำรุงรักษาของผู้ตรวจทานโค้ดเป็นตัวแปรอิสระ และจำนวนของข้อเสนอแนะเกี่ยวกับการบำรุงรักษาเป็นตัวแปรตาม



รูปที่ 4.8 เส้นกราฟการถดถอยที่แสดงแนวโน้มการให้ข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษา

จากรูปที่ 4.8 แสดงให้เห็นว่าข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาและจำนวนปีที่มีการให้คำแนะนำมีความสัมพันธ์เชิงเส้นตรงกัน เรียกว่า การวิเคราะห์การถดถอยเชิงเส้นอย่างง่าย (Simple Linear Regression Analysis) และเมื่อผู้วิจัยกำหนดค่าของตัวแปรอิสระ เส้นตรงดังกล่าวนี้จะใช้ในการคาดคะเนค่าของตัวแปรตามที่ต้องการพยากรณ์ หรือเรียกกันว่า เส้นพยากรณ์ (Prediction Line) หรือ เส้นถดถอย (Regression Line) ดังนั้นเส้นถดถอยที่ใช้คาดคะเนตัวแปรตามเมื่อรู้ค่าตัวแปรอิสระที่มีความสัมพันธ์ในเชิงเส้นตรงเรียกว่า สมการถดถอยเชิงเส้นตรง (Linear Regression Equation)

รูปแบบของสมการเส้นตรงมีดังนี้

$$\hat{y} = a + bx_i$$

จากสมการถดถอยเชิงเส้นตรง $\hat{y}$ = a + bx แทนค่าของ a และ b โดยที่

a: ค่าที่เส้นกราฟถดถอยตัดกับแกน Y (Y-intercept)

b: ค่าความชันของเส้นกราฟ (Slope of the line) ซึ่งแสดงถึงอัตราการเปลี่ยนแปลงของ Y เมื่อ X เปลี่ยนแปลง เรียกส่วนนี้ว่า สัมประสิทธิ์การถดถอย (Regression Coefficient) หรือสัมประสิทธิ์การพยากรณ์

การวิเคราะห์ข้อมูลของปีที่มีให้คำแนะนำในการแก้ไขปรับปรุงโค้ดเกี่ยวกับการบำรุงรักษากับจำนวนของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาที่เกิดขึ้นในโครงการ Eclipse และโครงการ Qt ผู้วิจัยได้ใช้โปรแกรม R ในการวิเคราะห์การถดถอย โดยกำหนดให้ระดับความเชื่อมั่นอยู่ที่ 95% และมีการสมมติฐานทางสถิติดังนี้

H0 : จำนวนของข้อเสนอแนะเกี่ยวกับการบำรุงรักษาไม่มีความสัมพันธ์กับปีที่มีให้คำแนะนำในการแก้ไขปรับปรุงโค้ดเกี่ยวกับการบำรุงรักษา

H1 : จำนวนของข้อเสนอแนะเกี่ยวกับการบำรุงรักษามีความสัมพันธ์กับปีที่มีให้คำแนะนำในการแก้ไขปรับปรุงโค้ดเกี่ยวกับการบำรุงรักษา

1.1) ผลลัพธ์ที่ได้จากการวิเคราะห์การถดถอยของโครงการ Eclipse

ผลลัพธ์ที่ได้จากการวิเคราะห์โดยใช้โปรแกรม R จะได้ค่า p-value = 0.0334 ซึ่งค่า p-value ที่ได้มีค่าน้อยกว่า 0.05 จึงทำการปฏิเสธ H0 ดังนั้นสามารถสรุปได้ว่า จำนวนของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษากับปีที่มีการให้คำแนะนำในการแก้ไขปรับปรุงโค้ดทางด้านการบำรุงรักษามีความสัมพันธ์กันอย่างมีนัยสำคัญทางสถิติ

ผู้วิจัยได้นำค่าสัมประสิทธิ์การพยากรณ์แทนค่าในสมการถดถอยเชิงเส้น ซึ่งจากสมการถดถอยระหว่างปีที่มีการให้คำแนะนำในการแก้ไขปรับปรุงโค้ดทางด้านการบำรุงรักษากับจำนวนของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาในโครงการ Eclipse ได้เป็น

$$\hat{y} = -5961003.6 + 2966.2\ (X)$$

เพื่อใช้พยากรณ์อัตราการให้ข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาที่จะเกิดขึ้นในปี ค.ศ. 2018 โดยนำปีที่ต้องการจะทำนายแทนค่า x ในสมการ จะได้ค่า $\hat{y}$ ดังนี้

$$\hat{y} = -5961003.6 + 2966.2\ (2018)$$

ผลลัพธ์ที่ได้จากการวิเคราะห์ถดถอยเชิงเส้นตรง คือ $-5961003.6 + 2966.2\ (2018) = 24{,}788$

จากการวิเคราะห์การถดถอยเชิงเส้นตรง เพื่อทำนายแนวโน้มของจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาของโครงการ Eclipse ที่จะเกิดขึ้นในปี ค.ศ. 2018 ได้เท่ากับ 24,788

1.2) ผลลัพธ์ที่ได้จากการวิเคราะห์การถดถอยของโครงการ Qt

ผลลัพธ์ที่ได้จากการวิเคราะห์โดยใช้โปรแกรม R จะได้ค่า p-value = 0.03701 ซึ่งค่า p-value ที่ได้มีค่าน้อยกว่า 0.05 จึงทำการปฏิเสธ H0 ดังนั้นสามารถสรุปได้ว่า จำนวนของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษากับปีที่มีการให้คำแนะนำในการแก้ไขปรับปรุงโค้ดทางด้านการบำรุงรักษามีความสัมพันธ์กันอย่างมีนัยสำคัญทางสถิติ

ผู้วิจัยได้นำค่าสัมประสิทธิ์การพยากรณ์แทนค่าในสมการถดถอยเชิงเส้น ซึ่งจากสมการถดถอยระหว่างปีที่มีการให้คำแนะนำในการแก้ไขปรับปรุงโค้ดทางด้านการบำรุงรักษากับจำนวนของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาในโครงการ Qt ได้เป็น

$$\hat{y} = -18627371 + 9264\ (X)$$

เพื่อใช้พยากรณ์อัตราการให้ข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาที่จะเกิดขึ้นในปี ค.ศ. 2018 โดยนำปีที่ต้องการจะทำนายแทนค่า x ในสมการ จะได้ค่า $\hat{y}$ ดังนี้

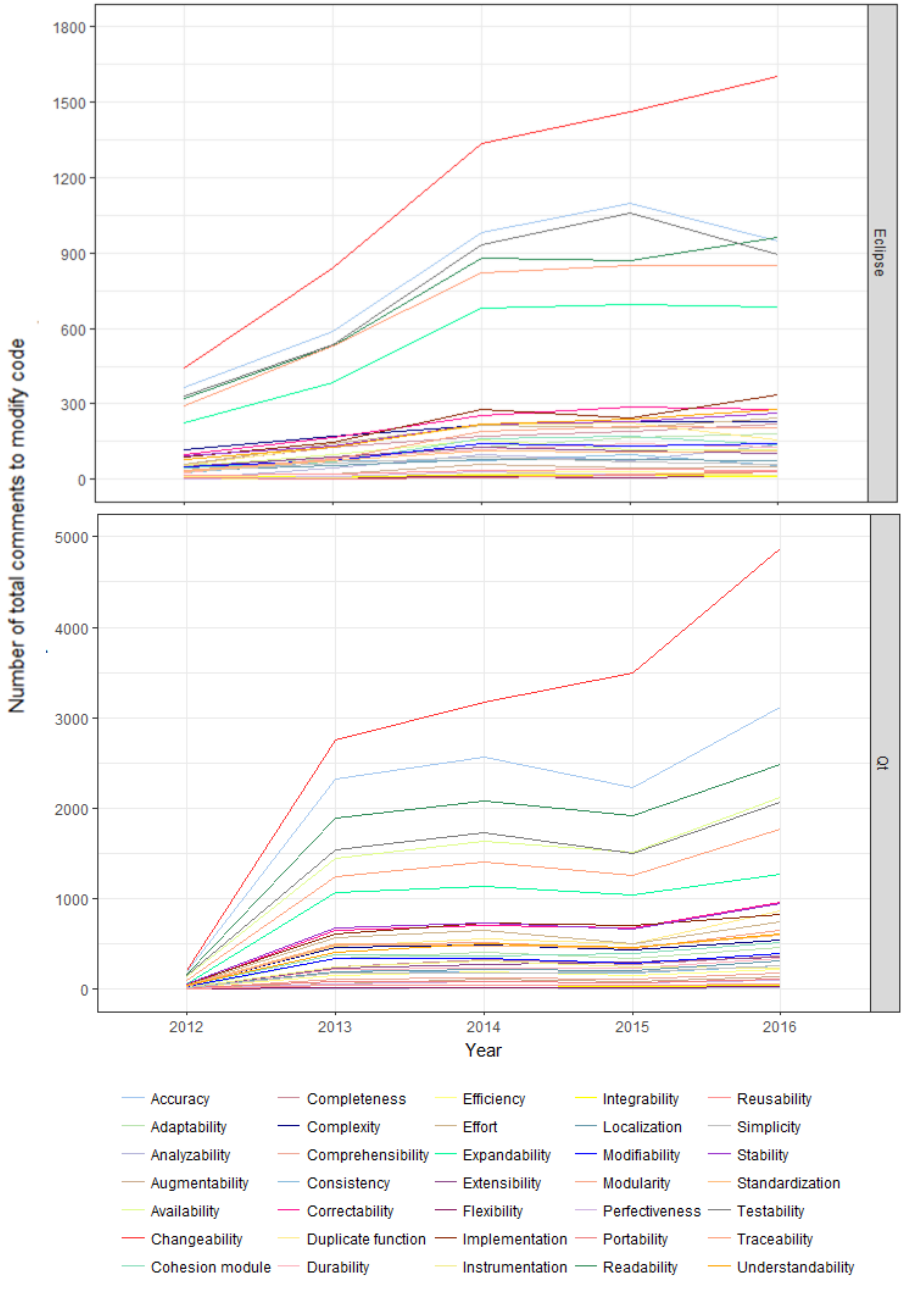
$$\hat{y} = -18627371 + 9264\ (2018)$$

ผลลัพธ์ที่ได้จากการวิเคราะห์ถดถอยเชิงเส้นตรง คือ -18627371 + 9264 (2018) = 67,381

จากการวิเคราะห์การถดถอยเชิงเส้นตรง เพื่อทำนายแนวโน้มของจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาของโครงการ Qt ที่จะเกิดขึ้นในปี ค.ศ. 2018 ได้เท่ากับ 67,381

2) แนวโน้มของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดทางด้านการบำรุงรักษา

จากการศึกษาแนวโน้มความสนใจของนักพัฒนาที่มีต่อการบำรุงรักษาซอฟต์แวร์แสดงให้เห็นว่า นักพัฒนาในชุมชนโอเพนซอร์สให้ความสำคัญกับการบำรุงรักษาเพิ่มมากขึ้น สังเกตได้จากจำนวนข้อเสนอแนะทางด้านการบำรุงรักษาที่ได้รับการแก้ไขในแต่ละโครงการที่มีจำนวนเพิ่มขึ้นตั้งแต่ปี ค.ศ. 2012 และมีแนวโน้มที่จะเพิ่มขึ้นอีกในอนาคตแสดงดังรูปที่ 4.9



รูปที่ 4.9 แนวโน้มของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาทั้ง 35 คุณลักษณะ

จากจำนวนข้อเสนอแนะที่ได้รับการแก้ไขโค้ดที่แสดงดังรูปที่ 4.9 ทำให้ผู้วิจัยเชื่อว่าเมื่อมีการแก้ไขหรือปรับปรุงซอฟต์แวร์บ่อยครั้งก็ยิ่งทำให้ระบบมีแนวโน้มที่จะเกิดความซับซ้อนและมีความเสี่ยงที่จะเกิดข้อผิดพลาดในส่วนของการแสดงผลลัพธ์ รวมไปถึงอาจทำให้เกิดข้อบกพร่องของระบบมากขึ้นตามไปด้วย โดยสังเกตได้จากเส้นกราฟการบำรุงรักษาประเภท Changeability และ Accuracy ของทั้ง 2 โครงการ ที่แสดงให้เห็นว่านักพัฒนาได้ทำการแก้ไขโค้ดมากกว่าประเภทอื่น ๆ เพื่อตรวจสอบว่าอัตราการแก้ไขโค้ดทางด้านการบำรุงรักษาจะเพิ่มสูงขึ้นในอนาคต ผู้วิจัยได้ใช้การวิเคราะห์การถดถอยในการพยากรณ์ค่าของตัวแปรเช่นเดียวกับหัวข้อที่ 1) แนวโน้มของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์

ในการวิเคราะห์ความสัมพันธ์ระหว่างจำนวนการแก้ไขโค้ดตามข้อเสนอแนะเกี่ยวกับการบำรุงรักษาและปีที่มีการแก้ไขโค้ดทางด้านการบำรุงรักษา ผู้วิจัยได้ทำการสรุปข้อมูลจากการรวบรวมข้อเสนอแนะที่ได้รับการแก้ไขปรับปรุงโค้ดเกี่ยวกับการบำรุงรักษาที่เกิดขึ้นในช่วงระยะเวลา 5 ปี ของโครงการ Eclipse และโครงการ Qt ปรากฏดังตารางที่ 4.18 และตารางที่ 4.19 ตามลำดับ

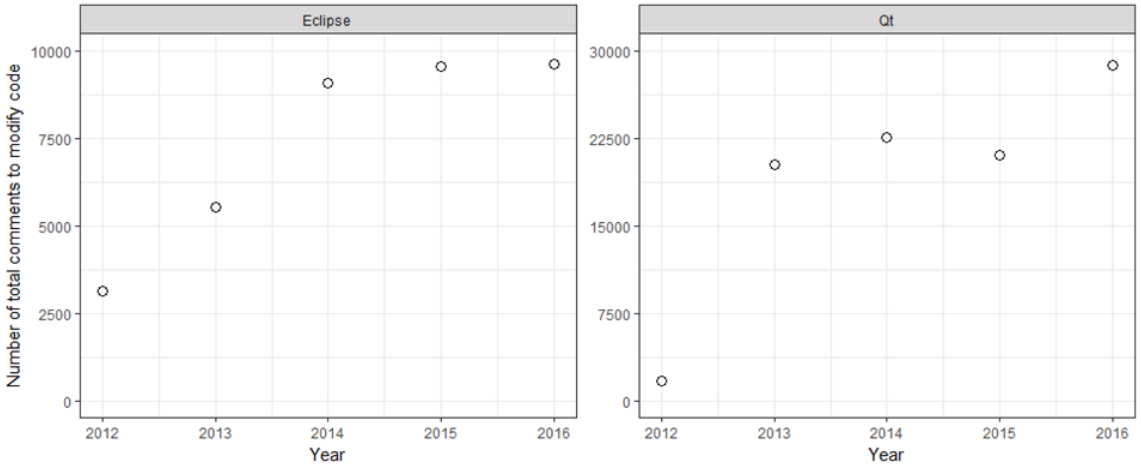
ตารางที่ 4.18 การเก็บรวบรวมข้อเสนอแนะที่ได้รับการแก้ไขโค้ดในโครงการ Eclipse ระยะเวลา 5 ปี

ปี	ปีที่มีแก้ไขโค้ด (x)	จำนวนของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดทางด้านการบำรุงรักษา (y)
1	2012	3,123
2	2013	5,542
3	2014	9,104
4	2015	9,569
5	2016	9,637

ตารางที่ 4.19 การเก็บรวบรวมข้อเสนอแนะที่ได้รับการแก้ไขโค้ดในโครงการ Qt ระยะเวลา 5 ปี

ปี	ปีที่มีแก้ไขโค้ด (x)	จำนวนของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดทางด้านการบำรุงรักษา (y)
1	2012	1,691
2	2013	20,227
3	2014	22,581
4	2015	21,035
5	2016	28,824

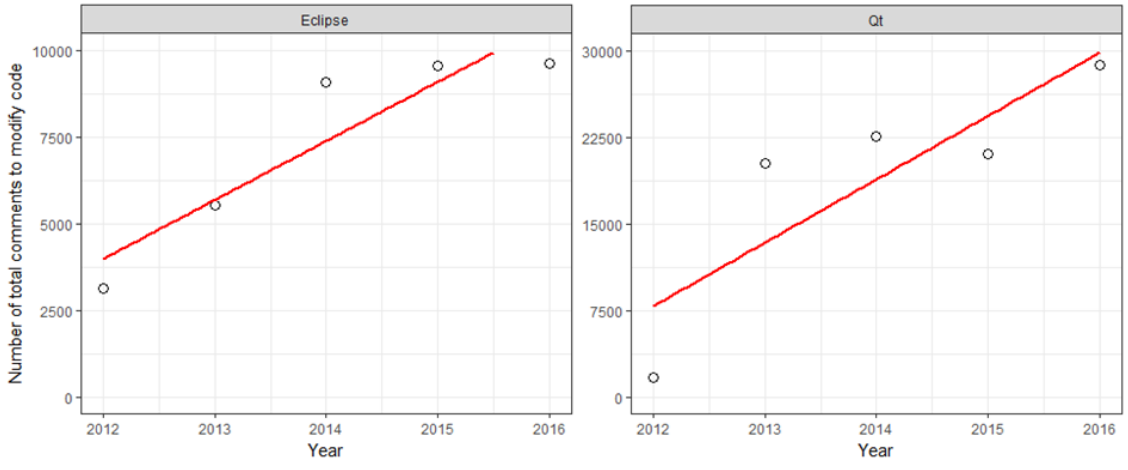
จากตารางที่ 4.18 และตารางที่ 4.19 แสดงข้อมูลของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาในแต่ละโครงการจะมีอยู่ด้วยกัน 5 ชุดข้อมูล เมื่อผู้วิจัยได้นำมาเขียนแผนภาพการกระจายแสดงความสัมพันธ์ของทั้ง 2 ตัวแปร โดยกำหนดให้แกนแนวตั้งแสดงจำนวนของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษา (y) และแกนแนวนอนแสดงปีที่นักพัฒนาซอฟต์แวร์ได้ทำการแก้ไขโค้ด (x) ดังรูปที่ 4.10 แสดงความสัมพันธ์ระหว่างจำนวนของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษากับปีที่นักพัฒนาซอฟต์แวร์ทำการแก้ไขโค้ด จากข้อมูลดังกล่าวทำให้สามารถวิเคราะห์ได้ว่า จำนวนปีและจำนวนข้อเสนอแนะที่ได้รับการแก้ไขโค้ดมีความสัมพันธ์ระดับสูงไปในทิศทางเดียวกัน ดังนั้นจึงสรุปได้ว่า เมื่อปีที่มีการแนะนำให้นักพัฒนาทำการปรับปรุงแก้ไขโค้ดทางด้านการบำรุงรักษาเพิ่มมากขึ้น จำนวนของการแก้ไขปรับปรุงโค้ดตามข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาก็จะเพิ่มขึ้นตามไปด้วย



รูปที่ 4.10 แผนภาพกระจายแสดงความสัมพันธ์ระหว่างจำนวนการแก้ไขโค้ดตามข้อเสนอแนะเกี่ยวกับการบำรุงรักษากับปีที่มีการแนะนำให้นักพัฒนาทำการปรับปรุงแก้ไขโค้ด

เมื่อผู้วิจัยได้ทำการพยากรณ์แนวโน้มของจำนวนการแก้ไขโค้ดตามคำแนะนำเกี่ยวกับการบำรุงรักษาในอนาคตโดยใช้การวิเคราะห์การถดถอย โดยกำหนดให้ปีที่มีการแนะนำให้นักพัฒนาทำการปรับปรุงแก้ไขโค้ดเป็นตัวแปรอิสระ และจำนวนการแก้ไขโค้ดตามข้อเสนอแนะเกี่ยวกับการบำรุงรักษาเป็นตัวแปรตาม โดยรูปที่ 4.11 แสดงให้เห็นว่าการแก้ไขปรับปรุงซอร์สโค้ดเกี่ยวกับการบำรุงรักษาและจำนวนปีที่มีการให้คำแนะนำมีความสัมพันธ์เชิงเส้นตรงกัน โดยตัวแปรทั้งสองมีความสัมพันธ์ในเชิงเส้นตรงกันจะถูกเรียกว่า การวิเคราะห์การถดถอยเชิงเส้นอย่างง่าย และเมื่อผู้วิจัยกำหนดค่าของตัวแปรอิสระ เส้นตรงดังกล่าวนี้จะใช้ในการคาดคะเนค่าของตัวแปรตามที่ต้องการพยากรณ์ หรือเรียกกันว่า

เส้นพยากรณ์ ดังนั้นเส้นถดถอยที่ใช้คาดคะเนตัวแปรตามเมื่อรู้ค่าตัวแปรอิสระที่มีความสัมพันธ์ในเชิงเส้นตรงเรียกว่า สมการถดถอยเชิงเส้นตรง โดยมีรูปแบบของสมการเส้นตรง คือ $\hat{y} = a + bx_i$



รูปที่ 4.11 กราฟการถดถอยที่แสดงแนวโน้มของจำนวนการแก้ไขโค้ดทางด้านการบำรุงรักษาซอฟต์แวร์ในอนาคต

การวิเคราะห์ข้อมูลของปีที่มีการแนะนำให้นักพัฒนาทำการปรับปรุงแก้ไขโค้ดเกี่ยวกับจำนวนการแก้ไขโค้ดตามข้อเสนอแนะเกี่ยวกับการบำรุงรักษาที่เกิดขึ้นในโครงการ Eclipse และโครงการ Qt ผู้วิจัยจึงใช้โปรแกรม R ในการวิเคราะห์การถดถอย โดยกำหนดให้ระดับความเชื่อมั่นอยู่ที่ 95% และมีการสมมติฐานทางสถิติดังนี้

H0 : จำนวนของข้อเสนอแนะทางด้านการบำรุงรักษาที่ได้รับการแก้ไขโค้ดไม่มีความสัมพันธ์กับปีที่มีการแนะนำให้นักพัฒนาทำการปรับปรุงแก้ไขโค้ด

H1 : จำนวนของข้อเสนอแนะทางด้านการบำรุงรักษาที่ได้รับการแก้ไขโค้ดมีความสัมพันธ์กับปีที่มีการแนะนำให้นักพัฒนาทำการปรับปรุงแก้ไขโค้ด

2.1) ผลลัพธ์ที่ได้จากการวิเคราะห์การถดถอยของโครงการ Eclipse

ผลลัพธ์ที่ได้จากการวิเคราะห์โดยใช้โปรแกรม R จะได้ค่า p-value = 0.02678 ซึ่งค่า p-value ที่ได้มีค่าน้อยกว่า 0.05 จึงทำการปฏิเสธ H0 ดังนั้นสามารถสรุปได้ว่า จำนวนของข้อเสนอแนะทางด้านการบำรุงรักษาที่ได้รับการแก้ไขโค้ดกับปีที่มีการแนะนำให้นักพัฒนาทำการปรับปรุงแก้ไขโค้ดมีความสัมพันธ์กันอย่างมีนัยสำคัญทางสถิติ

ผู้วิจัยได้นำค่าสัมประสิทธิ์การพยากรณ์แทนค่าในสมการถดถอยเชิงเส้น ซึ่งจากสมการถดถอยระหว่างปีที่มีการแนะนำให้ทำการแก้ไขปรับปรุงโค้ดทางด้านการบำรุงรักษากับจำนวนของการแก้ไขโค้ดตามข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาในโครงการ Eclipse ได้เป็น

$$\hat{y} = -3427482.0 + 1705.5\ (X)$$

เพื่อใช้พยากรณ์อัตราการแก้ไขโค้ดตามข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาที่จะเกิดขึ้นในปี ค.ศ. 2018 โดยนำปีที่ต้องการจะทำนายแทนค่า x ในสมการ จะได้ค่า $\hat{y}$ ดังนี้

$$\hat{y} = -3427482.0 + 1705.5\ (2018)$$

ผลลัพธ์ที่ได้จากการวิเคราะห์ถดถอยเชิงเส้นตรง -3427482.0 + 1705.5 (2018) = 14,217 คือ จำนวนการแก้ไขโค้ดตามข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาของโครงการ Eclipse ที่จะเกิดขึ้นในปี ค.ศ. 2018

2.2) ผลลัพธ์ที่ได้จากการวิเคราะห์การถดถอยของโครงการ Qt

ผลลัพธ์ที่ได้จากการวิเคราะห์โดยใช้โปรแกรม R จะได้ค่า p-value = 0.06514 ซึ่งค่า p-value ที่ได้มีค่ามากกว่า 0.05 จึงทำการยอมรับ H0 ดังนั้นสามารถสรุปได้ว่า จำนวนของข้อเสนอแนะทางด้านการบำรุงรักษาที่ได้รับการแก้ไขโค้ดกับปีที่มีการแนะนำให้นักพัฒนาทำการปรับปรุงแก้ไขโค้ดไม่มีความสัมพันธ์กันอย่างมีนัยสำคัญทางสถิติ

ผู้วิจัยได้นำค่าสัมประสิทธิ์การพยากรณ์แทนค่าในสมการถดถอยเชิงเส้น ซึ่งจากสมการถดถอยระหว่างปีที่มีการแนะนำให้ทำการแก้ไขปรับปรุงโค้ดทางด้านการบำรุงรักษากับจำนวนของการแก้ไขโค้ดตามข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาในโครงการ Qt ได้เป็น

$$\hat{y} = -11062952 + 5502\ (X)$$

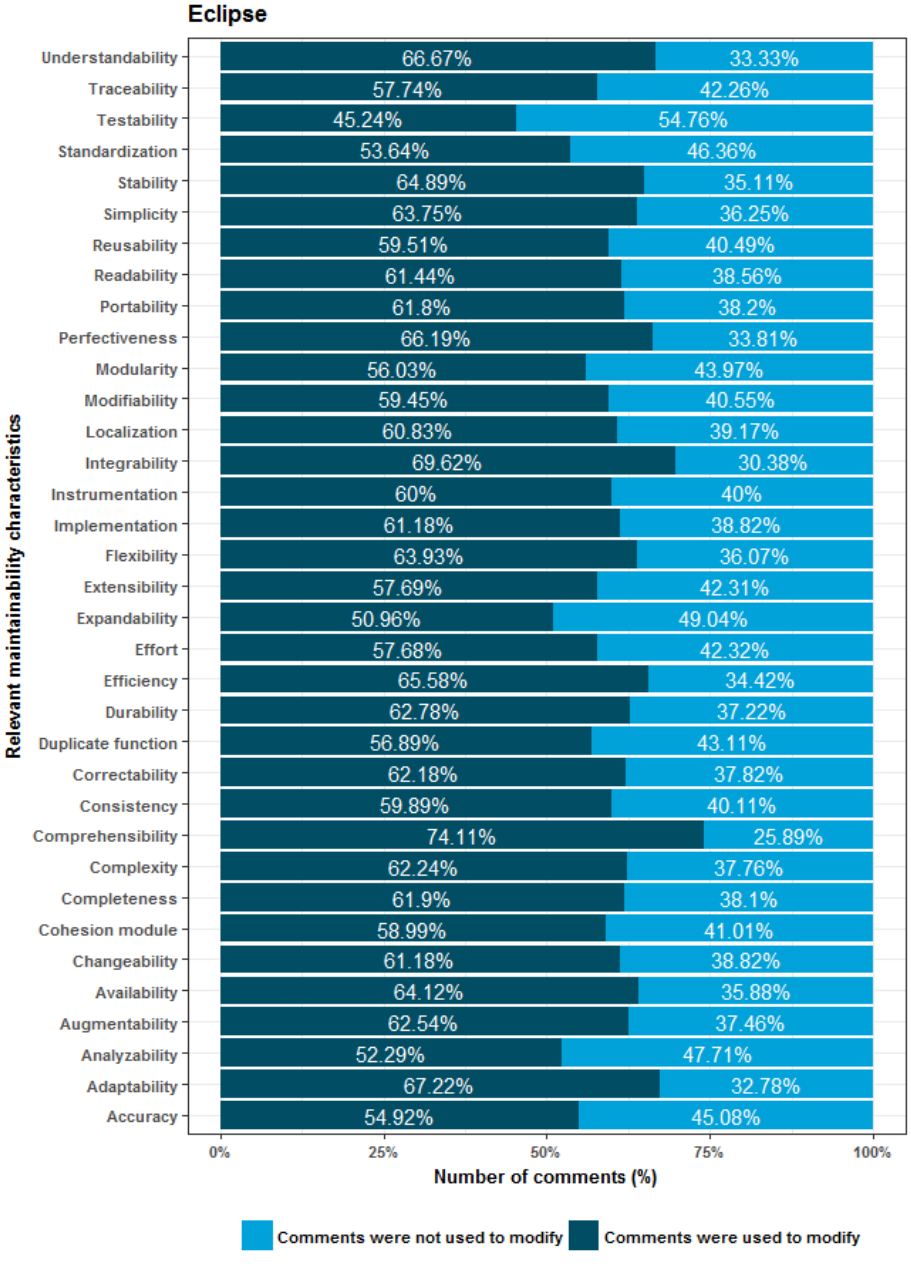
เพื่อใช้พยากรณ์อัตราการแก้ไขโค้ดตามข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาที่จะเกิดขึ้นในปี ค.ศ. 2018 โดยนำปีที่ต้องการจะทำนายแทนค่า x ในสมการ จะได้ค่า y ดังนี้

$$\hat{y} = -11062952 + 5502\ (2018)$$

ผลลัพธ์ที่ได้จากการวิเคราะห์ถดถอยเชิงเส้นตรง -11062952 + 5502 (2018) = 40,084 คือ จำนวนการแก้ไขโค้ดตามข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาของโครงการ Qt ที่จะเกิดขึ้นในปี ค.ศ. 2018

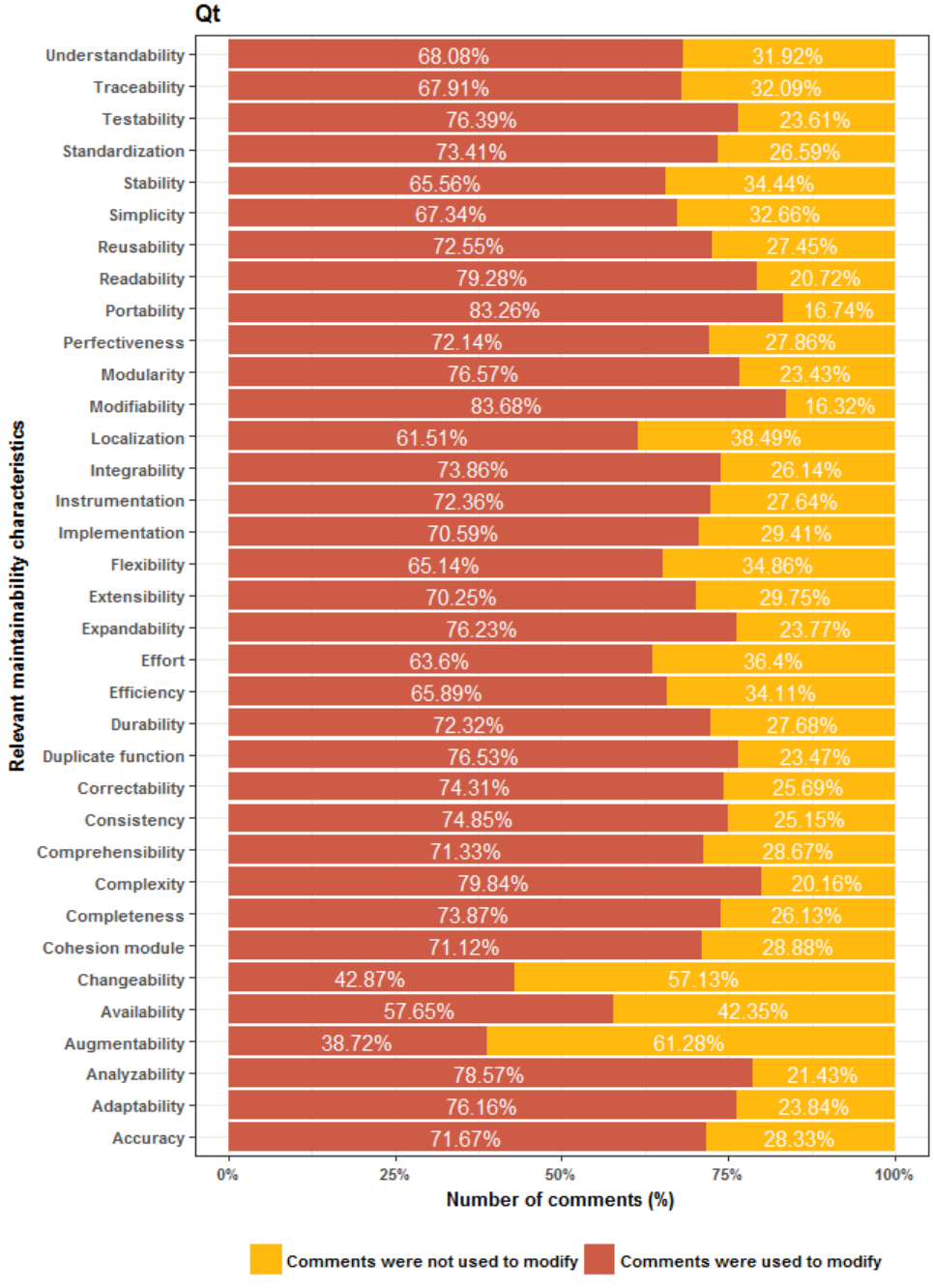
3) สรุปผลระหว่างแนวโน้มของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาและแนวโน้มการแก้ไขโค้ดตามข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์

เมื่อนำข้อมูลของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาและการแก้ไขโค้ดทางด้านการบำรุงรักษามาทำการแจกแจงเป็นจำนวนร้อยละดังรูปที่ 4.12 และรูปที่ 4.13 เป็นการแสดงอัตราส่วนของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาทั้ง 35 ประเภทของโครงการ Eclipse และโครงการ Qt



รูปที่ 4.12 อัตราส่วนของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดทางด้านการบำรุงรักษาทั้ง 35 ประเภท ในโครงการ Eclipse

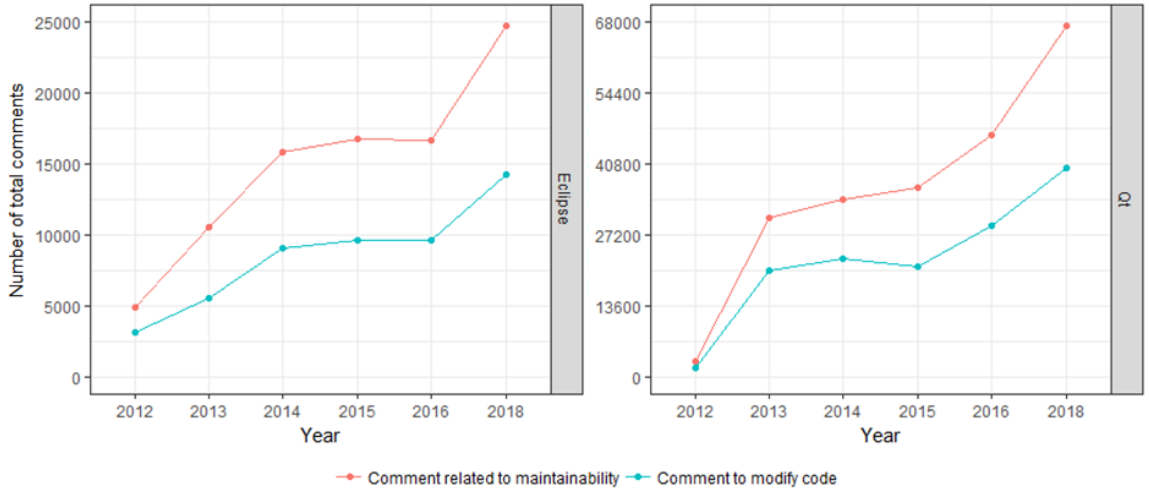
ซึ่งผลลัพธ์ที่แสดงดังรูปทำให้สามารถวิเคราะห์ได้ว่า นักพัฒนาซอฟต์แวร์ในโครงการ Eclipse ส่วนใหญ่ได้ทำการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ประมาณ 50-60% ถือว่าเป็นจำนวนร้อยละที่อยู่ในระดับปานกลาง เมื่อเทียบกับจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ทั้งหมด



รูปที่ 4.13 อัตราส่วนของข้อเสนอแนะที่ได้รับการแก้ไขโค้ดทางด้านการบำรุงรักษาทั้ง 35 ประเภท ในโครงการ Qt

ส่วนนักพัฒนาซอฟต์แวร์ในโครงการ Qt ได้ทำการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ประมาณ 60-70% ซึ่งถือว่าเป็นจำนวนที่มากพอสมควร เมื่อเทียบกับจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ทั้งหมด แต่จากผลลัพธ์จะสังเกตได้ว่าการบำรุงรักษาซอฟต์แวร์บางคุณลักษณะที่มีการแก้ไขโค้ดน้อยกว่า 50% คือ Changeability และ Augmentability ดังนั้นผลลัพธ์จากรูปที่ 4.12 และรูปที่ 4.13 จึงสามารถสรุปได้ว่า แนวโน้มของข้อเสนอแนะที่มีการแก้ไขปรับปรุงซอร์สโค้ดตามคำแนะนำทางด้านการบำรุงรักษาในช่วงระยะเวลา 5 ปี (ค.ศ. 2012 - 2016) มีจำนวนมากกว่า 50% เมื่อเทียบกับจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาทั้งหมดของโครงการ Eclipse และโครงการ Qt

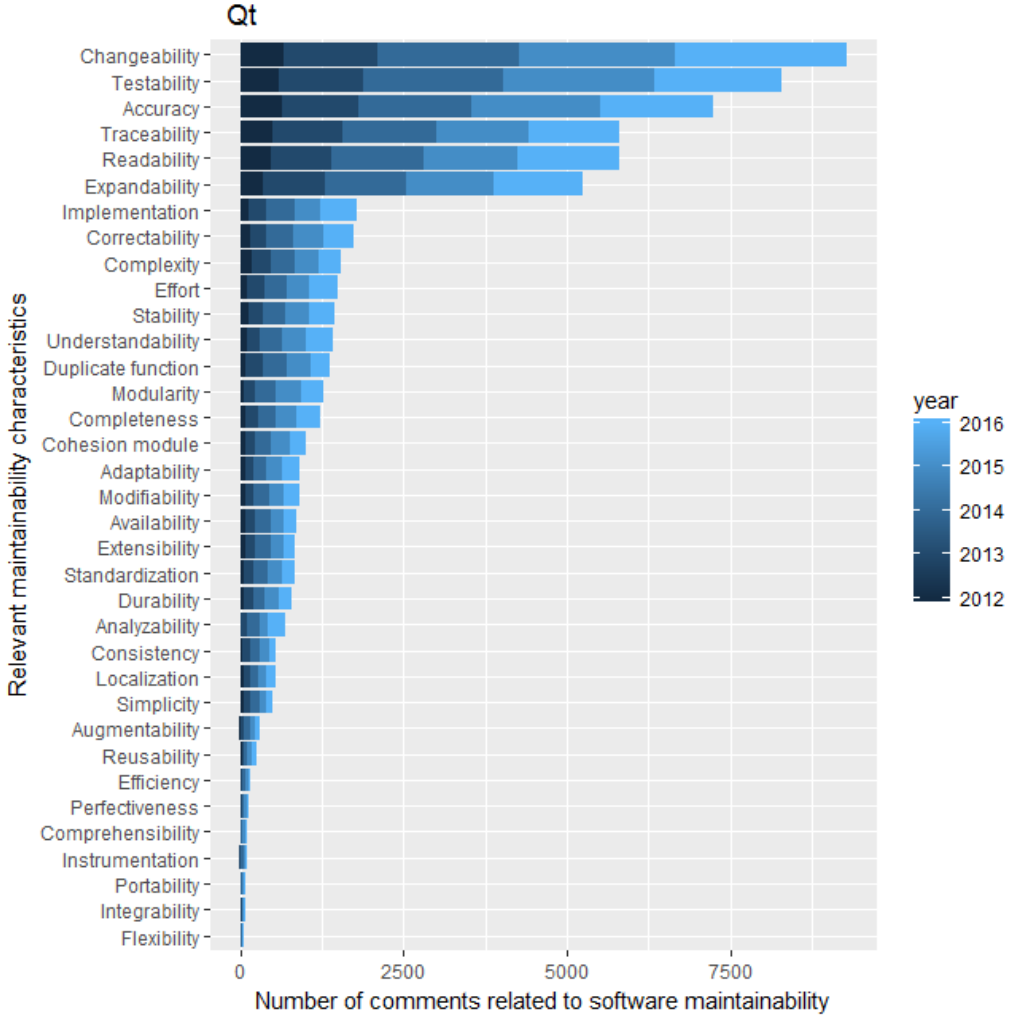
นอกจากนี้ผู้วิจัยได้ใช้การวิเคราะห์การถดถอยเชิงเส้นตรง เพื่อพยากรณ์แนวโน้มของจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาและแนวโน้มของจำนวนการแก้ไขโค้ดตามข้อเสนอแนะเกี่ยวกับการบำรุงรักษาทั้ง 35 คุณลักษณะ ที่จะเกิดขึ้นในปี ค.ศ. 2018 ซึ่งผลลัพธ์จากการใช้สถิติการวิเคราะห์การถดถอยของโครงการ Eclipse คือ ข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาและการแก้ไขโค้ดตามคำแนะนำทางด้านการบำรุงรักษาที่จะเกิดขึ้นในปี ค.ศ. 2018 มีจำนวน 24,788 และ 14,217 ข้อเสนอแนะ สำหรับผลลัพธ์ของโครงการ Qt คือ ข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาและการแก้ไขโค้ดตามคำแนะนำทางด้านการบำรุงรักษาที่จะเกิดขึ้นในปี ค.ศ. 2018 มีจำนวน 67,381 และ 40,084 ข้อเสนอแนะ โดยรูปที่ 4.14 แสดงให้เห็นว่าเมื่อแนวโน้มของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์เพิ่มสูงขึ้น แนวโน้มของการแก้ไขโค้ดตามข้อเสนอแนะเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ของโครงการ Eclipse และโครงการ Qt จะมีจำนวนเพิ่มมากขึ้นตามไปด้วย



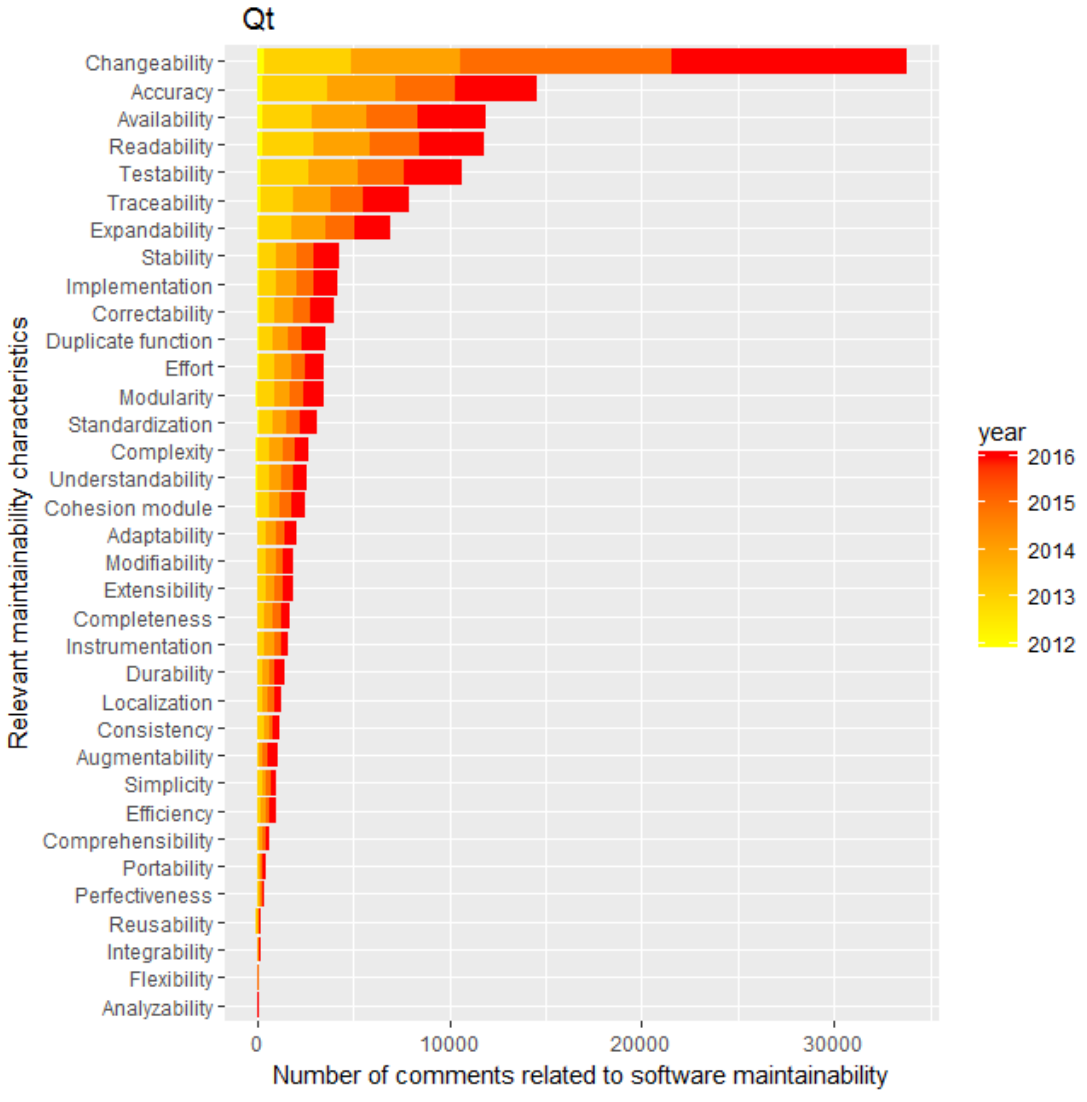
รูปที่ 4.14 กราฟแสดงแนวโน้มของข้อเสนอแนะเกี่ยวกับการบำรุงรักษาและแนวโน้มของการแก้ไขโค้ดที่ได้รับคำแนะนำทางด้านการบำรุงรักษาโครงการ Eclipse และโครงการ Qt

4.3 คุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ประเภทใดที่นักพัฒนาให้ความสนใจเป็นพิเศษ

ผู้วิจัยได้ทำการพิจารณาข้อเสนอแนะที่ได้รับคำแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ของโครงการ Eclipse และโครงการ Qt พบว่า ผู้ตรวจทานโค้ดของทั้ง 2 โครงการให้ความสำคัญกับการบำรุงรักษาซอฟต์แวร์บางคุณลักษณะมากเป็นพิเศษ โดยรูปที่ 4.15 เป็นการแสดงจำนวนของข้อเสนอแนะทางด้านการบำรุงรักษาที่ได้รับคำแนะนำจากผู้ตรวจทานโค้ดของโครงการ Eclipse และรูปที่ 4.16 เป็นการแสดงจำนวนของข้อเสนอแนะทางด้านการบำรุงรักษาที่ได้รับคำแนะนำจากผู้ตรวจทานโค้ดของโครงการ Qt



รูปที่ 4.15 จำนวนของข้อเสนอแนะทางด้านการบำรุงรักษาที่ได้รับคำแนะนำจากผู้ตรวจทานโค้ดของโครงการ Eclipse

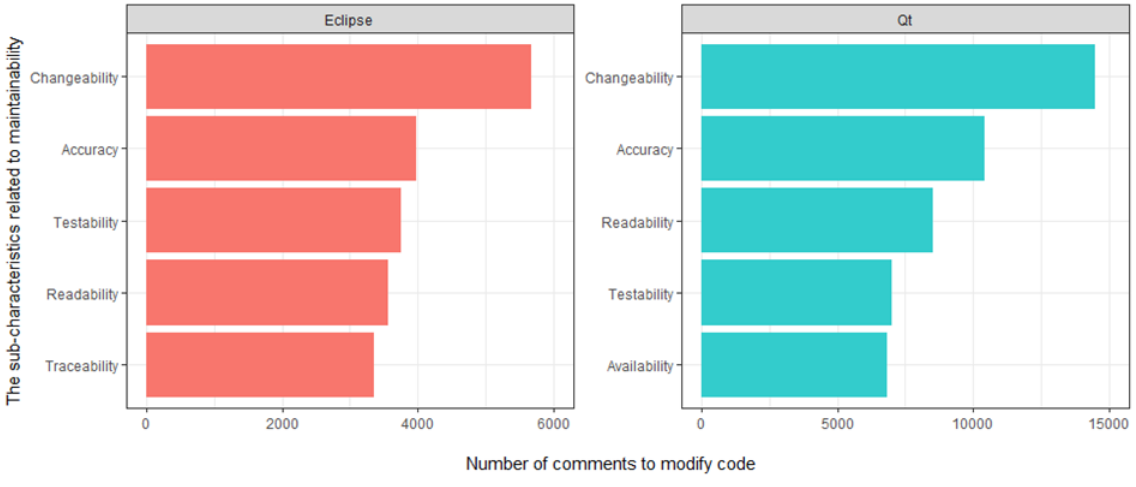


รูปที่ 4.16 จำนวนของข้อเสนอแนะทางด้านการบำรุงรักษาที่ได้รับคำแนะนำจากผู้ตรวจทานโค้ดของโครงการ Qt

เมื่อทำการเปรียบเทียบข้อมูลของข้อเสนอแนะเกี่ยวกับการบำรุงรักษาที่ได้รับจากผู้ตรวจทานโค้ดของโครงการ Eclipse และโครงการ Qt พบว่า คุณลักษณะที่ผู้ตรวจทานโค้ดของโครงการ Eclipse และโครงการ Qt ให้คำแนะนำมากที่สุด คือ ประเภท Changeability ดังแสดงในรูปที่ 4.15 และรูปที่ 4.16 ซึ่งข้อมูลดังกล่าวทำให้สามารถวิเคราะห์ได้ว่า ความสามารถในการเปลี่ยนแปลงเป็นคุณลักษณะที่ผู้ตรวจทานโค้ดให้ความสำคัญเป็นพิเศษ เพราะเป็นคุณลักษณะที่ทำการปรับปรุงแก้ไขข้อผิดพลาดและช่วยลดข้อบกพร่องของซอฟต์แวร์

สำหรับคุณลักษณะที่ได้รับความสนใจรองลงมาของโครงการ Eclipse คือ ประเภท Testability, Accuracy, Traceability และ Readability ตามลำดับ และคุณลักษณะที่ได้รับความสนใจรองลงมาของโครงการ Qt คือ ประเภท Accuracy, Availability, Readability และ Testability ตามลำดับ จากข้อมูลดังกล่าวสามารถสรุปได้ว่า Accuracy, Testability และ Readability เป็นคุณลักษณะที่ผู้ตรวจทานโค้ดทั้ง 2 โครงการ ให้ความสนใจเป็นพิเศษรองลงมาจากประเภท Changeability แต่ที่ต่างกันคือ โครงการ Eclipse ให้ความสนใจคุณลักษณะ Traceability และโครงการ Qt ให้ความสนใจคุณลักษณะ Availability

เมื่อได้คุณลักษณะที่ผู้ตรวจทานโค้ดส่วนใหญ่ให้คำแนะนำในการแก้ไขปรับปรุงโค้ดทางด้านการบำรุงรักษามากเป็นพิเศษแล้ว ผู้วิจัยจึงทำตรวจสอบว่า นักพัฒนาในโครงการโอเพนซอร์สทั้ง 2 โครงการ ได้ทำการแก้ไขปรับปรุงซอร์สโค้ดตามข้อเสนอแนะเกี่ยวกับการบำรุงรักษาทั้ง 5 ประเภทมากน้อยแค่ไหน เพื่อที่จะตรวจสอบว่านักพัฒนาในโครงการโอเพนซอร์ให้ความสำคัญกับคุณลักษณะเกี่ยวกับการบำรุงรักษาประเภทใดมากเป็นพิเศษ

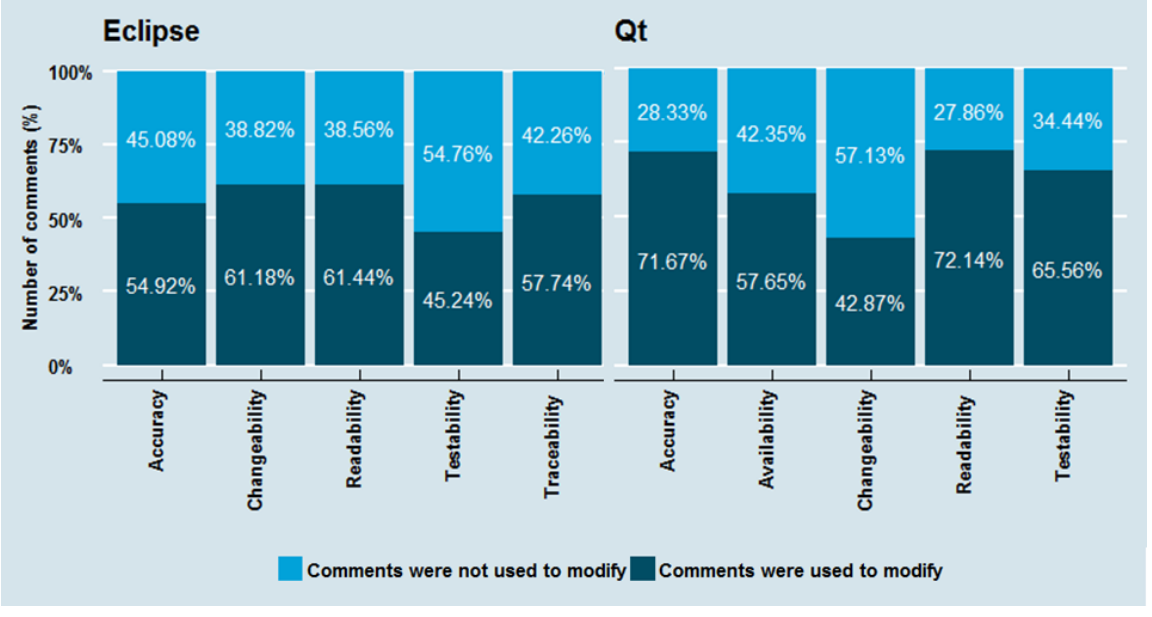


รูปที่ 4.17 จำนวนการแก้ไขปรับปรุงซอร์สโค้ดตามข้อเสนอแนะเกี่ยวกับการบำรุงรักษาทั้ง 5 ประเภทของโครงการ Eclipse และโครงการ Qt ที่ได้รับความนิยม

จากรูปที่ 4.17 เป็นการแสดงจำนวนการแก้ไขโค้ดตามข้อเสนอแนะเกี่ยวกับการบำรุงรักษา 5 ประเภทที่ได้รับความนิยมในการให้คำแนะนำของผู้ตรวจทานโค้ดของโครงการ Eclipse และโครงการ Qt ซึ่งข้อมูลจากรูปที่ 4.17 สามารถสรุปได้ว่า ชุมชนนักพัฒนาในโครงการโอเพนซอร์สให้ความสนใจคุณลักษณะเกี่ยวกับการบำรุงรักษามากที่สุด คือ ประเภท Changeability ส่วนประเภทที่มีการแก้ไขโค้ดรองลงมา คือ Accuracy สาเหตุที่นักพัฒนาให้ความสำคัญกับความเที่ยงตรงของซอฟต์แวร์

เพราะเป็นคุณลักษณะที่จะช่วยให้ผลิตภัณฑ์ซอฟต์แวร์มีผลลัพธ์ที่ถูกต้อง โดยสามารถยอมรับในระดับความแม่นยำ เช่น การดำเนินการของโปรแกรมสามารถคำนวณผลลัพธ์หรือแสดงค่าที่อ่านได้ใกล้เคียงค่าจริง

เมื่อผู้วิจัยได้ทำการพิจารณาจำนวนของข้อเสนอแนะกับจำนวนของการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาทั้ง 5 ประเภท พบว่าถึงแม้ประเภท Changeability และ Accuracy จะเป็นคุณลักษณะที่มีจำนวนการแก้ไขโค้ดเกี่ยวกับการบำรุงรักษาในโครงการ Eclipse และโครงการ Qt มาก แต่เมื่อทำการเปรียบเทียบอัตราส่วนในการแก้ไขโค้ดของทั้ง 5 คุณลักษณะแล้ว จึงสามารถสรุปได้ว่าประเภท Readability เป็นคุณลักษณะที่นักพัฒนาในโครงการโอเพนซอร์สทั้ง 2 โครงการ แก้ไขปรับปรุงมากที่สุด



รูปที่ 4.18 อัตราส่วนของการแก้ไขโค้ดทางด้านการบำรุงรักษาที่นักพัฒนาทำการแก้ไข

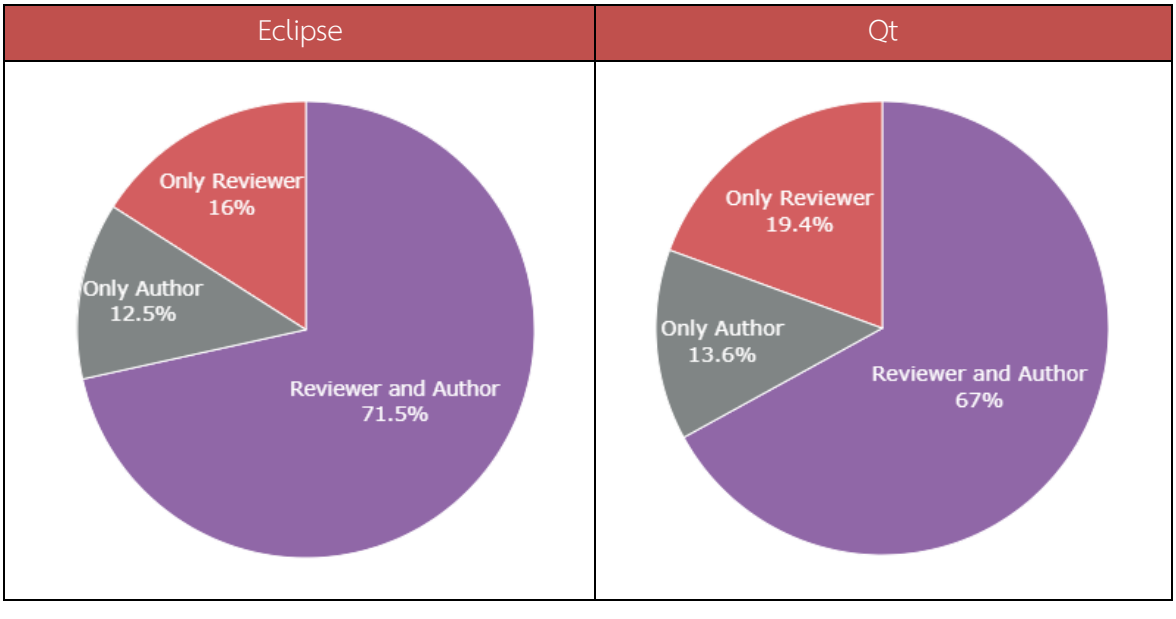
จากรูปที่ 4.18 สามารถนำมาเป็นหลักฐานเชิงประจักษ์ได้ว่า ชุมชนนักพัฒนาในโครงการโอเพนซอร์สได้ทำการแก้ไขโค้ดตามข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษามากที่สุด คือ ประเภท Readability ที่มีอัตราการแก้ไขโค้ดตามข้อเสนอแนะทางด้านการบำรุงรักษาในโครงการ Eclipse คิดเป็น 61.44% และโครงการ Qt คิดเป็น 72.14% เมื่อเทียบกับอัตราการแก้ไขโค้ดตามข้อเสนอแนะที่ได้รับการแก้ไขโค้ดทางด้านการบำรุงรักษาทั้ง 5 คุณลักษณะ

4.4 ขนาดกลุ่มของผู้ตรวจทานโค้ดส่งผลต่อจำนวนข้อเสนอแนะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์หรือไม่

สำหรับหัวข้อนี้ ผู้วิจัยได้ทำการศึกษาบทบาทของนักพัฒนาซอฟต์แวร์ที่มีอยู่ในโครงการโอเพนซอร์ส เพื่อทำการตรวจสอบว่า ในแต่ละเดือนมีจำนวนนักพัฒนาซอฟต์แวร์เท่าใดที่ทำงานเกี่ยวกับการตรวจทานโค้ด นอกจากนี้ผู้วิจัยยังใช้สถิติสหสัมพันธ์ (Correlation) ในการหาความสัมพันธ์ระหว่างขนาดกลุ่มของนักพัฒนาซอฟต์แวร์กับจำนวนของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ที่ได้รับในแต่ละเดือน เพื่อที่จะตอบคำถามว่า ขนาดกลุ่มหรือทีมของนักพัฒนาซอฟต์แวร์เป็นปัจจัยที่ส่งผลต่อจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์หรือไม่

4.4.1 บทบาทของนักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์ส

ในกระบวนการตรวจทานโค้ดของระบบเกอริต นักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์สมักจะมี 2 บทบาท นั่นคือ บทบาทในการเป็นผู้ตรวจทานโค้ดและบทบาทในการเป็นผู้แก้ไขโค้ดในโครงการโอเพนซอร์สนั่นเอง จากข้อมูลของข้อเสนอแนะทั้ง 2 โครงการ ผู้วิจัยได้ทำการสรุปบทบาทของนักพัฒนาซอฟต์แวร์ โดยจะแสดงอัตราส่วนร้อยละของแต่ละบทบาทที่เกิดขึ้นในโครงการ Eclipse และโครงการ Qt แสดงดังรูปที่ 4.19



รูปที่ 4.19 อัตราส่วนของบทบาทการเป็นผู้ตรวจทานโค้ดและนักพัฒนาที่ทำการแก้ไขโค้ด

รูปที่ 4.19 แสดงให้เห็นถึงบทบาทของนักพัฒนาในโครงการ Eclipse และโครงการ Qt แบ่งออกเป็น 3 บทบาท คือ

1) นักพัฒนาซอฟต์แวร์ที่ทำหน้าที่ในการตรวจทานโค้ดเพียงอย่างเดียว (Only Reviewer) เป็นบุคคลที่ทำการตรวจทานโค้ดและให้คำแนะนำในการปรับปรุงแก้ไขโค้ด โดยที่ไม่มีส่วนร่วมในการแก้ไขโค้ดตามคำแนะนำของผู้ตรวจทานโค้ดคนอื่น ๆ ในโครงการโอเพนซอร์ส

2) นักพัฒนาซอฟต์แวร์ที่ทำหน้าที่แก้ไขโค้ดตามคำแนะนำของผู้ตรวจทานโค้ดเพียงอย่างเดียว (Only Author) เป็นบุคคลที่ทำการอ่านข้อเสนอแนะที่ได้รับจากผู้ตรวจทานโค้ดและทำการแก้ไขปรับปรุงโค้ดตามข้อเสนอแนะดังกล่าว โดยที่ไม่มีส่วนร่วมในการตรวจทานโค้ดของผู้อื่น

3) นักพัฒนาซอฟต์แวร์ที่ทำหน้าที่ทั้ง 2 บทบาท (Reviewer and Author) เป็นบุคคลที่เป็นทั้งผู้ตรวจทานโค้ดของผู้อื่น และเป็นนักพัฒนาที่ทำการแก้ไขปรับปรุงโค้ดตามข้อเสนอแนะที่ได้รับจากผู้ตรวจทานโค้ดคนอื่น ๆ

ผลลัพธ์จากการศึกษาในรูปที่ 4.19 แสดงให้เห็นว่านักพัฒนาซอฟต์แวร์ส่วนใหญ่ของโครงการ Eclipse และโครงการ Qt เป็นบุคคลที่เป็นทั้งผู้ตรวจทานโค้ดและนักพัฒนาที่ทำการแก้ไขโค้ด โดยโครงการ Eclipse มีนักพัฒนาที่เป็นทั้ง 2 บทบาท จำนวน 71.5% และสำหรับโครงการ Qt มีนักพัฒนาที่เป็นทั้ง 2 บทบาท จำนวน 67% จากผลลัพธ์ดังกล่าวทำให้สามารถสรุปได้ว่า ชุมชนนักพัฒนาซอฟต์แวร์ในโครงการ Eclipse และโครงการ Qt ส่วนใหญ่จะเป็นทั้งผู้ที่ทำการแก้ไขโค้ดตามข้อเสนอแนะของผู้ตรวจทานโค้ดคนอื่น ๆ ไปพร้อม ๆ กับเป็นผู้ตรวจทานโค้ดที่ทำการตรวจทานโค้ดและคอยให้คำแนะนำในการปรับปรุงโค้ดแก่นักพัฒนาคนอื่น ๆ

นอกจากนี้โครงการ Eclipse มีนักพัฒนาที่มีบทบาทในการตรวจทานโค้ดเพียงอย่างเดียว 16% และนักพัฒนาที่มีบทบาทในการแก้ไขโค้ดเพียงอย่างเดียว 12.5% ส่วนโครงการ Qt มีนักพัฒนาที่มีบทบาทในการตรวจทานโค้ดเพียงอย่างเดียว 19.4% และนักพัฒนาที่มีบทบาทในการแก้ไขโค้ดเพียงอย่างเดียว 13.6% ซึ่งข้อมูลนี้ทำให้สามารถวิเคราะห์ได้ว่า ทั้ง 2 โครงการมีนักพัฒนาที่ทำหน้าที่ตรวจทานโค้ดหรือนักพัฒนาที่ทำหน้าที่ในการแก้ไขซอร์สโค้ดเพียงหน้าที่ใดหน้าที่หนึ่ง ถือว่ามีจำนวนน้อยและมีอัตราส่วนที่ต่างกันไม่มากนัก แต่ผลลัพธ์ดังกล่าวนี้เป็นที่น่าแปลกใจที่นักพัฒนาซอฟต์แวร์ทำการตรวจทานโค้ดเพียงอย่างเดียวโดยที่ไม่ทำการแก้ไขโค้ด เพราะโดยปกติทั่วไปแล้วนักพัฒนาในโครงการโอเพนซอร์สมักจะให้คำแนะนำเกี่ยวกับการปรับปรุงโค้ดและทำการแก้ไขโค้ดไปพร้อม ๆ กัน จึงอาจเป็นไปได้ว่า นักพัฒนาบางคนที่ทำการตรวจทานโค้ดมักจะให้คำแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์แต่อาจทำการแก้ไขโค้ดคุณลักษณะอื่น ๆ หรือด้านอื่น ๆ นอกเหนือจากคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ เช่น ด้านความมั่นคงของซอฟต์แวร์ (Security)

4.4.2 ขนาดกลุ่มของผู้ตรวจทานโค้ดที่ให้คำแนะนำเกี่ยวกับการบำรุงรักษา

เมื่อผู้วิจัยได้ทำการศึกษาเกี่ยวกับบทบาทของนักพัฒนาในโครงการโอเพนซอร์สทั้ง 2 โครงการ พบว่านักพัฒนาซอฟต์แวร์ส่วนใหญ่ในโครงการโอเพนซอร์สมักจะทำหน้าที่ให้คำแนะนำในการปรับปรุงแก้ไขโค้ดและทำหน้าที่ในการแก้ไขโค้ดตามคำแนะนำจากผู้ตรวจทานโค้ดคนอื่นด้วยเช่นกัน ด้วยเหตุนี้ผู้วิจัยจึงได้วิเคราะห์ว่า ช่วงระยะเวลา 5 ปีที่ผ่านมา มีนักพัฒนาซอฟต์แวร์ที่ให้คำแนะนำเกี่ยวกับการบำรุงรักษามากน้อยเท่าใด เมื่อเทียบกับจำนวนของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ที่ได้รับในแต่ละเดือน นอกจากนี้ผู้วิจัยยังทำการตรวจสอบขนาดกลุ่มหรือทีมของนักพัฒนาเพื่อแสดงให้เห็นถึงแนวโน้มของจำนวนนักพัฒนาซอฟต์แวร์ที่ทำหน้าที่ในการตรวจทานโค้ดช่วงปี ค.ศ. 2012 – 2016 โดยตารางที่ 4.20 และตารางที่ 4.21 จะแสดงจำนวนผู้ตรวจทานโค้ดที่ให้คำแนะนำเกี่ยวกับการปรับปรุงโค้ดทางด้านการบำรุงรักษาซอฟต์แวร์และจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ที่ได้รับจากผู้ตรวจทานโค้ดในแต่ละเดือนของโครงการ Eclipse และโครงการ Qt

ตารางที่ 4.20 จำนวนผู้ตรวจทานโค้ดที่ให้คำแนะนำในการปรับปรุงโค้ดและจำนวนข้อเสนอแนะเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ในแต่ละเดือนของโครงการ Eclipse

ปี 2012		
เดือนที่	จำนวนผู้ตรวจทานโค้ด	จำนวนข้อเสนอแนะ
1	11	172
2	14	234
3	27	239
4	29	300
5	37	421
6	30	208
7	43	578
8	45	524
9	31	374
10	39	485
11	44	618
12	36	753

ปี 2013		
เดือนที่	จำนวนผู้ตรวจทานโค้ด	จำนวนข้อเสนอแนะ
1	49	567
2	52	646
3	59	866
4	67	886
5	65	787
6	53	506
7	61	866
8	61	982
9	70	976
10	70	987
11	80	1,436
12	77	1,000

ตารางที่ 4.20 จำนวนผู้ตรวจทานโค้ดที่ให้คำแนะนำในการปรับปรุงโค้ดและจำนวนข้อเสนอแนะเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ในแต่ละเดือนของโครงการ Eclipse (ต่อ)

ปี 2014		
เดือนที่	จำนวนผู้ตรวจทานโค้ด	จำนวนข้อเสนอแนะ
1	81	1,154
2	83	1,118
3	91	1,443
4	103	1,817
5	102	1,166
6	92	1,185
7	96	1,354
8	101	1,835
9	80	923
10	83	1,261
11	96	1,521
12	90	1,077

ปี 2015		
เดือนที่	จำนวนผู้ตรวจทานโค้ด	จำนวนข้อเสนอแนะ
1	103	1,318
2	105	1,746
3	117	1,505
4	101	1,231
5	120	1,346
6	94	1,292
7	116	1,767
8	105	1,632
9	110	1,186
10	112	1,528
11	109	1,333
12	93	839

ปี 2016		
เดือนที่	จำนวนผู้ตรวจทานโค้ด	จำนวนข้อเสนอแนะ
1	107	1,619
2	99	1,098
3	113	1,590
4	121	1,772
5	111	1,597
6	111	1,511

ปี 2016		
เดือนที่	จำนวนผู้ตรวจทานโค้ด	จำนวนข้อเสนอแนะ
7	120	1,513
8	108	1,315
9	105	989
10	110	1,613
11	120	1,253
12	84	758

ตารางที่ 4.21 จำนวนนักพัฒนาซอฟต์แวร์ที่ให้คำแนะนำในการปรับปรุงโค้ดและจำนวนข้อเสนอแนะเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ในแต่ละเดือนของโครงการ Qt

ปี 2012		
เดือนที่	จำนวนผู้ตรวจทานโค้ด	จำนวนข้อเสนอแนะ
1	16	105
2	34	125
3	36	181
4	49	346
5	47	319
6	43	252
7	44	414
8	32	151
9	31	172
10	35	164
11	38	207
12	52	360

ปี 2013		
เดือนที่	จำนวนผู้ตรวจทานโค้ด	จำนวนข้อเสนอแนะ
1	74	940
2	115	1,723
3	132	2,882
4	143	2,465
5	121	1,782
6	130	2,060
7	121	2,211
8	165	2,747
9	150	3,949
10	160	3,287
11	163	3,612
12	146	2,757

ปี 2014		
เดือนที่	จำนวนผู้ตรวจทานโค้ด	จำนวนข้อเสนอแนะ
1	154	3,378
2	138	3,347
3	156	2,882
4	160	3,153
5	145	2,307
6	135	2,087

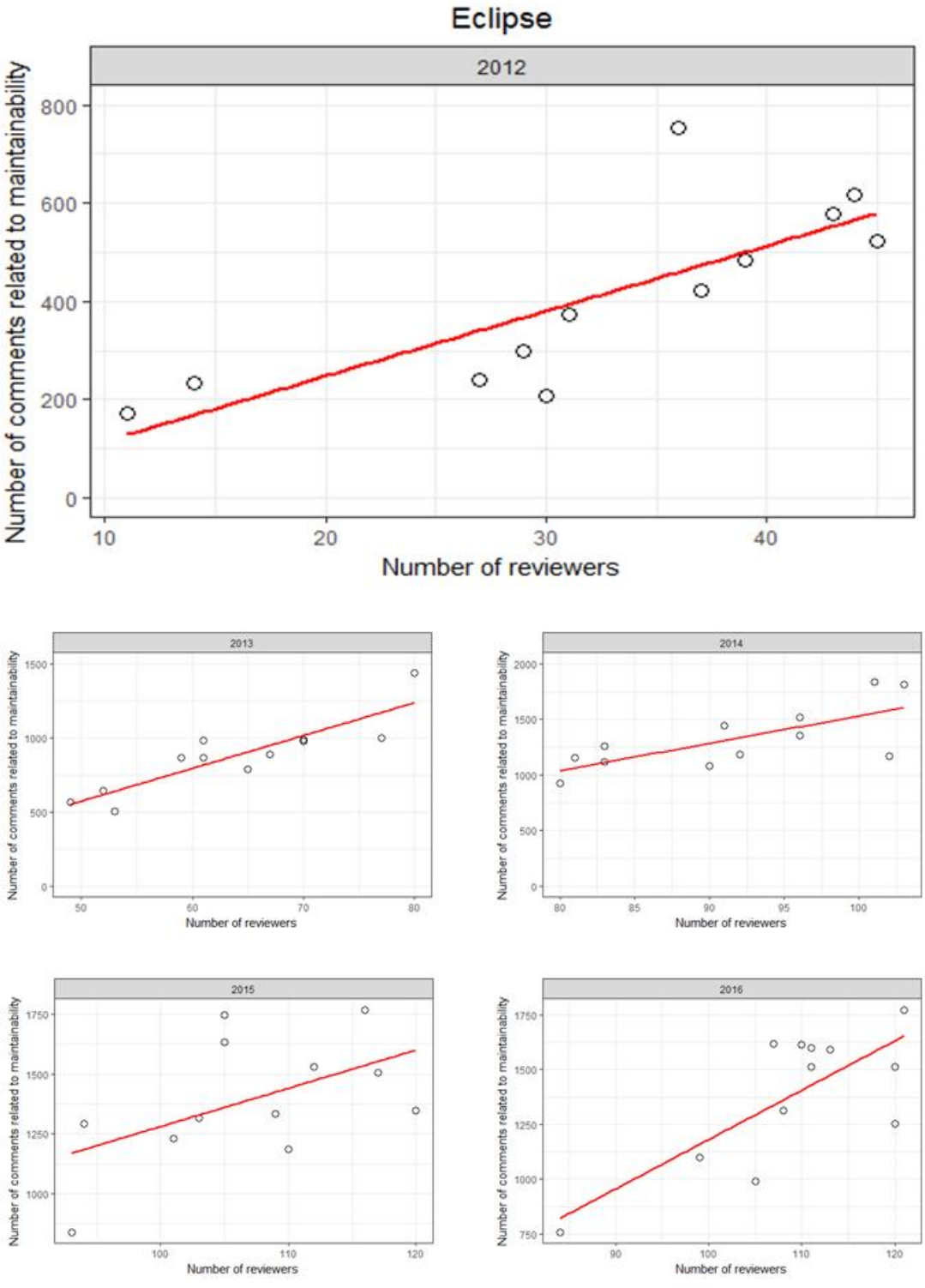
ปี 2014		
เดือนที่	จำนวนผู้ตรวจทานโค้ด	จำนวนข้อเสนอแนะ
7	141	3,258
8	151	3,207
9	142	2,738
10	156	3,055
11	139	2,348
12	134	2,233

ตารางที่ 4.21 จำนวนนักพัฒนาซอฟต์แวร์ที่ให้คำแนะนำในการปรับปรุงโค้ดและจำนวนข้อเสนอแนะเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ในแต่ละเดือนของโครงการ Qt (ต่อ)

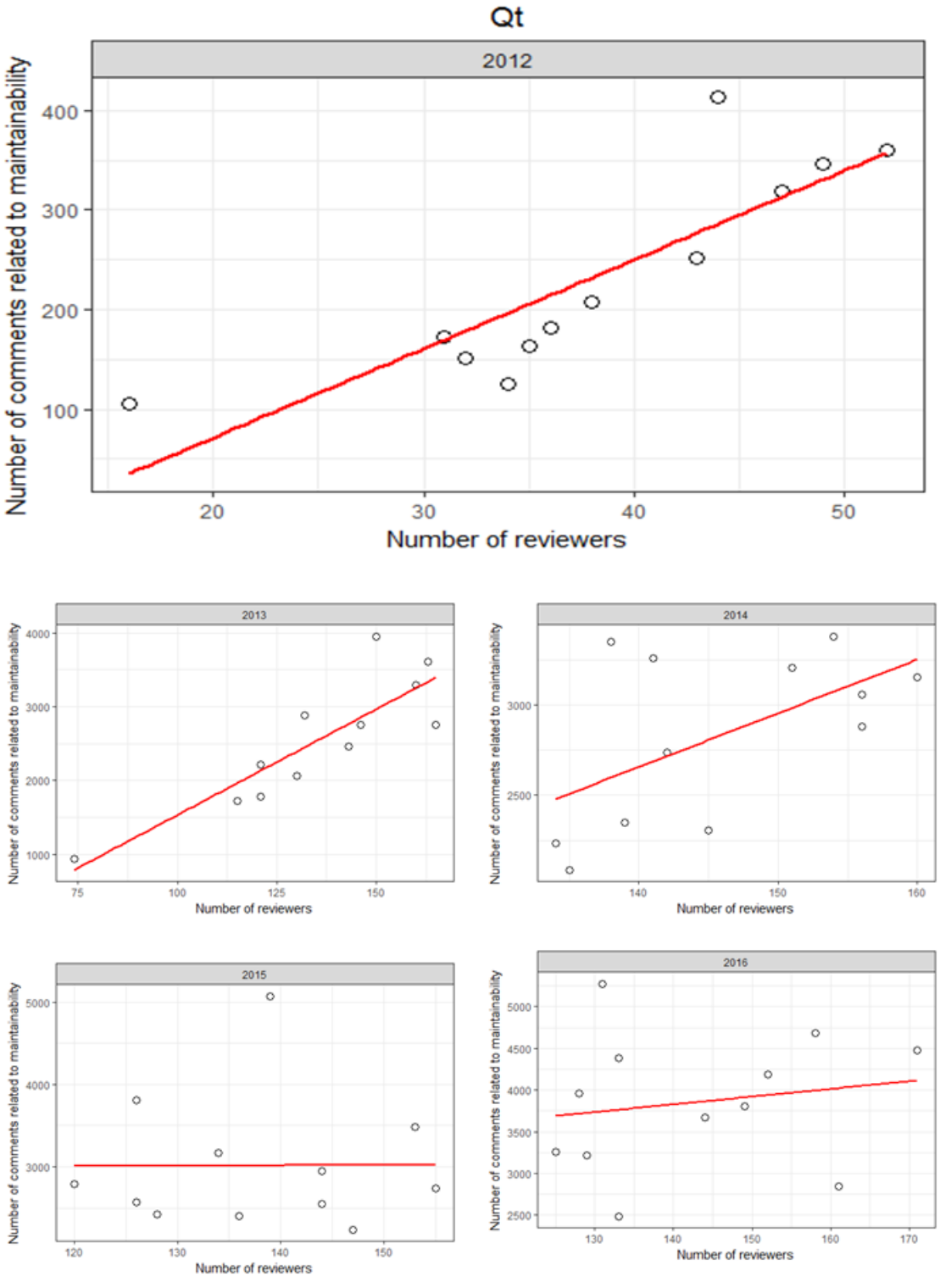
ปี 2015		
เดือนที่	จำนวนผู้ตรวจทานโค้ด	จำนวนข้อเสนอแนะ
1	144	2,545
2	153	3,480
3	155	2,735
4	128	2,424
5	126	2,566
6	139	5,070
7	134	3,162
8	136	2,404
9	126	3,808
10	120	2,789
11	144	2,950
12	147	2,234

ปี 2016		
เดือนที่	จำนวนผู้ตรวจทานโค้ด	จำนวนข้อเสนอแนะ
1	161	2,846
2	144	3,676
3	152	4,183
4	171	4,474
5	158	4,681
6	133	4,384
7	128	3,961
8	131	5,273
9	125	3,256
10	129	3,218
11	149	3,802
12	133	2,485

ผู้วิจัยได้ทำการวิเคราะห์ข้อมูลข้างต้นโดยใช้โปรแกรม R ในการคำนวณค่าสถิติสหสัมพันธ์ (Correlation) เพื่อหาความสัมพันธ์ระหว่างจำนวนของนักพัฒนาซอฟต์แวร์ที่ให้คำแนะนำเกี่ยวกับการปรับปรุงโค้ดทางด้านการบำรุงรักษาซอฟต์แวร์และจำนวนของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ในช่วงปี ค.ศ. 2012 - 2016 ซึ่งผลลัพธ์ที่ได้จากการวิเคราะห์ค่าสัมประสิทธิ์สหสัมพันธ์เพียร์สันของโครงการ Eclipse เท่ากับ 0.78 0.88 0.72 0.53 และ 0.75 ตามลำดับ สำหรับค่าสัมประสิทธิ์สหสัมพันธ์เพียร์สันของโครงการ Qt เท่ากับ 0.85 0.86 0.57 0.005 และ 0.17 ตามลำดับ โดยรูปที่ 4.20 จะแสดงให้เห็นถึงความสัมพันธ์ระหว่างขนาดกลุ่มของผู้ตรวจทานโค้ดกับจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาของโครงการ Eclipse ในแต่ละปี และรูปที่ 4.21 แสดงความสัมพันธ์ระหว่างขนาดกลุ่มของผู้ตรวจทานโค้ดกับจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาของโครงการ Qt ในแต่ละปี

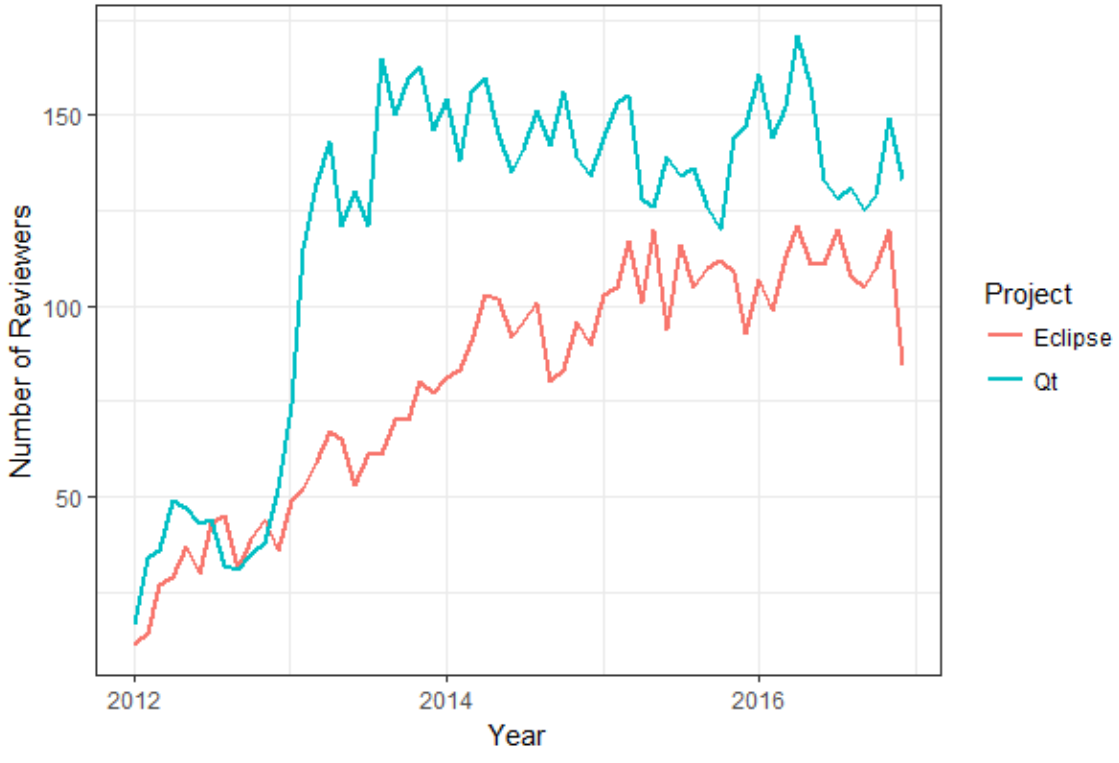


รูปที่ 4.20 ความสัมพันธ์ระหว่างจำนวนของผู้ตรวจทานโค้ดกับจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ทั้ง 12 เดือนของโครงการ Eclipse ในช่วงปี ค.ศ. 2012 – 2016



รูปที่ 4.21 ความสัมพันธ์ระหว่างจำนวนของผู้ตรวจทานโค้ดกับจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ทั้ง 12 เดือนของโครงการ Qt ในช่วงปี ค.ศ. 2012 – 2016

จากรูปที่ 4.20 แสดงให้เห็นว่าช่วงระยะเวลา 5 ปี (ค.ศ. 2012 - 2016) ขนาดกลุ่มของผู้ตรวจทานโค้ดกับจำนวนของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาของโครงการ Eclipse มีความสัมพันธ์กันในระดับสูง และเป็นความสัมพันธ์ที่เป็นไปในทิศทางเดียวกัน ยกเว้นความสัมพันธ์ของผู้ตรวจทานโค้ดและข้อเสนอแนะเกี่ยวกับการบำรุงรักษาในช่วงปี ค.ศ. 2015 ที่มีความสัมพันธ์กันในระดับปานกลาง และรูปที่ 4.21 สามารถสรุปได้ว่าช่วงระยะเวลา 5 ปีที่ผ่านมา มีเพียงปี ค.ศ. 2012 - 2013 ที่ขนาดกลุ่มของผู้ตรวจทานโค้ดกับจำนวนข้อเสนอแนะเกี่ยวกับการบำรุงรักษาของโครงการ Qt มีความสัมพันธ์กันในระดับสูง และเป็นความสัมพันธ์ที่เป็นไปในทิศทางเดียวกัน แต่ในปี ค.ศ. 2014 แสดงให้เห็นว่า ทั้ง 2 ข้อมูลมีความสัมพันธ์กันในระดับปานกลาง และสำหรับปี ค.ศ. 2015 และ 2016 ขนาดกลุ่มของผู้ตรวจทานโค้ดกับจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์มีความสัมพันธ์กันในระดับต่ำมาก



รูปที่ 4.22 แนวโน้มของจำนวนผู้ตรวจทานโค้ดในโครงการ Eclipse และโครงการ Qt ที่ให้คำแนะนำเกี่ยวกับการบำรุงรักษาในช่วงระยะเวลา 5 ปี

เมื่อผู้วิจัยได้นำข้อมูลข้างต้นมาทำการวิเคราะห์เพื่อดูแนวโน้มของจำนวนผู้ตรวจทานโค้ดในช่วงระยะเวลา 5 ปีที่ผ่านมา จากรูปที่ 4.22 แสดงให้เห็นถึงแนวโน้มของขนาดกลุ่มผู้ตรวจทานโค้ด

ที่ให้คำแนะนำเกี่ยวกับการบำรุงรักษามีจำนวนเพิ่มขึ้นตั้งแต่ปี ค.ศ. 2012 และมีแนวโน้มที่เพิ่มสูงขึ้นเรื่อย ๆ ในแต่ละปี ซึ่งผลลัพธ์ดังกล่าวแสดงให้เห็นว่า โครงการ Eclipse และโครงการ Qt มีนักพัฒนาซอฟต์แวร์รุ่นใหม่ที่ให้ความสนใจในการเข้าร่วมพัฒนาซอฟต์แวร์และให้คำแนะนำที่จะช่วยเพิ่มคุณภาพของซอฟต์แวร์ในโครงการโอเพนซอร์สมากยิ่งขึ้นในแต่ละปี

เพื่อตอบคำถามวิจัยว่า *"ขนาดกลุ่มหรือทีมของนักพัฒนาซอฟต์แวร์เป็นปัจจัยที่ส่งผลต่อจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์หรือไม่"* ผู้วิจัยจึงใช้โปรแกรม R ในการวิเคราะห์ค่าสหสัมพันธ์ โดยผลลัพธ์ที่ได้จากข้อมูลดังกล่าวข้างต้นทำให้สามารถวิเคราะห์ได้ว่า เมื่อจำนวนของผู้ตรวจทานโค้ดมีจำนวนเพิ่มมากขึ้น จำนวนของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาก็จะเพิ่มมากขึ้นด้วย แต่ในทางกลับกันก็อาจเป็นสิ่งที่บ่งบอกได้ว่า โครงการโอเพนซอร์สมีความซับซ้อนและมีข้อบกพร่องที่จะต้องทำการแก้ไขปรับปรุงมากขึ้นเช่นกัน อย่างไรก็ตามผลลัพธ์นี้สามารถสรุปได้ว่า ขนาดกลุ่มของนักพัฒนาซอฟต์แวร์ที่ให้คำแนะนำเกี่ยวกับการบำรุงรักษาเป็นปัจจัยที่ส่งผลต่อจำนวนของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์

4.5 ผู้ตรวจทานโค้ดทำงานเกี่ยวกับการตรวจทานโค้ดในโครงการโอเพนซอร์สมากน้อยเพียงใดต่อสัปดาห์

หลังจากที่ผู้วิจัยได้ทำการตรวจสอบปัจจัยเกี่ยวกับขนาดกลุ่มที่ส่งผลต่อจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์แล้ว ผู้วิจัยจึงได้ทำการตรวจสอบปัจจัยที่เกี่ยวข้องกับช่วงวันทำงานของนักพัฒนาซอฟต์แวร์ที่ให้คำแนะนำเกี่ยวกับการบำรุงรักษา โดยตรวจสอบเวลาที่ผู้ตรวจทานได้ให้คำแนะนำเกี่ยวกับการบำรุงรักษาแต่ละวันต่อสัปดาห์ในช่วงระยะเวลา 5 ปี นั่นคือ ปี ค.ศ. 2012 – 2016 และสำหรับข้อมูลเกี่ยวกับข้อเสนอแนะทางด้านการบำรุงรักษาที่ผู้วิจัยได้ทำการรวบรวมแสดงดังตารางที่ 4.22 เป็นการแสดงจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาของโครงการ Eclipse ในแต่ละวันของช่วงสัปดาห์ และตารางที่ 4.23 แสดงจำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาของโครงการ Qt ในแต่ละวันของช่วงสัปดาห์

ตารางที่ 4.22 จำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาของโครงการ Eclipse ในแต่ละวันของช่วงสัปดาห์

ลำดับ	เดือน	ปี				
		2012	2013	2014	2015	2016
1	Sunday	293	391	341	440	402
2	Monday	780	960	2,006	2,079	2,000
3	Tuesday	730	1,723	3,070	2,987	3,022
4	Wednesday	982	2,127	3,199	3,275	3,621
5	Thursday	847	2,389	3,147	3,355	3,213
6	Friday	921	1,941	2,884	3,264	3,230
7	Saturday	353	974	1,207	1,323	1,140

ตารางที่ 4.23 จำนวนข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาของโครงการ Qt ในแต่ละวันของช่วงสัปดาห์

ลำดับ	เดือน	ปี				
		2012	2013	2014	2015	2016
1	Sunday	85	1,501	1,129	1,259	1,090
2	Monday	520	4,698	5,377	6,481	7,247
3	Tuesday	496	5,735	6,505	7,500	9,595
4	Wednesday	609	5,497	7,038	6,898	9,818
5	Thursday	415	5,862	6,653	6,947	8,233
6	Friday	494	5,544	5,993	5,449	8,080
7	Saturday	177	1,578	1,298	1,633	2,176

สำหรับการวิเคราะห์เพื่อตรวจสอบเวลาในการทำงานของผู้ตรวจทานโค้ดในแต่ละโครงการ ผู้วิจัยได้ใช้สถิติการวิเคราะห์ความแปรปรวน (ANOVA หรือ Analysis of Variance) และ Post-hoc Comparisons แบบ Tukey's HSD (Honestly Significant Difference) ซึ่งเป็นการทดสอบเพื่อเปรียบเทียบค่าเฉลี่ยของข้อมูลได้ครั้งละหลายคู่ สาเหตุที่ผู้วิจัยใช้ Tukey's HSD เนื่องจากการทดสอบสมมติฐานของ ANOVA เป็นการทดสอบภาพรวมว่าค่าเฉลี่ยหลายค่านี้เป็นค่าเดียวกันหรือไม่ ดังนั้นผลลัพธ์ที่ได้จึงยังไม่สามารถตอบได้ว่า ถ้าแตกต่างกันแล้วค่าเฉลี่ยของข้อมูลคู่ใดบ้างที่ต่างกัน นอกจากสาเหตุดังกล่าวผู้วิจัยจึงเลือกใช้การวิเคราะห์ความแปรปรวนแบบ Tukey's HSD เพราะเป็นสถิติที่มักจะได้รับความนิยมมากในงานวิจัยทางด้านวิศวกรรมซอฟต์แวร์

การทดสอบ Post Hoc เป็นการควบคุมค่าความคลาดเคลื่อนที่อาจเพิ่มขึ้น ซึ่ง Post Hoc Test แบบ Tukey's HSD (Honestly Significant Difference) เป็นการทดสอบค่าเฉลี่ยหลายคู่ (Family-Wise Error. Rate: FWE) ให้ความคลาดเคลื่อนที่เกิดขึ้นจริงเท่ากันกับความคลาดเคลื่อนที่ตั้งไว้ แม้ว่าจะจับคู่ทดสอบค่าเฉลี่ยทุกความเป็นไปได้ก็ตาม (All Possible Pairwise) สำหรับงานวิจัยนี้ ผลลัพธ์ที่ได้จากการใช้ Post Hoc Test แบบ Tukey's HSD จะแสดงให้เห็นถึงค่าเฉลี่ยหลายคู่ของจำนวนข้อเสนอแนะที่ได้รับจากผู้ตรวจทานโค้ดในแต่ละวันของสัปดาห์

ผู้วิจัยได้ใช้โปรแกรม R ในการวิเคราะห์ความแปรปรวน โดยกำหนดให้ระดับความเชื่อมั่นอยู่ที่ 95% และมีสมมติฐานของการทดสอบดังนี้

H0: ค่าเฉลี่ยของจำนวนคำแนะนำเกี่ยวกับการบำรุงรักษาแต่ละวันในช่วงสัปดาห์ไม่แตกต่างกัน

H1: ค่าเฉลี่ยของจำนวนคำแนะนำเกี่ยวกับการบำรุงรักษาแต่ละวันในช่วงสัปดาห์แตกต่างกัน

ผลลัพธ์ที่ได้จากการวิเคราะห์โดยใช้โปรแกรม R โครงการ Eclips จะได้ค่า p-value = 0.000649 และโครงการ Qt จะได้ค่า p-value = 0.0105 ซึ่งค่า p-value ของทั้ง 2 โครงการ มีค่าน้อยกว่า 0.05 จึงทำการปฏิเสธ H0 ดังนั้นสามารถสรุปได้ว่า ค่าเฉลี่ยของแต่ละวันในช่วงสัปดาห์ที่ผู้ตรวจทานโค้ดให้คำแนะนำเกี่ยวกับการบำรุงรักษามีความแตกต่างกันในกลุ่ม เพื่อตรวจสอบความแตกต่างระหว่างกลุ่มของข้อมูล ผู้วิจัยจึงได้นำข้อมูลมาทำการวิเคราะห์ต่อ โดยหาค่าความแปรปรวนแบบ Tukey's HSD

ผู้วิจัยได้ใช้โปรแกรม R ในการวิเคราะห์ความแปรปรวนแบบ Tukey's HSD โดยกำหนดให้ระดับความเชื่อมั่นอยู่ที่ 95% และมีสมมติฐานของการทดสอบเพื่อเปรียบเทียบคาเฉลี่ยในแต่ละคู่ดังนี้

H0 : ค่าเฉลี่ยของจำนวนคำแนะนำเกี่ยวกับการบำรุงรักษาในแต่ละวันไม่แตกต่างกัน

H1 : ค่าเฉลี่ยของจำนวนคำแนะนำเกี่ยวกับการบำรุงรักษาในแต่ละวันแตกต่างกัน

```
Tukey multiple comparisons of means
    95% family-wise confidence level

                      diff          lwr       upr     p adj
Monday-Friday       -883.0 -2566.73106  800.7311 0.6443123
Saturday-Friday    -1448.6 -3132.33106  235.1311 0.1278870
Sunday-Friday      -2074.6 -3758.33106 -390.8689 0.0085778
Thursday-Friday      142.2 -1541.53106 1825.9311 0.9999635
Tuesday-Friday      -141.6 -1825.33106 1542.1311 0.9999644
Wednesday-Friday     192.8 -1490.93106 1876.5311 0.9997833
Saturday-Monday     -565.6 -2249.33106 1118.1311 0.9329528
Sunday-Monday      -1191.6 -2875.33106  492.1311 0.3049134
Thursday-Monday     1025.2  -658.53106 2708.9311 0.4776216
Tuesday-Monday       741.4  -942.33106 2425.1311 0.7988824
Wednesday-Monday    1075.8  -607.93106 2759.5311 0.4211734
Sunday-Saturday     -626.0 -2309.73106 1057.7311 0.8960318
Thursday-Saturday   1590.8   -92.93106 3274.5311 0.0734706
Tuesday-Saturday    1307.0  -376.73106 2990.7311 0.2113721
Wednesday-Saturday  1641.4   -42.33106 3325.1311 0.0597012
Thursday-Sunday     2216.8   533.06894 3900.5311 0.0043373
Tuesday-Sunday      1933.0   249.26894 3616.7311 0.0166102
Wednesday-Sunday    2267.4   583.66894 3951.1311 0.0033908
Tuesday-Thursday    -283.8 -1967.53106 1399.9311 0.9980390
Wednesday-Thursday    50.6 -1633.13106 1734.3311 0.9999999
Wednesday-Tuesday    334.4 -1349.33106 2018.1311 0.9951694
```

รูปที่ 4.23 ผลลัพธ์จากการวิเคราะห์ค่าความแปรปรวนแบบ Tukey's HSD ของโครงการ Eclipse

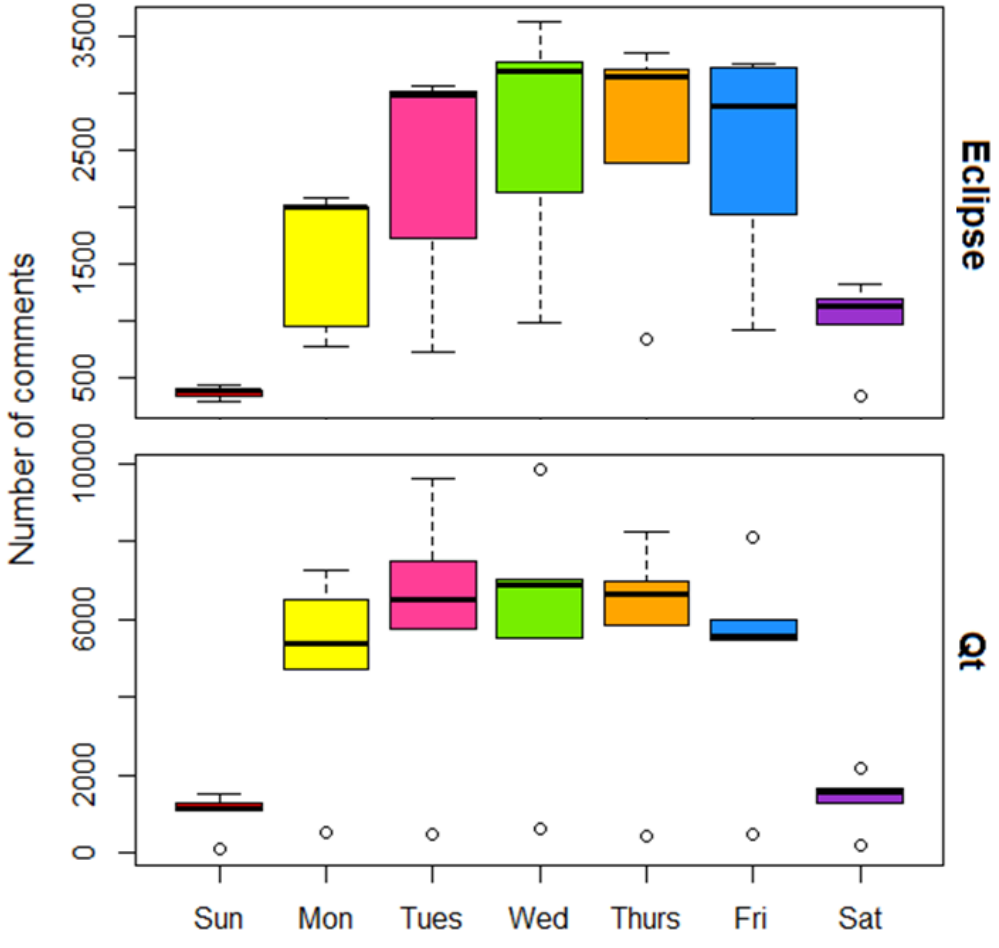
```
Tukey multiple comparisons of means
    95% family-wise confidence level

                     diff        lwr       upr     p adj
Monday-Friday       -107.4 -5355.5895  5140.79 1.0000000
Saturday-Friday    -3879.6 -9127.7895  1368.59 0.2589816
Sunday-Friday      -4099.2 -9347.3895  1148.99 0.2056828
Thursday-Friday      510.0 -4738.1895  5758.19 0.9999167
Tuesday-Friday       854.2 -4393.9895  6102.39 0.9983869
Wednesday-Friday     860.0 -4388.1895  6108.19 0.9983247
Saturday-Monday    -3772.2 -9020.3895  1475.99 0.2882500
Sunday-Monday      -3991.8 -9239.9895  1256.39 0.2306510
Thursday-Monday      617.4 -4630.7895  5865.59 0.9997466
Tuesday-Monday       961.6 -4286.5895  6209.79 0.9968880
Wednesday-Monday     967.4 -4280.7895  6215.59 0.9967835
Sunday-Saturday     -219.6 -5467.7895  5028.59 0.9999994
Thursday-Saturday   4389.6  -858.5895  9637.79 0.1483632
Tuesday-Saturday    4733.8  -514.3895  9981.99 0.0978611
Wednesday-Saturday  4739.6  -508.5895  9987.79 0.0971534
Thursday-Sunday     4609.2  -638.9895  9857.39 0.1141546
Tuesday-Sunday      4953.4  -294.7895 10201.59 0.0739613
Wednesday-Sunday    4959.2  -288.9895 10207.39 0.0734060
Tuesday-Thursday     344.2 -4903.9895  5592.39 0.9999918
Wednesday-Thursday   350.0 -4898.1895  5598.19 0.9999910
Wednesday-Tuesday      5.8 -5242.3895  5253.99 1.0000000
```

รูปที่ 4.24 ผลลัพธ์จากการวิเคราะห์ค่าความแปรปรวนแบบ Tukey's HSD ของโครงการ Qt

เพื่อตอบคำถามวิจัย *"ผู้ตรวจทานโค้ดทำงานเกี่ยวกับการตรวจทานโค้ดในโครงการโอเพนซอร์สมากน้อยเพียงใดต่อสัปดาห์"* ผู้วิจัยได้ใช้สถิติความแปรปรวนและ Post-hoc Comparisons แบบ Tukey's HSD ในการวิเคราะห์ค่าเฉลี่ยของข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาที่ได้รับจากผู้ตรวจทานโค้ดในแต่ละวันของสัปดาห์

สำหรับผลลัพธ์ที่ได้จากการวิเคราะห์ หากค่า p ของสมมุติฐานคู่ใดมีค่ามากกว่า 0.05 จะทำการยอมรับ H0 แต่หากค่า p ของสมมุติฐานคู่ใดมีค่าน้อยกว่า 0.05 จะทำการปฏิเสธ H0 ซึ่งผลลัพธ์ที่ได้จากการโครงการ Eclipse เมื่อวิเคราะห์ด้วยสถิติความแปรปรวนแบบ Tukey's HSD พบว่าวันทำงาน (วันจันทร์-วันเสาร์) ของผู้ตรวจทานโค้ดในโครงการ Eclipse ไม่มีความแตกต่างกัน แต่มีความแตกต่างกันอย่างมีนัยสำคัญระหว่างการให้คำแนะนำเกี่ยวกับการบำรุงรักษาในวันอาทิตย์ ยกเว้นค่าเฉลี่ย

ของวันจันทร์ที่ผู้ตรวจทานโค้ดให้ข้อเสนอแนะเกี่ยวกับการบำรุงรักษาไม่มีความแตกต่างกับวันอาทิตย์อย่างมีนัยสำคัญ และสำหรับค่าเฉลี่ยการทำงานในวันเสาร์กับวันอาทิตย์ของผู้ตรวจทานโค้ดไม่มีความแตกต่างกันมากนัก ดังนั้นหลักฐานนี้อาจบ่งบอกว่า ผู้ตรวจทานในโครงการ Eclipse มักจะทำการตรวจสอบซอร์สโค้ดและให้คำแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ระหว่างวันจันทร์ถึงวันเสาร์มากกว่าการให้คำแนะนำในช่วงวันอาทิตย์ซึ่งเป็นวันหยุดสุดสัปดาห์ และสำหรับผลลัพธ์ที่ได้จากการโครงการ Qt เมื่อวิเคราะห์ด้วยสถิติความแปรปรวนแบบ Tukey's HSD พบว่าไม่มีความแตกต่างกันทางสถิติระหว่างวันทำงานปกติและวันหยุดสุดสัปดาห์ ดังนั้นจึงสามารถสรุปได้ว่า ผู้ตรวจทานโค้ดในโครงการ Qt มักจะทำการตรวจทานโค้ดและให้คำแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ทุกวัน



รูปที่ 4.25 กราฟแสดงจำนวนข้อเสนอแนะเกี่ยวกับการบำรุงรักษาแต่ละวันต่อสัปดาห์ในช่วงระยะเวลา 5 ปี (ค.ศ. 2012-2016) ของโครงการ Eclipse และโครงการ Qt

จากรูปที่ 4.25 แสดงแผนภาพกล่อง (Boxplot) ที่อธิบายการแจกแจงจำนวนข้อเสนอแนะเกี่ยวกับการบำรุงรักษาที่ได้จากผู้ตรวจทานโค้ดแต่ละวันในช่วงระยะเวลา 5 ปี ตั้งแต่ปี ค.ศ. 2012 - 2016 ของโครงการ Eclipse และโครงการ Qt จากรูปแสดงให้เห็นว่าจำนวนข้อเสนอแนะในวันทำงานทั่วไป (วันจันทร์ อังคาร พุธ พฤหัสบดี และศุกร์) มีค่ามัธยฐานสูงกว่าข้อเสนอแนะเกี่ยวกับการบำรุงรักษาที่ได้รับในช่วงวันเสาร์และวันอาทิตย์ หลักฐานนี้อาจเป็นสิ่งที่บ่งบอกได้ว่า ถึงแม้โครงการโอเพนซอร์สจะเป็นโครงการที่อาศัยความสมัครใจในการทำงานที่ไม่ได้รับค่าตอบแทน แต่ชุมชนผู้ตรวจทานโค้ดหรือนักพัฒนาในโครงการโอเพนซอร์สก็มักจะหยุดทำงานในช่วงวันหยุดสุดสัปดาห์เช่นกัน

บทที่ 5

สรุปผลการวิจัย

งานวิจัยนี้เป็นการศึกษาข้อเสนอแนะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ของโครงการโอเพนซอร์สภายใต้กระบวนการตรวจทานโค้ดในระบบเกอริต โดยผู้วิจัยได้ทำการศึกษาและรวบรวมคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ เพื่อนำไปวิเคราะห์ด้วยเทคนิคการทำเหมืองข้อความ สำหรับผลลัพธ์ที่ได้จากกระบวนการดำเนินงาน ผู้วิจัยได้นำเสนอผลลัพธ์และวิเคราะห์ผลลัพธ์ดังกล่าวโดยใช้สถิติ ได้แก่ การวิเคราะห์การถดถอย การวิเคราะห์สหสัมพันธ์ และการวิเคราะห์ความแปรปรวน ซึ่งผลจากการศึกษานี้จะเป็นหลักฐานเชิงประจักษ์เกี่ยวกับการบำรุงรักษาซอฟต์แวร์ที่เกิดขึ้นภายในโครงการ Eclipse และโครงการ Qt ในบทนี้ ผู้วิจัยได้ทำการนำเสนอด้วยกัน 4 หัวข้อ นั่นคือ 1) อภิปรายและสรุปผลการวิจัย 2) ข้อเสนอแนะและข้อจำกัดของงานวิจัย 3) ภัยคุกคามต่อความถูกต้องที่เกิดขึ้นในการทำงานวิจัย 4) งานวิจัยในอนาคต โดยรายละเอียดในแต่ละหัวข้อดังนี้

5.1 การอภิปรายและสรุปผลการวิจัย

ผู้วิจัยได้ทำการศึกษาและรวบรวมคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ พร้อมทั้งสร้างชุดคำสั่งในโปรแกรม R สำหรับพัฒนากระบวนการค้นหาและวิเคราะห์ข้อเสนอแนะของโครงการโอเพนซอร์ส Eclipse และโครงการ Qt ภายใต้กระบวนการตรวจทานโค้ด โดยมีวัตถุประสงค์เพื่อที่จะตอบคำถามวิจัยที่ผู้วิจัยได้กำหนดไว้ในงานวิจัยนี้ คือ

1) คุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์สแบ่งออกเป็นกี่คุณลักษณะ
2) นักพัฒนาซอฟต์แวร์ให้ความสนใจกับความสามารถในการบำรุงรักษาซอฟต์แวร์ภายใต้การตรวจทานโค้ดในโครงการโอเพนซอร์สหรือไม่

3) คุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ประเภทใดที่นักพัฒนาให้ความสนใจเป็นพิเศษ
4) ขนาดกลุ่มของนักพัฒนาซอฟต์แวร์ส่งผลต่อจำนวนข้อเสนอแนะที่ได้รับการแก้ไขโค้ดเกี่ยวกับความสามารถในการบำรุงรักษาซอฟต์แวร์หรือไม่
5) นักพัฒนาซอฟต์แวร์ทำงานเกี่ยวกับการตรวจทานโค้ดในโครงการโอเพนซอร์สมากน้อยเพียงใดต่อสัปดาห์

การรวบรวมข้อเสนอแนะที่เกิดขึ้นในกระบวนการตรวจทานโค้ดของโครงการโอเพนซอร์สในช่วงระยะเวลา 5 ปี นั่นคือ ค.ศ. 2012 – 2016 โดยโครงการ Eclipse มีข้อเสนอแนะจำนวน 108,357 ข้อเสนอแนะ และโครงการ Qt มีข้อเสนอแนะจำนวน 309,165 ข้อเสนอแนะ เพื่อที่จะค้นหาข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ ผู้วิจัยจึงได้ทำการพิจารณาคุณลักษณะย่อยที่มีความเกี่ยวข้องกับการบำรุงรักษาหรือคุณลักษณะย่อยที่มีผลกระทบต่อการบำรุงรักษาจากวารสารวิชาการของ Ghosh และคณะ (2011) จากการพิจารณาคุณลักษณะย่อย พบว่ามีเพียง 33 ประเภท ที่มีความเกี่ยวข้องกับคำนิยามของความสามารถในการบำรุงรักษาซอฟต์แวร์ ผู้วิจัยจึงได้นำคุณลักษณะดังกล่าวมาเป็นคำหลักเพื่อใช้ในการค้นหาข้อเสนอแนะ และได้ทำการค้นหาคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์เพิ่มเติมจากที่มีอยู่ โดยใช้อัลกอริทึม LDA ซึ่งคุณลักษณะที่ค้นพบเพิ่มเติมจากโครงการโอเพนซอร์สมีด้วยกัน 2 คุณลักษณะ คือ Duplicate function และ Cohesion module ดังนั้นข้อมูลเชิงประจักษ์นี้สามารถตอบคำถามวิจัยได้ว่า คุณลักษณะที่เกี่ยวข้องกับความสามารถในการบำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์สมีด้วยกันทั้งหมด 35 คุณลักษณะ ดังนี้

- Accuracy
- Adaptability
- Analyzability
- Augmentability
- Availability
- Changeability
- Cohesion module
- Completeness
- Complexity
- Comprehensibility
- Consistency

- Correctability
- Duplicate function
- Durability
- Efficiency
- Effort
- Expandability
- Extensibility
- Flexibility
- Implementation
- Instrumentation
- Integrability
- Localization
- Modifiability
- Modularity
- Perfectiveness
- Portability
- Readability
- Reusability
- Simplicity
- Stability
- Standardization
- Testability
- Traceability
- Understandability

ผลลัพธ์ที่ได้จากการค้นหาข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ที่ได้รับจากผู้ตรวจทานโค้ดในโครงการโอเพนซอร์ส โดยโครงการ Eclipse พบข้อเสนอแนะเกี่ยวกับการบำรุงรักษาจำนวน 36.58% และโครงการ Qt พบข้อเสนอแนะเกี่ยวกับการบำรุงรักษาจำนวน 30.28% เนื่องจากผู้ตรวจทานโค้ดจะให้คำแนะนำในการแก้ไขปรับปรุงโค้ดทางด้านการบำรุงรักษาหลังจากที่ซอฟต์แวร์ผ่านการพัฒนาและการทดสอบระบบแล้วเท่านั้น นอกเหนือจากข้อเสนอแนะทางด้านการบำรุงรักษาประมาณ 30% ที่พบในโครงการแล้ว ผู้วิจัยยังพบข้อเสนอแนะที่เกิดขึ้นระหว่างการพัฒนา

ฟังก์ชันหลักและฟังก์ชันย่อย ๆ ของการทำงานระบบและข้อเสนอแนะเกี่ยวกับคุณลักษณะด้านต่าง ๆ เช่น ความมั่นคง (Bosu, et al., 2014) ซึ่งนอกเหนือจากการบำรุงรักษาแล้ว ความมั่นคงถือว่าเป็นคุณลักษณะที่สำคัญมากเช่นกัน เนื่องจากผู้ใช้งานทั่วโลกต้องการความมั่นใจว่า ซอฟต์แวร์โอเพนซอร์สที่จะนำซอฟต์แวร์ไปใช้งานต้องเป็นซอฟต์แวร์ที่สามารถรักษาความลับ (Confidentiality) และรักษาถูกต้อง (Integrity) ของผู้ใช้งาน เพื่อป้องกันการเข้าถึงหรือแก้ไขโปรแกรมคอมพิวเตอร์หรือข้อมูลโดยไม่ได้รับอนุญาต รวมถึงซอฟต์แวร์ที่จะถูกนำไปเผยแพร่ต้องมีความพร้อมที่จะใช้ (Availability) ดังนั้นทีมนักพัฒนาหลักที่ทำหน้าที่ทางด้านความมั่นคงมักจะพยายามค้นหาช่องโหว่ที่อาจจะเกิดขึ้นระหว่างการแก้ไขซอร์สโค้ด (Vulnerable Code Changes : VCC) อยู่เสมอ

เมื่อผู้วิจัยได้ทำการตรวจสอบจำนวนข้อเสนอแนะที่ได้รับการแก้ไขโค้ดทางด้านการบำรุงรักษาซอฟต์แวร์พบว่า โครงการ Eclipse มีจำนวนข้อเสนอแนะที่ได้รับการแก้ไขปรับปรุงโค้ดทางด้านการบำรุงรักษาจำนวน 57.22% และโครงการ Qt มีจำนวนข้อเสนอแนะที่ได้รับการแก้ไขปรับปรุงโค้ดทางด้านการบำรุงรักษา 63.10% ซึ่งผลลัพธ์ดังกล่าวแสดงให้เห็นว่า นักพัฒนาในโครงการโอเพนซอร์สทำการแก้ไขโค้ดทางด้านการบำรุงรักษามากกว่า 50% ดังนั้นจึงสามารถตอบคำถามวิจัยได้ว่า นักพัฒนาซอฟต์แวร์ทั้ง 2 โครงการ ให้ความสนใจกับความสามารถในการบำรุงรักษาซอฟต์แวร์ภายใต้การตรวจทานโค้ดในโครงการโอเพนซอร์ส สาเหตุที่ชุมชนนักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์สทั้ง 2 โครงการ ให้ความสนใจกับการบำรุงรักษาซอฟต์แวร์เพิ่มมากขึ้น อาจเป็นเพราะหลังจากที่ซอฟต์แวร์ Eclipse และ Qt ถูกนำไปใช้งานอย่างแพร่หลายในการพัฒนาแอปพลิเคชันที่ใช้ในอุตสาหกรรม ธุรกิจเชิงพาณิชย์ หน่วยงานรัฐบาล หรือการศึกษาวิจัยในศาสตร์แขนงต่าง ๆ ที่เกี่ยวข้องกับเทคโนโลยีแล้ว ผู้ใช้ก็จะมีการให้ข้อมูลเกี่ยวกับการใช้งาน หรือที่เรียกกันว่า การให้คำแนะนำ/คำติชม (Feedback) แต่ในการให้ข้อมูลกลับมาของผู้ใช้งาน นอกจากเป็นการแจ้งปัญหาที่เกิดขึ้นระหว่างการใช้งานแล้ว ผู้ใช้งานยังมีการร้องขอความต้องการอื่น ๆ ที่จำเป็น ดังนั้นชุมชนนักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์สและนักพัฒนาซอฟต์แวร์รุ่นใหม่จึงต้องทำการพัฒนาฟังก์ชันหรือคุณลักษณะใหม่ เพื่อรองรับความต้องการของผู้ใช้งานทั่วโลก ด้วยเหตุนี้จึงทำให้ซอฟต์แวร์กลับเข้าสู่วงจรชีวิตของการพัฒนาซอฟต์แวร์อีกครั้ง โดยเฉพาะขั้นตอนการบำรุงรักษาซอฟต์แวร์ที่นักพัฒนาซอฟต์แวร์จะต้องทำการแก้ไขปรับปรุงซอร์สโค้ด โดยการเปลี่ยนแปลงนั้นจะต้องไม่ส่งผลกระทบต่อการทำงานของระบบเดิม นอกจากนี้ผลวิจัยยังแสดงให้เห็นถึงแนวโน้มของการให้คำแนะนำและการแก้ไขโค้ดทางด้านการบำรุงรักษา ซึ่งถือว่าเป็นสัญญาณที่ดีสำหรับคุณภาพของซอฟต์แวร์ที่ทำให้สามารถทำนายได้ว่า ชุมชนนักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์สทั้ง 2 โครงการ จะให้ความสนใจกับการบำรุงรักษาเพิ่มมากขึ้นในอนาคต

ผลลัพธ์ที่ได้จากการดำเนินงานนี้เป็นการตรวจสอบความสนใจที่มีต่อความสามารถในการบำรุงรักษา ซึ่งมีความแตกต่างกับงานวิจัยอื่น ๆ เพราะนักวิจัยทางด้านวิศวกรรมซอฟต์แวร์มักจะทำการศึกษาโครงการ Eclipse และโครงการ Qt ในเรื่องการบำรุงรักษาทางด้านอื่น ตัวอย่างเช่น

การใช้ตัวชี้วัดในการตรวจสอบซอร์สโค้ดเพื่อตรวจสอบการบำรุงรักษา เช่น งานวิจัยของ Yamashita และคณะ ได้ทำการตรวจสอบไฟล์ที่เสี่ยงต่อการเพิ่มขนาดและการเกิดความซับซ้อนของซอฟต์แวร์ (Yamashita, et al., 2016) หรืองานวิจัยของ Caglayan และคณะ ที่ใช้ตัวชี้วัดเพื่อทำนายข้อบกพร่องในโครงการโอเพนซอร์ส (Caglayan, et al., 2009) รวมถึงงานวิจัยที่เกี่ยวข้องกับการวัดค่าความสามารถในการบำรุงรักษาซอฟต์แวร์ (Maintainability Index : MI) เช่น งานวิจัยของ Counsell และคณะ ได้ทำการศึกษาความสัมพันธ์ระหว่างค่าของความสามารถในการบำรุงรักษากับการจับคู่หรือการเชื่อมต่อฟังก์ชันเข้าด้วยกัน (Coupling) (Counsell, et al., 2015) ดังนั้นผลลัพธ์ที่ได้จากงานวิจัยที่แสดงให้เห็นถึงความสนใจที่มีต่อ “ความสามารถในการบำรุงรักษา” อาจมีผลลัพธ์ที่แตกต่างกับงานวิจัยอื่น ๆ เพราะมีปัจจัยในเรื่องของเทคนิค กระบวนการ และเครื่องมือที่นำมาใช้ดำเนินการ รวมไปถึงสภาพแวดล้อมต่าง ๆ ของโครงการโอเพนซอร์สที่มีการเปลี่ยนแปลงไป สิ่งเหล่านี้อาจทำให้ผลลัพธ์เกี่ยวกับการบำรุงรักษาซอฟต์แวร์มีความแตกต่างกันถึงแม้ว่าจะทำการศึกษาในโครงการเดียวกัน

ปัจจัยที่มีอิทธิพลต่อจำนวนการเกิดข้อเสนอแนะเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ที่ได้รับจากผู้ตรวจทานโค้ดมีด้วยกัน 2 ปัจจัย ได้แก่ (1) บทบาทและขนาดกลุ่มของผู้ตรวจทานโค้ด (2) เวลาการทำงานของผู้ตรวจทานโค้ด จากการศึกษาพบว่านักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์สมีบทบาทด้วยกัน 2 บทบาท นั่นคือ บทบาทนักพัฒนาที่ทำหน้าที่ในการพัฒนาซอฟต์แวร์และแก้ไขปรับปรุงโค้ดตามคำแนะนำของนักพัฒนาคนอื่น ๆ และบทบาทนักพัฒนาที่ทำหน้าที่ในการตรวจทานซอร์สโค้ดที่นักพัฒนาคนอื่น ๆ ได้พัฒนาขึ้น ในการตรวจสอบขนาดกลุ่มหรือทีมของนักพัฒนาซอฟต์แวร์ที่ทำงานเกี่ยวกับการตรวจทานโค้ด พบว่าปัจจัยดังกล่าวส่งผลต่อจำนวนข้อเสนอแนะทางด้านการบำรุงรักษาซอฟต์แวร์ที่เพิ่มขึ้นในแต่ละเดือนของช่วงปี ค.ศ. 2012 - 2016 นอกจากนี้ผลลัพธ์จากการตรวจสอบการทำงานของผู้ตรวจทานโค้ดพบว่า ผู้ตรวจทานโค้ดในโครงการ Eclipse และโครงการ Qt มักจะให้คำแนะนำเกี่ยวกับการบำรุงรักษาในช่วงวันจันทร์-ศุกร์ และมักจะหยุดทำงานในวันเสาร์-อาทิตย์ แต่ก็มีผู้ตรวจทานโค้ดบางคนที่ทำงานในวันหยุดสุดสัปดาห์เช่นกัน จากข้อมูลดังกล่าวทำให้สามารถสรุปได้ว่า ถึงแม้ว่าโครงการโอเพนซอร์สจะเป็นโครงการที่มีความอิสระในการทำงาน แต่วันหยุดสุดสัปดาห์ก็เป็นปัจจัยที่ส่งผลต่อการทำงานเกี่ยวกับการตรวจทานโค้ดจึงทำให้นักพัฒนาซอฟต์แวร์มักจะให้คำแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์และทำการปรับปรุงโค้ดในช่วงวันทำงานปกติ นอกเหนือจากปัจจัยข้างต้น งานวิจัยของ Bosu และ Carver ได้ทำการตรวจสอบว่า การมีชื่อเสียงหรือเป็นที่รู้จักในกลุ่มนักพัฒนาซอฟต์แวร์เป็นปัจจัยที่ส่งผลต่อการตรวจทานโค้ดในโครงการโอเพนซอร์สหรือไม่ (Bosu and Carver, 2014) ผลที่ได้ คือ เมื่อนักพัฒนาซอฟต์แวร์ที่ถูกยอมรับให้เข้าร่วมทีมนักพัฒนาซอฟต์แวร์หลัก (ซึ่งเป็นผู้ที่มีส่วนร่วมในการพัฒนาและวิวัฒนาการของโครงการโอเพนซอร์ส (Ye and Kishida, 2003) เป็นเวลานาน) มักจะทำการตอบรับคำร้องขอการตรวจสอบซอร์สโค้ดในครั้งแรกและเสร็จสิ้นกระบวนการตรวจทานโค้ดในเวลาที่รวดเร็ว อีกทั้งยังมีแนวโน้มที่จะยอมรับการเปลี่ยนแปลงโค้ดและอนุญาตให้นำไป

รวมกับโค้ดต้นฉบับ (Code Base) ดังนั้นจึงสามารถสรุปได้ว่า การที่นักพัฒนาซอฟต์แวร์ได้รับการยอมรับให้เป็นหนึ่งในทีมนักพัฒนาหลัก จึงทำให้สามารถสร้างแรงจูงใจหรือกระตุ้นให้นักพัฒนาซอฟต์แวร์มุ่งเน้นที่จะทำงานเกี่ยวกับการตรวจทานโค้ดมากยิ่งขึ้น ซึ่งสังเกตได้ว่างานวิจัยที่เกี่ยวข้องกับการตรวจทานโค้ดของนักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์ส ไม่ว่าจะเป็นงานวิจัยที่ผู้วิจัยได้ทำการศึกษาปัจจัยขนาดกลุ่มหรือการทำงานของนักพัฒนาซอฟต์แวร์ที่ส่งผลต่อจำนวนข้อเสนอแนะที่ได้รับจากนักพัฒนาซอฟต์แวร์ และงานวิจัยของ Bosu และ Carver เป็นการศึกษาปัจจัยทางด้านชื่อเสียงของนักพัฒนาซอฟต์แวร์ที่ส่งผลต่อการทำงานเกี่ยวกับการตรวจทานโค้ดที่เกิดขึ้นในโครงการโอเพนซอร์ส (Bosu and Carver, 2014) ผลลัพธ์ที่ได้จะมีความสอดคล้องกันในเรื่องของการทำงานทางด้านการตรวจทานโค้ด ดังนั้นจึงสามารถสรุปได้ว่า งานวิจัยเกี่ยวกับการตรวจสอบปัจจัยที่ส่งผลต่อการทำงานของนักพัฒนาซอฟต์แวร์ภายใต้กระบวนการตรวจทานโค้ดอาจให้ผลลัพธ์ที่มีความคล้ายคลึงกับในโครงการ Eclipse และโครงการ Qt

จากผลการวิจัยแสดงให้เห็นว่าผู้ตรวจทานโค้ดในโครงการ Eclipse และโครงการ Qt มีการให้คำแนะนำเพื่อปรับปรุงคุณภาพทางด้านการบำรุงรักษาซอฟต์แวร์เพิ่มมากขึ้น สาเหตุอาจเป็นเพราะผู้ตรวจทานโค้ดที่คอยให้คำแนะนำแก่นักพัฒนา มักจะเป็นผู้ตรวจทานโค้ดคนเดิม ๆ ที่มีประสบการณ์ทางด้านการพัฒนาระบบและมีความรู้ความเชี่ยวชาญเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ นอกจากนี้ผู้ตรวจทานโค้ดบางคนยังเป็นทีมพัฒนาหลักของโครงการอีกด้วย นอกจากนี้ผลการวิจัยยังพบว่าทั้ง 2 โครงการมีจำนวนของผู้ตรวจทานโค้ดเพิ่มขึ้น ซึ่งเป็นสิ่งที่อาจจะบ่งบอกได้ว่า ในอนาคตจะมีนักพัฒนาซอฟต์แวร์ที่ให้ความสนใจกับโครงการโอเพนซอร์สเพิ่มมากขึ้น ด้วยเหตุนี้ผลลัพธ์ที่ได้จึงสอดคล้องกับจำนวนการแก้ไขปรับปรุงโค้ดตามคำแนะนำเกี่ยวกับการบำรุงรักษาที่เพิ่มมากขึ้นตามไปด้วย โดยเฉพาะคุณลักษณะความสามารถในการเปลี่ยนแปลง (Changeability) ที่นักพัฒนาซอฟต์แวร์ทั้ง 2 โครงการให้ความสนใจเป็นพิเศษ แต่เมื่อทำการศึกษาการทำงานของนักพัฒนาซอฟต์แวร์ทำให้ทราบว่า ชุมชนโอเพนซอร์สทั้ง 2 โครงการ ไม่ได้มุ่งเน้นที่จะให้คำแนะนำหรือแก้ไขปรับปรุงโค้ดเฉพาะความสามารถในการเปลี่ยนแปลงหรือคุณลักษณะใดคุณลักษณะหนึ่ง แต่จะให้คำแนะนำเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ที่ส่งผลต่อการทำงานหลักของซอฟต์แวร์ และทำการแก้ไขปรับปรุงโค้ดตามปัญหาที่เกิดขึ้นระหว่างการพัฒนา ซึ่งปัญหาดังกล่าวจะได้รับการรายงานจากนักพัฒนาซอฟต์แวร์หรือถูกค้นพบโดยผู้ใช้งานซอฟต์แวร์ โดยปกติแล้วโครงการซอฟต์แวร์ที่มีขนาดใหญ่มักจะมีการรายงานข้อบกพร่องจำนวนมากที่ปรากฏอยู่ในซอร์สโค้ด (Anjali, 2015) จึงเป็นเรื่องปกติที่โครงการโอเพนซอร์สหลายโครงการมีการรายงานข้อผิดพลาดที่เกิดขึ้นในแต่ละวัน ซึ่งสอดคล้องกับงานวิจัยของ Zhang และคณะ ได้ทำการตรวจสอบการรายงานข้อบกพร่องในแต่ละวันของโครงการ Eclipse[29] (Zhang, et al., 2015) ผลที่ได้คือ

[29] https://bugs.eclipse.org/bugs/

มีการรายงานข้อผิดพลาดหรือข้อบกพร่องที่ค้นพบโดยเฉลี่ย 29 รายงานต่อหนึ่งวัน อย่างไรก็ตามหากนักพัฒนาซอฟต์แวร์หรือผู้บำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์สไม่มีความพยายามที่จะป้องกันและลดการเกิดข้อผิดพลาดหรือข้อบกพร่องมากมายที่เกิดขึ้นก็อาจเป็นต้นเหตุที่ทำให้เกิดปัญหาร้ายแรงในอนาคตจนนำไปสู่ความล้มเหลวของซอฟต์แวร์ เช่นเดียวกับงานวิจัยของ Coelho และ Valente ที่มีการระบุว่า หนึ่งในสาเหตุที่ทำให้โครงการโอเพนซอร์สเกิดความล้มเหลว คือ ขาดกระบวนการบำรุงรักษาที่ดีหรือมีค่าการบำรุงรักษาซอฟต์แวร์ที่ต่ำเกินไป (Low Maintainability) (Coelho and Valente, 2017) ในปัจจุบันโครงการโอเพนซอร์สที่ได้รับความนิยมมักจะมีการพัฒนาซอฟต์แวร์อย่างต่อเนื่อง ทำให้ซอฟต์แวร์มีจำนวนซอร์สโค้ดที่เพิ่มมากขึ้นจึงอาจทำให้มีโอกาสที่จะเกิดความซับซ้อนและการเกิดข้อบกพร่อง เช่น การสร้างผลิตภัณฑ์ที่ไม่ตรงกับความต้องการ หรือการสร้างผลิตภัณฑ์ที่ถูกด้วยวิธีที่ผิด ดังนั้นจึงผู้วิจัยมีความเห็นว่า ปัญหาเหล่านี้จะเกิดน้อยลงหากนักพัฒนาซอฟต์แวร์ให้ความสำคัญกับความสามารถในการบำรุงรักษาก็จะสามารถกำจัดข้อบกพร่องที่เกิดขึ้นก่อนที่จะมีการเผยแพร่ซอฟต์แวร์สู่สาธารณชนและถูกนำไปใช้งานจริง รวมถึงยังช่วยป้องกันผลกระทบจากการแก้ไขโค้ดระหว่างการพัฒนาและทำให้โครงการโอเพนซอร์สสามารถผลิตซอฟต์แวร์ที่มีคุณภาพมากยิ่งขึ้น

ถึงแม้ว่าผลลัพธ์ที่ได้จากงานวิจัยนี้เป็นเพียงการสรุปผลขั้นพื้นฐาน แต่ข้อมูลที่มีอยู่ก็เพียงพอสำหรับนำมาเป็นแนวทางในการพัฒนาหรือปรับปรุงคุณภาพของซอฟต์แวร์ในโครงการโอเพนซอร์ส ดังนั้นผู้วิจัยจึงคาดหวังว่าข้อมูลเชิงประจักษ์ในงานวิจัยนี้จะเป็นประโยชน์สำหรับการปรับปรุงซอร์สโค้ดแก่ชุมชนนักพัฒนาในโครงการโอเพนซอร์ส รวมถึงเป็นการเพิ่มความรู้และหลักฐานที่จะทำให้นักวิจัยทางด้านวิศวกรรมซอฟต์แวร์ที่ต้องการศึกษาเกี่ยวกับโครงการโอเพนซอร์สสามารถนำไปเป็นแนวทางในการศึกษาต่อยอดได้ อีกทั้งยังแสดงให้เห็นถึงแนวโน้มของการตรวจทานโค้ดที่มีต่อคุณภาพของซอฟต์แวร์และแนวโน้มของการแก้ไขปรับปรุงซอร์สโค้ดเกี่ยวกับการบำรุงรักษาในโครงการโอเพนซอร์สที่จะเป็นสิ่งที่ช่วยในการตัดสินใจให้กับนักพัฒนาที่สนใจจะเข้าร่วมการพัฒนาในโครงการโอเพนซอร์สว่าควรให้ความสำคัญกับการบำรุงรักษาซอฟต์แวร์เหมือนกับนักพัฒนาซอฟต์แวร์คนอื่น ๆ ในโครงการโอเพนซอร์ส รวมถึงเตรียมพร้อมที่จะรับมือกับการแก้ไขปรับปรุงโค้ดในสถานการณ์หรือสิ่งแวดล้อมต่าง ๆ ที่อาจเปลี่ยนแปลงในอนาคต

5.2 ข้อเสนอแนะและข้อจำกัดของงานวิจัย

จากการศึกษางานวิจัยนี้ ตั้งแต่กระบวนการคิดริเริ่มงานวิจัย จนกระทั่งการขั้นตอนสุดท้ายในการสรุปผลวิจัย ผู้วิจัยมีข้อเสนอแนะบางประการแก่นักวิจัยหรือนักพัฒนาซอฟต์แวร์ที่สนใจในงานวิจัยทางด้านวิศวกรรมซอฟต์แวร์ โดยเฉพาะเกี่ยวกับทางด้านการบำรุงรักษาซอฟต์แวร์ ซึ่งผู้วิจัยได้ให้ข้อเสนอแนะและคำแนะนำที่ได้จากการเรียนรู้ในงานวิจัยดังต่อไปนี้

5.2.1 ข้อเสนอแนะเกี่ยวกับข้อมูลที่นำมาใช้ในงานวิจัย

การกำหนดโครงการซอฟต์แวร์โอเพนซอร์สที่จะนำมาศึกษา ควรตรวจสอบให้แน่ใจว่าเป็นโครงการที่สามารถเข้าถึงแหล่งข้อมูลได้ เพราะแหล่งที่ใช้ในการจัดเก็บข้อมูลของโครงการโอเพนซอร์สมีทั้งแบบที่สามารถให้บุคคลที่สนใจเข้าถึงข้อมูลได้ และแบบที่ไม่อนุญาตให้บุคคลทั่วไปสามารถเข้าถึงข้อมูลได้ นอกจากบุคคลที่มีส่วนเกี่ยวข้องที่จะสามารถเข้าถึงข้อมูลได้ เช่น CodeFlow ของบริษัทไมโครซอฟท์ สำหรับงานวิจัยนี้ ผู้วิจัยได้ใช้ระบบเกอริต ซึ่งเป็นระบบที่เปิดให้บุคคลที่สนใจสามารถเข้าถึงข้อมูลได้ แต่ก็มีข้อจำกัดบางประการ คือ แต่ละโครงการมีการเข้าถึงแหล่งข้อมูลที่แตกต่างกัน โดยโครงการ Eclipse และโครงการ Qt ที่ผู้วิจัยนำมาศึกษา ผู้วิจัยได้ทำการสืบค้นข้อมูลจากการทำงานผ่านเชลล์ (Shell) ซึ่งเป็นโปรแกรมที่ใช้ในการติดต่อสื่อสารระหว่างผู้ใช้งานและเว็บเซอร์วิสของระบบเกอริต

5.2.2 ข้อเสนอแนะทางด้านเครื่องมือ

สำหรับกระบวนดำเนินงาน ควรมีการเลือกเครื่องมือที่เหมาะสมสำหรับแต่ละขั้นตอนในการดำเนินงาน เช่น เครื่องมือสำหรับการทำความสะอาดข้อมูล เครื่องมือสำหรับใช้ในการเขียนชุดคำสั่งหรือฟังก์ชันสำหรับการวิเคราะห์ข้อมูล เครื่องมือสำหรับการคำนวณทางด้านสถิติ ในปัจจุบันเครื่องมือสำหรับการดำเนินงานมีมากมายหลากหลายแบบให้เลือกใช้ โดยมีทั้งแบบที่เสียค่าใช้จ่ายสำหรับซอฟต์แวร์และแบบที่ไม่เสียค่าใช้จ่ายสำหรับซอฟต์แวร์หรือก็คือสามารถดาวน์โหลด เพื่อนำมาใช้งานได้ฟรีนั่นเอง ซึ่งการเลือกเครื่องมือที่จะใช้ในงานวิจัย ควรเลือกเครื่องมือที่เหมาะสมสำหรับชุดข้อมูลและการดำเนินงานต่าง ๆ โดยหากสามารถเลือกเครื่องมือที่มีคุณสมบัติในการทำงานได้ตามที่ต้องการสำหรับงานวิจัยก็จะเป็นการประหยัดค่าใช้จ่ายและเวลาในการประมวลผลหรือใช้งานโปรแกรม ซึ่งในงานวิจัยนี้ผู้วิจัยได้เลือกโปรแกรม R ซึ่งเป็นโปรแกรมที่มีคุณสมบัติในการทำงานที่หลากหลายและได้รับความนิยมในการใช้งานจากนักวิจัยทางด้านวิศวกรรมซอฟต์แวร์ นอกจากนี้ฮาร์ดแวร์ที่ใช้ในการประมวลผลข้อมูล เช่น คอมพิวเตอร์หรือโน้ตบุ๊กจะต้องมีสมรรถภาพในการประมวลผลข้อมูลจำนวนมากหรือข้อมูลขนาดใหญ่ได้

5.3 ภัยคุกคามต่อความถูกต้องที่เกิดขึ้นในการทำงานวิจัย

อุปสรรคหรือภัยคุกคามต่อความถูกต้อง (Threats to validity) เป็นปัจจัยที่ส่งผลให้ความถูกต้องและความน่าเชื่อถือของการศึกษาวิจัยลดลง รวมไปถึงการทำให้ผลลัพธ์เปลี่ยนไปจากความเป็นจริง ซึ่งในหัวข้อนี้จะเป็นการระบุถึงภัยคุกคามต่อความถูกต้องที่ได้เรียนรู้จากงานวิจัยนี้ โดยภัยคุกคามต่อความถูกต้องสำหรับงานวิจัยนี้แบ่งออกเป็น 3 รูปแบบ คือ 1) ความถูกต้องเชิงโครงสร้าง 2) ความถูกต้องภายใน และ 3) ความถูกต้องภายนอก ซึ่งภัยคุกคามแต่ละประเภทมีรายละเอียดดังนี้

5.3.1 ความถูกต้องเชิงโครงสร้าง (Construct Validity)

ภัยคุมคามต่อความถูกต้องเชิงโครงสร้าง เป็นการพิจารณาถึงคุณภาพของเครื่องมือที่จะใช้ในงานวิจัย เพื่อตรวจสอบว่าเครื่องมือนั้นสามารถใช้ในการวัดผล วิเคราะห์ข้อมูล และประเมินผลได้ตามวัตถุประสงค์ของงานวิจัยหรือไม่ นอกจากนี้ยังรวมไปการพิจารณาเครื่องมือที่ใช้ในการสืบค้นและรวบรวมข้อมูลเพื่อที่จะได้นำมาซึ่งความถูกต้องของข้อมูลที่จะนำไปใช้ในงานวิจัย สำหรับงานวิจัยในครั้งนี้ผู้วิจัยได้ทำการระบุถึงภัยคุกคามที่อาจส่งผลต่อความถูกต้องเชิงโครงสร้างไว้ดังนี้

1) งานวิจัยนี้ได้ทำการรวบรวมข้อเสนอแนะของโครงการซอฟต์แวร์โอเพนซอร์สภายใต้กระบวนการตรวจทานโค้ดจากระบบที่มีชื่อว่า “เกอริต” ซึ่งผู้วิจัยได้ทำการศึกษาโครงสร้างการจัดเก็บข้อมูลของระบบเกอริต เพื่อนำมาใช้ในการออกแบบและพัฒนาซอฟต์แวร์สำหรับสืบค้นและจัดเก็บข้อมูล โดยข้อมูลที่ได้ทำการรวบรวมมาเป็นชุดข้อความหรือตัวอักษร จึงทำให้ข้อมูลบางส่วนอาจไม่ครบถ้วนสมบูรณ์ตามที่แสดงบนระบบเกอริต เช่น สัญลักษณ์และรูปภาพต่าง ๆ อย่างไรก็ตามข้อมูลที่ผู้วิจัยได้ทำการรวบรวมมานั้นก็เป็นข้อมูลหลักที่ผู้วิจัยต้องการนำมาทำการวิเคราะห์คุณภาพของโครงการโอเพนซอร์ส โดยเฉพาะทางด้านความสามารถในการบำรุงรักษา ซึ่งถือได้ว่าเป็นข้อมูลที่สำคัญและเพียงพอสำหรับการสรุปผลเพื่อตอบคำถามวิจัยและตรงตามวัตถุประสงค์ของงานวิจัยที่ได้กำหนดไว้

2) สำหรับขั้นตอนการดำเนินงานวิจัย ผู้วิจัยได้ใช้โปรแกรม R ในกระบวนการทำเหมืองข้อความและการประมวลผลสำหรับวิเคราะห์ข้อมูล เพื่อค้นหาข้อเสนอแนะเกี่ยวกับการบำรุงรักษาซอฟต์แวร์และการวิเคราะห์ผลลัพธ์ด้วยวิธีการทางสถิติต่าง ๆ เช่น การวิเคราะห์ความแปรปรวน การวิเคราะห์สหสัมพันธ์ และการวิเคราะห์การถดถอย ดังนั้นหากผู้วิจัยใช้โปรแกรมหรือเครื่องมืออื่น ๆ เกี่ยวกับการทำเหมืองข้อความก็อาจส่งคืนผลลัพธ์ที่แตกต่างกัน

3) งานวิจัยนี้มีการนำคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ 33 คุณลักษณะมาจากวารสารวิชาการของ Ghosh และคณะ (2011) และผู้วิจัยได้ทำการค้นพบคุณลักษณะ

ประเภทใหม่เพิ่มอีก 2 คุณลักษณะ โดยการใช้อัลกอริทึม LDA ซึ่งเป็นอัลกอริทึมที่ได้รับการยอมรับโดยทั่วไปในงานวิจัยทางด้านวิศวกรรมซอฟต์แวร์ หากนักวิจัยทางด้านวิศวกรรมซอฟต์แวร์ทำการค้นหาคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาด้วยวิธีอื่น ๆ อาจเจอคุณลักษณะใหม่เพิ่มเติมจากที่ผู้วิจัยได้ทำการค้นพบจากโครงการโอเพนซอร์ส

5.3.2 ความถูกต้องภายใน (Internal Validity)

ภัยคุมคามต่อความถูกต้องภายใน (Internal Validity) เป็นการพิจารณาผลของการวิจัยที่เกิดขึ้นว่าสามารถตอบคำถามวิจัยได้ถูกต้องตามหลักความเป็นจริงหรือไม่ หรือเป็นคุณลักษณะที่สามารถสรุปได้ว่า ผลลัพธ์ที่ได้จากการวิจัยที่เกิดขึ้นนั้นเป็นผลที่มาจากสิ่งที่ได้ดำเนินการภายในงานวิจัยเพียงอย่างเดียว ไม่ได้เกิดมาจากปัจจัยอื่น ๆ ที่นักวิจัยไม่ได้ทำการศึกษา สำหรับงานวิจัยในครั้งนี้ ผู้วิจัยได้ทำการระบุถึงภัยคุกคามที่อาจส่งผลต่อความถูกต้องภายในไว้ดังนี้

1) งานวิจัยนี้เป็นการนำข้อเสนอแนะที่เกิดขึ้นในกระบวนการตรวจทานโค้ดของโครงการ Eclipse และโครงการ Qt มาทำการค้นหาข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ สำหรับกระบวนการค้นหาข้อเสนอแนะ ผู้วิจัยได้ทำการรวบรวมคำหลักและคำที่มีความหมายสื่อถึงคำหลักของแต่ละคุณลักษณะย่อยที่มีความเกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์มาเป็นตัวช่วยในการค้นหา ซึ่งผู้วิจัยจะทำการวิเคราะห์เฉพาะข้อเสนอแนะที่มีการระบุถึงคำหลักและคำที่มีความหมายสื่อถึงคำหลักที่ผู้วิจัยได้กำหนดไว้เท่านั้น หรือในกรณีที่ข้อเสนอแนะของผู้ตรวจทานโค้ดมีการใช้คำที่ไม่ตรงตามที่ผู้วิจัยได้กำหนดไว้ ข้อเสนอแนะนั้นก็จะไม่ถูกนำมาวิเคราะห์นั่นเอง

เมื่อได้ผลลัพธ์จากกระบวนการทำเหมืองข้อความแล้ว ผู้วิจัยได้ทำการตรวจสอบผลลัพธ์ที่ได้ โดยการอ่านข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ทั้งหมดก่อนที่จะนำไปประมวลผลต่อเพื่อวิเคราะห์ว่าข้อเสนอแนะเหล่านั้นได้รับการแก้ไขปรับปรุงจากนักพัฒนาหรือไม่ อย่างไรก็ตามการอ่านข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาอาจก่อให้เกิดความลำเอียง (Bias) ได้ ผู้วิจัยจึงได้พยายามลดความลำเอียง โดยการตรวจสอบข้อเสนอแนะหลายครั้งเพื่อให้มั่นใจว่าข้อเสนอแนะเหล่านั้นมีความเกี่ยวข้องกับความสามารถในการบำรุงรักษาจริง ซึ่งขั้นตอนของการตรวจสอบข้อเสนอแนะดังกล่าว ผู้วิจัยจะพิจารณาข้อเสนอแนะโดยตรวจสอบดูว่าในข้อเสนอแนะนั้นมีคำหลักหรือคำที่สื่อถึงคุณลักษณะที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ปรากฏอยู่ในข้อความตามที่กำหนดไว้หรือไม่ นอกจากนี้ผู้วิจัยได้ให้ผู้เชี่ยวชาญเกี่ยวกับการตรวจทานโค้ดทำการตรวจสอบผลลัพธ์ โดยการอ่านข้อเสนอแนะประมาณ 30% ของจำนวนทั้งหมด เพื่อตรวจสอบว่าผลลัพธ์ที่ได้มีความถูกต้องจริง สาเหตุที่ผู้วิจัยและผู้เชี่ยวชาญทำการตรวจสอบข้อเสนอแนะใช้วิธีการตรวจสอบด้วยตัวเอง (Manual) เพราะในปัจจุบันยังไม่มีเครื่องมือสำหรับการตรวจสอบข้อเสนอแนะโดยอัตโนมัติ

2) การวิเคราะห์ผลการวิจัยนี้เป็นการวิเคราะห์เพื่อที่จะตอบคำถามวิจัยว่า นักพัฒนาให้ความสำคัญกับความสามสามารถในการบำรุงรักษามากน้อยแค่ไหน โดยตรวจสอบผ่านคุณลักษณะต่าง ๆ ที่เกี่ยวข้องกับการบำรุงรักษาซอฟต์แวร์ที่เกิดขึ้นในช่วงระยะเวลา 5 ปีที่ผ่านมาของโครงการ Eclipse และโครงการ Qt เท่านั้น เนื่องจากมีปัจจัยเรื่องระยะเวลาเข้ามาเกี่ยวข้อง จึงอาจทำให้มีผลต่อประสบการณ์ของนักพัฒนาซอฟต์แวร์ ความต้องการของผู้ใช้ที่เพิ่มมากขึ้น รวมไปถึงสภาพแวดล้อมต่าง ๆ ที่เปลี่ยนแปลงไป

5.3.3 ความถูกต้องภายนอก (External Validity)

ภัยคุมคามต่อความถูกต้องภายนอก (External Validity) เป็นการพิจารณาว่าผลที่ได้จากการทำวิจัยสามารถนำไปประยุกต์ใช้กับกลุ่มตัวอย่างอื่น ๆ หรือกลุ่มตัวอย่างที่มีขนาดใหญ่กว่าได้หรือไม่ รวมไปถึงสถานการณ์หรือเหตุการณ์อื่น ๆ ที่มีลักษณะหรือเงื่อนไขเดียวกันกับการวิจัยในครั้งนี้ นอกจากนี้ยังเป็นการพิจารณาว่าวิธีการดำเนินงาน วิธีการวัด และเครื่องมือต่าง ๆ ไปใช้กับกลุ่มตัวอย่างอื่น ๆ ได้มากน้อยแค่ไหนอีกด้วย สำหรับงานวิจัยในครั้งนี้ ผู้วิจัยได้ทำการระบุถึงภัยคุกคามที่อาจส่งผลต่อความถูกต้องภายนอกไว้ดังนี้

1) ในงานวิจัยนี้ได้ทำการศึกษาข้อเสนอแนะของโครงการ Eclipse และโครงการ Qt เป็นระยะเวลา 5 ปี นั่นคือ ช่วงปี ค.ศ. 2012 – 2016 ซึ่งผู้วิจัยไม่ได้ทำการนำเสนอข้อมูลเกี่ยวกับข้อเสนอแนะของปี ค.ศ. 2017 จึงอาจทำให้แนวโน้มของการให้ความสนใจของนักพัฒนาต่อการแก้ไขปรับปรุงซอร์สโค้ดเกี่ยวกับการบำรุงรักษาในแต่ละประเภทมีความคลาดเคลื่อนเล็กน้อย

2) ผลลัพธ์ของการศึกษานี้เป็นเพียงหลักฐานเชิงประจักษ์เบื้องต้นของโครงการโอเพนซอร์สเพียงแค่ 2 โครงการเท่านั้น จึงอาจไม่ครอบคลุมสำหรับโครงการโอเพนซอร์สทุกโครงการ เนื่องจากโครงการโอเพนซอร์สแต่ละโครงการมีโครงสร้างการทำงานภายใน กระบวนการตรวจทานโค้ด และกระบวนการพัฒนาซอฟต์แวร์ที่แตกต่างกัน อย่างไรก็ตามผู้วิจัยเชื่อว่า ผลการศึกษานี้เป็นแนวทางเบื้องต้นสำหรับการศึกษาอื่น ๆ ที่เกี่ยวข้องกับคุณภาพของโครงการโอเพนซอร์สที่มีความคล้ายคลึงกัน

5.4 งานวิจัยในอนาคต

เนื่องจากผลลัพธ์ของงานวิจัยนี้เป็นเพียงการสรุปผลขั้นพื้นฐาน แต่ข้อมูลที่มีอยู่ก็เพียงพอสำหรับการนำมาเป็นแนวทางในการพัฒนาหรือปรับปรุงคุณภาพของซอฟต์แวร์โอเพนซอร์สให้ดียิ่งขึ้น รวมไปถึงช่วยให้นักพัฒนาซอฟต์แวร์ได้ตระหนักถึงความสำคัญของการปรับปรุงแก้ไขโค้ดทางด้านการบำรุงรักษา อีกทั้งยังช่วยให้นักวิจัยที่ให้ความสนใจเกี่ยวกับการบำรุงรักษาซอฟต์แวร์สามารถนำงานวิจัยนี้ไปทำการศึกษาเพิ่มเติมได้ ดังนั้นผู้วิจัยจึงนำเสนองานวิจัยหรือคำถามที่นักวิจัยทางด้านวิศวกรรมซอฟต์แวร์สามารถทำการศึกษาหรือพัฒนาต่อยอดในอนาคต เช่น การพัฒนาเครื่องมือหรือส่วนเสริมโปรแกรม (Plug-in) ในระบบเกอริตที่สามารถตรวจสอบว่า ข้อเสนอแนะที่เกี่ยวข้องกับการบำรุงรักษาที่ได้รับจากผู้ตรวจทานโค้ดนั้นเป็นคำแนะนำที่เป็นประโยชน์และสามารถนำไปปรับใช้ได้จริง พร้อมทั้งแสดงข้อมูลประเภทของข้อเสนอแนะเกี่ยวกับการบำรุงรักษาบนบรรทัดของโค้ดที่ควรแก้ไข เช่น ความสามารถในการอ่าน (Readability) ความสามารถในการทดสอบ (Testability) เพื่อให้นักพัฒนาซอฟต์แวร์สามารถตระหนักได้ว่า ควรทำการแก้ไขโค้ดให้ครอบคลุมกับคำนิยายของคุณลักษณะนั้น ๆ นอกจากนี้อาจทำการพัฒนาเครื่องมือหรือกระบวนการที่สามารถตรวจสอบได้ว่า นักพัฒนาในโครงการโอเพนซอร์สได้แก้ไขโค้ดตามคำแนะนำของผู้ตรวจทานโค้ดภายใต้กระบวนการตรวจทานโค้ดของระบบเกอริต ซึ่งหากมีการพัฒนาชุดคำสั่งหรือโปรแกรมเพื่อตรวจสอบการแก้ไขโค้ดในระบบเกอริตแบบอัตโนมัติและมีการแจ้งเตือนไปยังผู้ตรวจทานโค้ดที่ให้คำแนะนำในการแก้ไขปรับปรุงโค้ด โดยอำนวยความสะดวกแก่ชุมชนนักพัฒนาซอฟต์แวร์ในการตรวจสอบสิ่งที่ได้ทำการแก้ไขไปแล้ว อีกทั้งยังช่วยให้ผู้ตรวจทานโค้ดสามารถรับทราบและให้คำแนะนำในส่วนอื่น ๆ เพิ่มเติมได้รวดเร็วยิ่งขึ้น

เอกสารอ้างอิง

สมคิด ทุ่นใจ. (2559). “Open Source Software : อิสรภาพแห่งการสร้างคุณค่าและพัฒนางานวิจัย.”, *วารสารวิชาการมหาวิทยาลัยราชภัฏอุตรดิตถ์*, 11(2), 1-12.

Aberdour, M. (2007). “Achieving quality in open source software.” *IEEE software*, 24(1), 58–64.

Abreu, F. B. e., and Melo, W. (1996). “Evaluating the Impact of Object-Oriented Design on Software Quality.”, *Proceedings of The 3rd International Symposium on Software Metrics: From Measurement to Empirical Results (METRICS),* Berlin, Germany: 25-26 March 1996.

Anjali, S. K. S. (2015). “Bug Triaging: Profile Oriented Developer Recommendation.” *International Journal of Innovative Research in Advanced Engineering*, 2(1), 36–42.

Asundi, J., and Jayant, R. (2007). “Patch review processes in open source software development communities: A comparative case study.”, *Proceedings of The 40th Annual Hawaii International Conference on System Sciences (HICSS),* Waikoloa, HI, USA: 3-6 January, 2007.

Bacchelli, A., and Bird, C. (2013). “Expectations, Outcomes, and Challenges of Modern Code Review.”, *Proceedings of The 13th International Conference on Software Engineering (ICSE),* San Francisco, CA, USA: 18-26 May, 2013.

Baghela, V. S., and S.P.Tripathi, D. (2012). “Text Mining Approaches To Extract Interesting Association Rules from Text Documents.” *International Journal of Computer Science Issues*, 9(3), 545–552.

Baker Jr., R. A. (1997). “Code Reviews Enhance Software Quality.”, *Proceedings of The 19th International Conference on Software Engineering (ICSE),* Boston, MA, USA: 17-23 May, 1997.

Bakar, N. S., A. A., and Arsat, N. (2014). “Investigating the factors that influence the quality of open source systems.”, *Proceedings of The 5th International Conference on Information and Communication Technology for The Muslim World (ICT4M),* Kuching, Malaysia: 17-18 November, 2014.

Balachandran, V. (2013). “Reducing Human Effort and Improving Quality in Peer Code Reviews Using Automatic Static Analysis and Reviewer Recommendation.”, *Proceedings of The 35th International Conference on Software Engineering (ICSE),* San Francisco, CA, USA: 18-26 May, 2013.

Baysal, O., Kononenko, O., Holmes, R., and Godfrey, M. W. (2013). “The influence of non-technical factors on code review.”, *Proceedings of The 20th Working Conference on Reverse Engineering (WCRE),* Koblenz, Germany: 14-17 October, 2013.

Bernhart, M., Mauczka, A., and Grechenig, T. (2010). “Adopting code reviews for agile software development.”, *Proceedings of The Agile Conference (AGILE),* Nashville, TN, USA: 9-13 August, 2010.

Bird, C., Pattison, D., D’Souza, R., Filkov, V., and Devanbu, P. (2008). “Latent Social Structure in Open Source Projects.”, *Proceedings of The 16th International Symposium on Foundations of Software Engineering (SIGSOFT),* Atlanta, Georgia: 9-14 November, 2008.

Blei, D. M., Edu, B. B., Ng, A. Y., Edu, A. S., Jordan, M. I., and Edu, J. B. (2003). “Latent Dirichlet Allocation.” *Journal of Machine Learning Research*, 3, 993–1022.

Bosu, A. (2013). “Modeling Modern Code Review Practices in Open Source Software Development Organizations.”, *Proceedings of The 11th International Doctoral Symposium on Empirical Software Engineering (IDoESE),* Baltimore, Maryland USA: 9 October, 2013.

Bosu, A., and Carver, J. C. (2012). “Peer code review in open source communitiesusing reviewboard.”, *Proceedings of The 4th annual workshop on Evaluation and usability of programming languages and tools (PLATEAU)*, Tucson, Arizona, USA: 21-21 October, 2012.

Bosu, A., and Carver, J. C. (2013). “Impact of peer code review on peer impression formation: A survey.” *Proceedings of The 7th International Symposium on Empirical Software Engineering and Measurement (ESEM),* Baltimore, MD, USA: 10-11 October, 2013.

Bosu, A., and Carver, J. C. (2014). "Impact of Developer Reputation on Code Review Outcomes in OSS Projects: An Empirical Investigation.", *Proceedings of The 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM),* Torino, Italy: 18-19 September, 2014.

Bosu, A., Carver, J. C., Hafiz, M., Hilley, P., and Janni, D. (2014). "Identifying the Characteristics of Vulnerable Code Changes: An Empirical Study.", *Proceedings of The 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*, Hong Kong, China: 16-21 November, 2014.

Bosu, A., Greiler, M., and Bird, C. (2015). "Characteristics of Useful Code Reviews: An Empirical Study at Microsoft.", *Proceedings of The 12th Working Conference on Mining Software Repositories (MSRX),* Florence, Italy: 16-24 May, 2015.

Broy, M. and Denert, E. (2002). *A history of software inspections. Software pioneers: contributions to software engineering*. Springer-Verlag, Inc., Berlin, Heidelberg.

Caglayan, B., Bener, A., and Koch, S. (2009). "Merits of Using Repository Metrics in Defect Prediction for Open Source Projects." *Proceedings of The 2nd International Workshop on Emerging Trends in Free/Libre/Open Source Software research and development (FLOSS),* Vancouver, BC, Canada: 18 May, 2009.

Capiluppi, A., Lago, P., and Morisio, M. (2003). "Characteristics of open source projects.", *Proceedings of The 7th European Conference on Software Maintenance and Reengineering (CSMR),* Benevento, Italy, Italy: 28-28 March 2003.

Chen, T.-H., Thomas, S. W., and Hassan, A. E. (2016). "A survey on the use of topic models when mining software repositories." *Empirical Software Engineering*, 21(5), 1843-1919.

Ciolkowski, M., Laitenberger, O., Rombach, D., Shull, F., and Perry, D. (2002). "Software Inspections, Reviews & Walkthroughs.", *Proceedings of The 24th International Conference on Software Engineering (ICSE),* Orlando, Florida: 19-25 May, 2002.

Coelho, J., and Valente, M. T. (2017). "Why Modern Open Source Projects Fail.", *Proceedings of The 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE),* Paderborn, Germany: 4-8 September, 2017.

Comino, S., and Manenti, F. M., and Parisi, M. L. (2005). “From planning to mature: on the determinants of open source take-off.” (Online) Available on https://perso .univ-rennes1.fr/eric.darmon/floss/nice/papers/COMINO_MANENTI_PARISI.pdf (5 November 2016).

Counsell, S., Liu, X., Eldh, S., Tonelli, R., Marchesi, M., Concas, G., and Murgia, A. (2015). “Re-visiting the 'Maintainability Index' Metric from an Object-Oriented Perspective.”, *Proceedings of The 41st Euromicro Conference on Software Engineering and Advanced Applications (SEAA),* Funchal, Portugal: 26-28 August, 2015.

Czerwonka, J., and Greiler, M. (2015). “Code Reviews Do Not Find Bugs. How the Current Code Review Best Practice Slows Us Down.”, *Proceedings of The 37th International Conference on Software Engineering (ICSE),* Florence, Italy: 16-24 May, 2015.

Deissenboeck, F., Wagner, S., Pizka, M., Teuchert, S., and Girard, J.-F. (2007). “An Activity-Based Quality Model for Maintainability.”, *Proceedings of The 23rd IEEE International Conference on Software Maintenance (ICSM),* Paris, France 2-5 October, 2007.

Ebert, C., and Cain, J. (2016). “Cyclomatic Complexity.” *IEEE Software*, 33(6), 27–29.

Fagan, M. E. (1976). “Design and Code Inspections to Reduce Errors in Program Development.” *IBM Systems Journal*, 15(3), 182–211.

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). “Knowledge Discovery and Data Mining: Towards a Unifying Framework.”, *Proceedings of The 2nd International Conference on Knowledge Discovery and Data Mining (KDD)*, Portland, Oregon: 2-4 August, 1996.

Gamalielsson, J., and Lundell, B. (2012). “Long-term sustainability of open source software communities beyond a fork: A case study of LibreOffice.” *IFIP Advances in Information and Communication Technology*, 378, 29–47.

Ghosh, S., Dubey, S., and Rana, A. (2011). “Comparative Study of the Factors that Affect Maintainability.” *International Journal on Computer Science and Engineering*, 3(12), 3763–3769.

Glass, R. L. (2001). “Frequently Forgotten Fundamental Facts About Software Engineering.” *IEEE Software*, 18(3), 111–112.

Glass, R. L. (2002). *Software Engineering: Facts and Fallacies*, Addison-Wesley Longman Publishing Co., Inc., USA.

Hamasaki, K., Kula, R. G., Yoshida, N., Cruz, A. E. C., Fujiwara, K., and Iida, H. (2013). “Who Does What During a Code Review? Datasets of OSS Peer Review Repositories.”, *Proceedings of The 10th Working Conference on Mining Software Repositories (MSR)*, San Francisco, CA, USA: 18-19 May, 2013.

Hammond, J., Santinelli, P., Billings, J. J., and Ledingham, B. (2016). “The North Bridge & Black Duck Future of Open Source Study marks the 10th Anniversary of this survey.” (Online) Available on http://www.northbridge.com/2016-future-open-source-survey-results (26 September 2016).

Harrison, W., Magel, K., Kluczny, R., and DeKock, A. (1982). “Applying Software Complexity Metrics to Program Maintenance.” *Computer*, 15(9), 65–79.

Hinkle, D. E., Wiersma, W., and Jurs, S. G. (2003). *Applied Statistics for the Behavioral Sciences (5th ed.).* Boston: Houghton Mifflin.

Iso Iec, I. I. (2011). *Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models.* Iso, BS ISO/IEC, Inc., UK.

Kádár, I., Heged\Hus, P., Ferenc, R., and Gyimóthy, T. (2016). “A Manually Validated Code Refactoring Dataset and Its Assessment Regarding Software Maintainability.”, *Proceedings of The 12th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE),* Ciudad Real, Spain: 9 September, 2016.

Khan, S., and Khan, P. R. (2012). “Analyzability Quantification Model of Object Oriented Design.” *Procedia Technology*, 4, 536–542.

Kononenko, O., Baysal, O., Guerrouj, L., Cao, Y., and Godfrey, M. W. (2015). “Investigating code review quality: Do people and participation matter?”, *Proceedings of The 31st International Conference on Software Maintenance and Evolution (ICSME),* Bremen, Germany: 29 September - 1 October, 2015.

Lee, G. K., and Cole, R. E. (2003). “From a Firm-Based to a Community-Based Model of Knowledge Creation: The Case of the Linux Kernel Development.” *Organization Science*, 14(6), 633–649.

Mcintosh, S., Kamei, Y., Adams, B., and Hassan, A. E. (2014). “The Impact of Code Review Coverage and Code Review Participation on Software Quality Categories and Subject Descriptors.”, *Proceedings of The 11th Working Conference on Mining Software Repositorie (MSR)*, Hyderabad, India: 31 May 31 - 1 June, 2014.

Mockus, A., Fielding, R. T., and Herbsleb, J. D. (2002). “Two case studies of open source software development: Apache and Mozilla.” *ACM Transactions on Software Engineering and Methodology (TOSEM),* 11(3), 309–346.

Mooney, R. J., and Nahm, U. Y. (2003). “Text mining with Information extraction.”, *Proceedings of The 4th International MIDP Colloquiu,* Bloemfontein, South Africa: 22-23 September, 2003.

Mukadam, M., Bird, C., and Rigby, P. C. (2013). “Gerrit Software Code Review Data from Android.”, *Proceedings of The 10th Working Conference on Mining Software Repositories (MSR),* San Francisco, CA, USA: 18-19 May, 2013.

Nasir, Z., and Abbasi, A. (2010). “A framework for software maintenance and support phase.”, *Proceedings of The 2nd International Conference on Information and Emerging Technologies (ICIET),* Karachi, Pakistan, 14-16 June, 2010.

Norick, B., Krohn, J., Howard, E., Welna, B., and Izurieta, C. (2010). “Effects of the Number of Developers on Code Quality in Open Source Software: A Case Study.”, *Proceedings of The 10th ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Bolzano-Bozen, Italy: 16-17 September, 2010.

Oh, J.-S. and Choi, H.-J. (2005). “A Reflective Practice of Automated and Manual Code Reviews for a Studio Project.”, *Proceedings of The 4th Annual ACIS International Conference on Computer and Information Science (ICIS),* Jeju Island, South Korea, South Korea: 14-16 July, 2005.

O’Reilly, T. (1999). “Lesson from Open- Source Software Development.” Communications of the ACM, 42(4), 32-37.

Paulson, J. W., Succi, G., and Eberlein, A. (2004). “An empirical study of open-source and closed-source software products.” *IEEE Transactions on Software Engineering*, 30(4), 246–256.

Peercy, D. E. (1981). “A Software Maintainability Evaluation Methodology.” *IEEE Transactions on Software Engineering*, 7(4), 343-351.

Rizvi, S. W. A. and Khan, R. A. (2010). “Maintainability Estimation Model for Object-Oriented Software in Design Phase (MEMOOD).” *Journal of Computing*, 2(4), 26-32.

Porter, A., Siy, H., Mockus, A., and Votta, L. (1998). “Understanding the Sources of Variation in Software Inspections.” *ACM Transactions on Software Engineering and Methodology,* 7(1), 41–79.

Raymond, E. S. (1999). *The Cathedral and the Bazaar,* O'Reilly Media, Inc., CA.

Rigby, P. C. (2011). “Understanding Open Source Software Peer Review: Review Processes, Parameters and Statistical Models, and Underlying Behaviours and Mechanisms.”, Doctoral Dissertation, University of Victoria.

Rigby, P. C., and Bird, C. (2013). “Convergent contemporary software peer review practices.”, *Proceedings of The 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE),* Saint Petersburg, Russia: 18-26 August, 2013.

Rigby, P. C., and German, D. M. (2006). “A preliminary examination of code review processes in open source projects.” *Technical Report DCS-305-IR,* University of Victoria, Canada.

Rigby, P. C., German, D. M., and Storey, M.-A. (2008). “Open source software peer review practices: a case study of the apache server.”, *Proceedings of The 30th international conference on Software engineering (ICSE)*, Leipzig, Germany: 10-18 May, 2008.

Sauer, C., Jeffery, D. R., Land, L., and Yetton, P. (2000). “The Effectiveness of Software Development Technical Reviews: A Behaviorally Motivated Program of Research.” *IEEE Transactions on Software Engineering,* 26(1), 1–14.

Seref, B., and Tanriover, O. (2016). “Software Code Maintainability : A Literature Review.” *International Journal of Software Engineering & Applications*, 7(3), 69–87.

Scacchi, W. (2002). “Understanding the requirements for developing open source software systems.” *IEE Proceedings - Software*, 149(1), 24-39.

Schmidt, D. C., and Porter, A. (2001). “Leveraging open-source communities to improve the quality and performance of open-source software.”, *Proceedings of The 1st Workshop on Open Source Software Engineering (ICSE),* Toronto, Canada: 15 May, 2001.

Stamelos, I., Angelis, L., Oikonomou, A., and Bleris, L. (2002). “Code quality analysis in open source software development.” *Information Systems Journal*, 12(1), 43-60.

Stroggylos, K., and Spinellis, D. (2007). “Refactoring--Does It Improve Software Quality?”, *Proceedings of The 5th International Workshop on Software Quality (WoSQ),* Minneapolis, MN, USA: 20-26 May, 2007.

Tao, Y., Han, D., and Kim, S. (2014). “Writing acceptable patches: An empirical study of open source project patches.”, *Proceedings of The 30th International Conference on Software Maintenance and Evolution (ICSME),* Victoria, BC, Canada: 29 September - 3 October, 2014.

The Economist. (2006). “Open-source business – Open, but not as usual.” (Online) Available on http://www.economist.com/displaystory.cfm?story_id=E1_VGNQ JQQ (5 November 2016).

Thongtanunam, P., Tantithamthavorn, C., Kula, R. G., Yoshida, N., Iida, H., and Matsumoto, K. I. (2015). “Who should review my code? A file location-based code-reviewer recommendation approach for Modern Code Review.”, *Proceedings of The 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER),* Montreal, QC, Canada: 2-6 March 2015.

Tiwari, V. (2010). “Some Observations on Open Source Software Development on Software Engineering Perspectives.” *International Journal of Computer Science & Information Technology*, 2(6), 113–125.

Tiwari, V., and Pandey, R. (2012). “Open Source Software and Reliability Metrics.” *International Journal of Advanced Research in Computer and Communication Engineering*, 1, 808–815.

Tymchuk, Y., Mocci, A. and Lanza, M. (2015). "Code review: Veni, ViDI, vici.", *Proceedings of The 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER),* Montreal, QC, Canada: 2-6 March, 2015.

Wagey, C. B., Hendradjaya, B., and Mardiyanto, M. (2015). "A proposal of software maintainability model using code smell measurement.", *Proceedings of The 2nd International Conference on Data and Software Engineering (ICoDSE),* Yogyakarta, Indonesia: 25-26 November, 2015.

Wahyudin, D., Schatten, A., Winkler, D., and Biffl, S. (2007). "Aspects of Software Quality Assurance in Open Source Software Projects: Two Case Studies from Apache Project.", *Proceedings of The 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO)*, Lubeck, Germany: 28-31 August, 2007.

Walden, J., Doyle, M., Welch, G. A., and Whelan, M. (2009). "Security of Open Source Web Applications.", *Proceedings of The 3rd International Symposium on Empirical Software Engineering and Measurement (ESEM),* Lake Buena Vista, Florida USA: 15-16 October, 2009.

Walia, G., and Carver, J. (2013). "Using error information to improve software quality." *Software Reliability Engineering*, 18(4), 4799.

Yamashita, K., Huang, C., Nagappan, M., Kamei, Y., Mockus, A., Hassan, A. E. and Ubayashi, N. (2016). "Thresholds for Size and Complexity Metrics: A Case Study from the Perspective of Defect Density.", *Proceedings of The 2nd IEEE International Conference on Software Quality, Reliability and Security (QRS),* Vienna, Austria: 1-3 August, 2016.

Ye, Y., and Kishida, K. (2003). "Toward an Understanding of the Motivation Open Source Software Developers.", *Proceedings of The 25th International Conference on Software Engineering (ICSE)*, Portland, Oregon: 3-10 May, 2003.

Zanjani, M. B., Kagdi, H., and Bird, C. (2016). "Automatically Recommending Peer Reviewers in Modern Code Review." *IEEE Transactions on Software Engineering*, 42(6), 530–543.

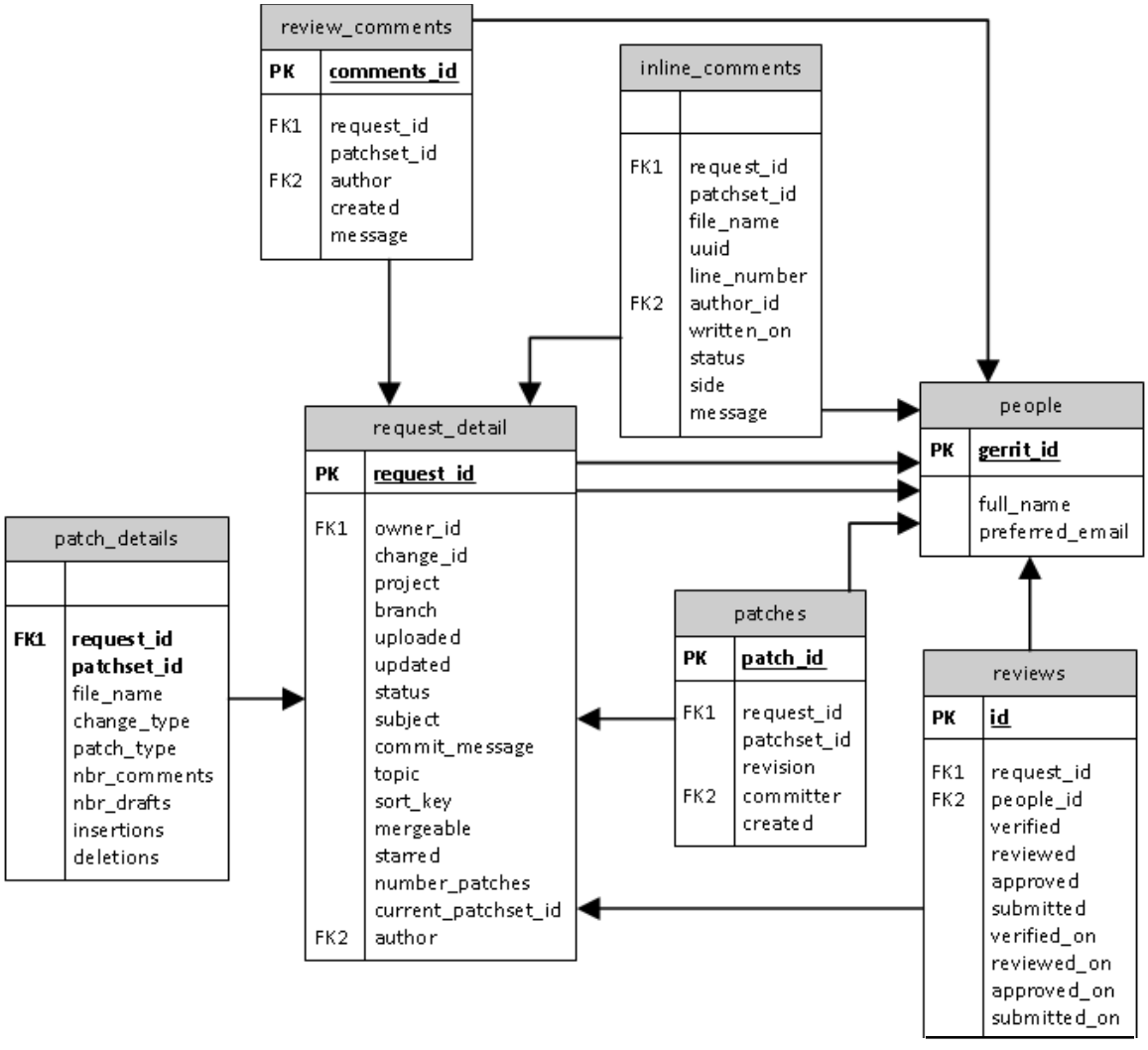
Zhang, J., Wang, X., Hao, D., Xie, B., Zhang, L., and Mei, H. (2015). "A survey on bug-report analysis." *Science China Information Sciences*, 58, 1–24.

Zhou, Y., and Davis, J. (2005). “Open Source Software Reliability Model: An Empirical Approach.” *ACM SIGSOFT Software Engineering Notes*, 30(4), 1–6.

ภาพผนวก

ภาพผนวก ก

ลักษณะโครงสร้างการเก็บข้อมูลในระบบเกอริต



รูปที่ ก.1 โครงสร้างการเก็บข้อมูลของระบบเกอริต[30]

[30] http://amiangshu.com/sna-code-review/index.html

ตารางที่ ก.1 ตารางเก็บข้อมูลของผู้เข้าใช้งาน

People		
ชื่อคอลัมน์	คำอธิบาย	คีย์
Gerrit_id	ข้อมูลผู้ใช้งาน	PK
Full_name	ชื่อผู้ใช้งาน	
Preferred_email	อีเมล์ที่ใช้ในการติดต่อ	

ตารางที่ ก.2 ตารางเก็บข้อมูลของการส่งคำร้องขอการตรวจทานโค้ด

Reguest_detail		
ชื่อคอลัมน์	คำอธิบาย	คีย์
Request_id	หมายเลขของการส่งคำร้องขอ	PK
Owner_id	หมายเลขของผู้ส่งคำร้องขอ	FK1
Change_id	หมายเลขของการเปลี่ยนแปลงโค้ด	
Project	ชื่อโครงการ	
Branch	เส้นทางหรือตำแหน่งของการจัดเก็บข้อมูล	
Uploaded	วันและเวลาที่มีการอัปโหลดไฟล์	
Status	สถานะของคำร้องขอ	
Subject	ประเภทของการส่งคำร้องขอ	
Commit_message	ข้อความที่แจ้งรายละเอียดของการอัปโหลดไฟล์	
Topic	เรื่องที่ต้องการให้ทำการตรวจสอบ	
Sort_key	ลำดับในการแก้ไข	
Mergeable	ชุดโปรแกรมแก้ไขที่ถูกรวมกับโปรแกรมต้นฉบับล่าสุด	
Starred	เครื่องหมายที่ถูกแสดง	
Number_patches	จำนวนของชุดโปรแกรม	
Current_patchset_id	หมายเลขปัจจุบันของชุดโปรแกรมที่ถูกแก้ไข	
Author	ชื่อผู้ส่งคำร้องขอ	FK2

ตารางที่ ก.3 ตารางเก็บข้อมูลของชุดโปรแกรม

Patches		
ชื่อคอลัมน์	คำอธิบาย	คีย์
Patch_id	หมายเลขของชุดโปรแกรม	PK
Request_id	หมายเลขของการส่งคำร้องขอ	FK1
Patchset_id	หมายเลขของชุดโปรแกรมที่ถูกแก้ไข	
Revision	หมายเลขของการแก้ไข	
Committer	ชื่อของผู้ที่ต้องการแก้ไขชุดโปรแกรม	FK2
Creates	วันและเวลาในการแก้ไข	

ตารางที่ ก.4 ตารางเก็บข้อมูลของโปรแกรมที่ถูกแก้ไข

Patch_details		
ชื่อคอลัมน์	คำอธิบาย	คีย์
Request_id	หมายเลขของการส่งคำร้องขอ	FK1
Patchset_id	หมายเลขของชุดโปรแกรมที่ถูกแก้ไข	
File_name	ชื่อไฟล์ของชุดโปรแกรมที่ทำการเปลี่ยนแปลง	
Change_type	ประเภทของการเปลี่ยนแปลง	
Patch_type	ประเภทของชุดโปรแกรมที่ถูกแก้ไข	
Nbr_comments	จำนวนของข้อเสนอแนะ	
Nbr_drafts	จำนวนของแบบร่างชุดโปรแกรม	
Insertions	สิ่งที่ถูกเพิ่มเข้ามา	
Deletions	สิ่งที่ถูกลบออกไป	

ตารางที่ ก.5 ตารางเก็บข้อมูลของการตรวจทานโค้ด

Reviews		
ชื่อคอลัมน์	คำอธิบาย	คีย์
Id	หมายเลขของลำดับการตรวจทานโค้ด	PK
Request_id	หมายเลขของการส่งคำร้องขอ	FK1
People_id	หมายเลขของผู้ใช้งาน	FK2
Verified	สถานะของการยืนยันในการตรวจทานโค้ด	
Reviewed	คะแนนของการตรวจทานโค้ด	
Approved	สถานะของการอนุมัติการเปลี่ยนแปลง	
Submitted	สถานะของการยอมรับให้ชุดโปรแกรมรวมกับต้นฉบับ	
Verified_on	วันและเวลาของการยืนยันในการตรวจทานโค้ด	
Reviewed_on	วันและเวลาที่ทำการตรวจทานโค้ด	
Approved_on	วันและเวลาที่อนุมัติการเปลี่ยนแปลง	
Submitted_on	วันและเวลาที่ยอมรับการรวมชุดโปรแกรม	

ตารางที่ ก.6 ตารางเก็บข้อมูลของข้อเสนอแนะ

Review_comments		
ชื่อคอลัมน์	คำอธิบาย	คีย์
Comments_id	หมายเลขของข้อเสนอแนะ	PK
Request_id	หมายเลขของการส่งคำร้องขอ	FK1
Patchset_id	หมายเลขของชุดโปรแกรมที่ถูกแก้ไข	
Author	ชื่อผู้ตรวจทานโค้ด	FK2
Created	วันและเวลาของการให้ข้อเสนอแนะ	
Message	ข้อความของข้อเสนอแนะ	

ตารางที่ ก.7 ตารางเก็บข้อมูลรายละเอียดของข้อเสนอแนะ

Inline_comments		
ชื่อคอลัมน์	คำอธิบาย	คีย์
Request_id	หมายเลขของการส่งคำร้องขอ	FK1
Patchset_id	หมายเลขของชุดโปรแกรมที่ถูกแก้ไข	
File_name	ชื่อไฟล์ของชุดโปรแกรม	
Uuid	หมายเลขประจำตัวของข้อมูล	
Line_number	หมายเลขของบรรทัดในซอร์สโค้ดที่มีการให้ข้อเสนอแนะ	
Author_id	หมายเลขของผู้ตรวจทานโค้ด	FK2
Written_on	วันและเวลาที่ให้คำแนะนำบนบรรทัดของซอร์สโค้ด	
Status	สถานะของข้อเสนอแนะ	
Side	ตำแหน่งที่มีข้อเสนอแนะ	
Message	ข้อความของข้อเสนอแนะ	

ภาคผนวก ข

โครงสร้างฐานข้อมูล MySQL ที่ใช้ในการจัดเก็บข้อมูล

ตารางที่ ข.1 โครงสร้างของฐานข้อมูลที่ใช้ในงานวิจัย

Comment_detail		
ชื่อคอลัมน์	คำอธิบาย	คีย์
Id	หมายเลขของชุดโปรแกรมที่ทำการเปลี่ยนแปลง	PK
Link_id	URL ของชุดโปรแกรมที่ทำการเปลี่ยนแปลง	
Patchset_number	หมายเลขของชุดโปรแกรมที่ถูกแก้ไข	
Uploader	ชื่อของผู้อัปโหลดชุดโปรแกรม	
Author	ชื่อของผู้ร้องขอการตรวจทานโค้ด	
Reviewer	ชื่อของผู้ตรวจทานโค้ด	
Created_on	วันและเวลาที่ผู้ตรวจทานโค้ดให้คำแนะนำ	
File	ชุดโปรแกรมที่มีการร้องขอให้ตรวจทาน	
Line	หมายเลขบรรทัดของซอร์สโค้ดที่ให้คำแนะนำ	
Message	ข้อความของข้อเสนอแนะ	
SizeInsertion	จำนวนข้อเสนอแนะทั้งหมดของชุดโปรแกรมที่มีการร้องขอให้ตรวจทาน	
SizeDeletions	จำนวนข้อเสนอแนะที่ถูกลบภายในไฟล์ที่มีการร้องขอให้ตรวจทาน	
Kind	ประเภทของข้อเสนอแนะ	

ประวัติผู้เขียน

ชื่อ สกุล นางสาวธัญญรัตน์ กิจพาณิชย์

รหัสประจำตัวนักศึกษา 5930223002

วุฒิการศึกษา

วุฒิ	ชื่อสถาบัน	ปีที่สำเร็จการศึกษา
วิทยาศาสตรบัณฑิต (วิศวกรรมซอฟต์แวร์)	มหาวิทยาลัยสงขลานครินทร์	2559

ทุนการศึกษา

กองทุนอุดหนุนวิจัยจากวิทยาลัยการคอมพิวเตอร์

การตีพิมพ์เผยแพร่ผลงาน

Nanthaamornphong, A., Leatongkam, A., Kitpanich, T., and Thongnuan, P. (2015). "Bytecode-based class dependency extraction tool: Bytecode-CDET.", *Proceedings of The 7th International Conference on Information Technology and Electrical Engineering (ICITEE)*, Chiang Mai, Thailand: 29-30 October, 2015.

ธัญญรัตน์ กิจพาณิชย์ และ อชีส นันทอมรพงศ์. (2560). “หลักฐานเชิงประจักษ์เกี่ยวกับความสามารถในการบำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์ส”, *งานประชุมวิชาการระดับชาติด้านคอมพิวเตอร์และเทคโนโลยีสารสนเทศ ครั้งที่ 13*, โรงแรมอโนมา แกรนด์ กรุงเทพมหานคร: 6-7 กรกฎาคม 2560.

Nanthaamornphong, A., and Kitpanich, T. (2017). “The Study of Code Reviews based on Software Maintainability in Open Source Projects.” *Journal of Telecommunication, Electronic and Computer Engineering*, 9(3-4), 123-129.