อิทธิพลของการทำรีแฟคทอริงต่อการตรวจทานโค้ดในโครงการซอฟต์แวร์โอเพนซอร์ส

Refactoring Influences in the Code Review for Open Source Software Projects

อปัตตา ชัยสุตานนท์

Apatta Chaisutanon

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

2560

อิทธิพลของการทำรีแฟคทอริงต่อการตรวจทานโค้ดในโครงการซอฟต์แวร์โอเพนซอร์ส

Refactoring Influences in the Code Review for Open Source Software Projects

อปัตตา ชัยสุตานนท์

Apatta Chaisutanon

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
2560

ชื่อวิทยานิพนธ์ อิทธิพลของการทำรีแฟคทอริงต่อการตรวจทานโค้ดในโครงการซอฟต์แวร์โอเพนซอร์ส

ผู้เขียน นางสาวอปัตตา ชัยสุตานนท์

สาขาวิชา เทคโนโลยีสารสนเทศ

อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

.....

(ผู้ช่วยศาสตราจารย์ ดร.อซีส นันทอมรพงศ์)

คณะกรรมการสอบ

.....ประธานกรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.รัตนา เวทย์ประสิทธิ์)

.....กรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.ทรงศรี ตั้งศรีไพโรจน์)

.....กรรมการ

(ผู้ช่วยศาสตราจารย์ ดร.อซีส นันทอมรพงศ์)

บัณฑิตวิทยาลัย มหาวิทยาลัยสงขลานครินทร์ อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ

.....

(รองศาสตราจารย์ ดร.ธีระพล ศรีชนะ)

คณบดีบัณฑิตวิทยาลัย

ขอรับรองว่า ผลงานวิจัยนี้มาจากการศึกษาวิจัยของนักศึกษาเอง และได้แสดงความขอบคุณบุคคลที่มีส่วนช่วยเหลือแล้ว

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

ลงชื่อ.....
(อปัตตา ชัยสุตานนท์)
นักศึกษา

ข้าพเจ้าขอรับรองว่า ผลงานวิจัยนี้ไม่เคยเป็นส่วนหนึ่งในการอนุมัติปริญญาในระดับใดมาก่อน และไม่ได้ถูกใช้ในการยื่นขออนุมัติปริญญาในขณะนี้

ลงชื่อ.....
(อปัตตา ชัยสุตานนท์)
นักศึกษา

ชื่อวิทยานิพนธ์ อิทธิพลของการทำรีแฟคทอริงต่อการตรวจทานโค้ดในโครงการซอฟต์แวร์โอเพนซอร์ส

ผู้เขียน นางสาวอปัตตา ชัยสุตานนท์
เทคโนโลยีสารสนเทศ

ปีการศึกษา 2559

บทคัดย่อ

ในปัจจุบันซอฟต์แวร์โอเพนซอร์สได้กลายมาเป็นตัวเลือกที่สำคัญในการนำซอฟต์แวร์ไปใช้งาน และมีแนวโน้มที่จะถูกนำไปใช้งานสูงขึ้น ซอฟต์แวร์โอเพนซอร์สค่อนข้างได้รับความน่าเชื่อถือในแง่ของความถูกต้องของการทำงาน (Functionality) เนื่องจากได้รับการพัฒนาและปรับปรุงจากนักพัฒนาซอฟต์แวร์ที่มีมุมมองและประสบการณ์ที่หลากหลาย แต่อาจจะมีข้อจำกัดในเรื่องคุณภาพบางประการ เพราะนักพัฒนาส่วนใหญ่จะมุ่งเน้นไปที่การพัฒนาการทำงานของระบบซอฟต์แวร์ให้ทำงานได้ตามต้องการ ทำให้คุณภาพของซอร์สโค้ดอาจจะไม่ดีเท่าที่ควร คุณลักษณะเหล่านี้มีแนวโน้มที่จะก่อให้เกิดร่องรอยของโค้ดที่ไม่ดี (Code Smell) ซึ่งเป็นสาเหตุที่ทำให้เกิดปัญหาในด้านคุณภาพซอฟต์แวร์ เช่น ความสามารถในการใช้งานและการบำรุงรักษา หลักการปฏิบัติอย่างหนึ่งในงานวิศวกรรมซอฟต์แวร์ที่สามารถลดปัญหาดังกล่าวลงได้ คือ การทำรีแฟคทอริง (Refactoring) การทำรีแฟคทอริงเป็นการปรับปรุงโครงสร้างโค้ดโดยไม่ทำให้การทำงานของซอฟต์แวร์เปลี่ยนแปลงไป โดยทั่วไปนักพัฒนาซอฟต์แวร์โครงการโอเพนซอร์สจะใช้วิธีการตรวจทานโค้ด (Code Review) ในการค้นหาข้อบกพร่องของซอร์สโค้ดที่ได้รับการปรับปรุงหรือแก้ไข จากการทบทวนวรรณกรรมที่เกี่ยวข้องกับการพัฒนาซอฟต์แวร์โอเพนซอร์ส ผู้วิจัยยังไม่พบการศึกษาการทำรีแฟคทอริงจากผลของการตรวจทานโค้ด

เพื่อเพิ่มความเข้าใจในกระบวนการตรวจทานโค้ดในโครงการโอเพนซอร์ส ผู้วิจัยมีความต้องการรวบรวมหลักฐานเชิงประจักษ์จากข้อเสนอแนะที่ผู้ตรวจทานโค้ดได้ระบุให้มีการทำรีแฟคทอริงเพื่อแก้ไขร่องรอยของโค้ดที่ไม่ดีในโครงการโอเพนซอร์ส ในงานวิจัยนี้ผู้วิจัยทำการศึกษาถึงความสัมพันธ์ระหว่างข้อเสนอแนะที่เกิดขึ้นในกระบวนการตรวจทานโค้ด และการทำรีแฟคทอริงจากร่องรอยโค้ดที่ไม่ดีในโครงการโอเพนซอร์ส นอกจากนี้ผู้วิจัยทำการจัดกลุ่มของร่องรอยโค้ดที่ไม่ดีซึ่งมักจะปรากฏในโครงการซอฟต์แวร์โอเพนซอร์ส โดยใช้อัลกอริทึม Latent Dirichlet Allocation

มาประยุกต์ใช้ในการจัดกลุ่มร่องรอยโค้ดที่ไม่ดี ผู้วิจัยได้ทำการศึกษาการตรวจทานโค้ดจากข้อมูลในโครงการโอเพนซอร์ส 2 โครงการ ได้แก่ โครงการ OpenStack และ WikiMedia ซึ่งเป็นโครงการที่มีนักพัฒนาและผู้ตรวจทานโค้ดร่วมอยู่ในโครงการเป็นจำนวนมาก

จากผลการวิจัยพบว่า ผู้ตรวจทานโค้ดส่วนใหญ่ไม่ค่อยให้ความสำคัญกับการทำรีแฟคทอริง มีผู้ตรวจทานโค้ดแค่บางกลุ่มที่พยายามกระตุ้นในนักพัฒนาทำการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง โดยแนวโน้มของการให้คำแนะนำเกี่ยวกับร่องรอยโค้ดที่ไม่ดีมีอัตราที่เพิ่มสูงขึ้นเรื่อย ๆ ซึ่งสิ่งนี้อาจจะเป็นสัญญาณที่ดีว่าผู้ตรวจทานโค้ดเริ่มที่จะหันมาให้ความสนใจเกี่ยวกับการปรับปรุงคุณภาพของโค้ดด้วยวิธีการทำรีแฟคทอริง ผู้วิจัยเชื่อว่าผลที่ได้จากงานวิจัยนี้จะช่วยเพิ่มองค์ความรู้ใหม่ที่เกี่ยวกับการเพิ่มคุณภาพของซอฟต์แวร์ในโครงการซอฟต์แวร์โอเพนซอร์ส และเพิ่มหลักฐานเชิงประจักษ์ให้กับนักวิจัยด้านวิศวกรรมซอฟต์แวร์ ผู้วิจัยคาดหวังว่างานวิจัยนี้จะช่วยให้นักพัฒนาซอฟต์แวร์เห็นถึงความสำคัญของการนำวิธีการปฏิบัติทางด้านวิศวกรรมซอฟต์แวร์ เช่น การนำรีแฟคทอริงไปช่วยในการพัฒนาระบบซอฟต์แวร์ทั่วไป และซอฟต์แวร์โอเพนซอร์ส

คำสำคัญ: วิศวกรรมซอฟต์แวร์ วิศวกรรมซอฟต์แวร์เชิงประจักษ์ ซอฟต์แวร์โอเพนซอร์ส รีแฟคทอริง กระบวนการตรวจทานโค้ด ร่องรอยโค้ดที่ไม่ดี

Thesis Title	Refactoring Influences in the Code Review for Open Source Software Projects
Author	Ms. Apatta Chaisutanon
Major Program	Information Technology
Academic Year	2016

ABSTRACT

Recently, the use of open source software (OSS) has been growing. The OSS users claim that OSS is reliable in perspective of functional correctness because it is either developed or improved by various experienced developers. Rather than focusing on all quality aspects, the OSS developers emphasize on software functionality based on software requirement specifications. This characteristic will impose the poor code quality, which possibly introduces code smell in software. The code smells represent design situations that can affect the software quality, such as usability and maintainability. The well-known practice that the software engineering community often used to mitigate code smells is refactoring. The goal of refactoring is to improve the software design and code structure without changing the software's functionality. Typically, the OSS developers perform the code review to recover the defects in modified code. Based on the literature review, there is no study investigating the refactoring suggestions received from the code review process.

To better understand the code review process in OSS projects, the author would like to collect empirical evidence that the code reviewers advised the developers to remove the code smell by refactoring. In this research, the author would like to investigate the relationship between code review's comments and refactoring based on detected code smell. Additionally, the author would like to classify the groups of code smell existing in OSS projects by adopting the algorithm, called Latent Dirichlet Allocation. The author does examine two OSS projects,

including OpenStack and WikiMedia, which are contributed by many developers and code reviewers.

The results of this study show that the reviewer most rarely concerned about refactoring. There were few revierwes who were attempting to encourage developers to improve the code by refactoring methods. However, the trend of comments related to code smells is increasing. This evidence indicates a good sign that the reviewers concentrate on improving the quality of software by refactoring. The author believes that this research would increase the knowledge and empirical evidence about the quality of OSS projects. Finally, the author expects that this research would also help software developers realize on employing software engineering practices, such as refactoring on both the traditional software development and OSS development.

Keyword: Software Engineering, Empirical Software Engineering, Open Source Software, Refactoring, Code Review, Code Smell

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยความช่วยเหลือจาก ผู้ช่วยศาสตราจารย์ ดร.อชีส นันทอมรพงศ์ อาจารย์ที่ปรึกษาวิทยานิพนธ์ ขอขอบพระคุณอาจารย์ที่ให้คำปรึกษาและชี้แนวทาง พร้อมทั้งให้ความรู้ในด้านต่าง ๆ ตลอดระยะเวลาของการทำวิทยานิพน์ จนสำเร็จลุล่วงไปด้วยดี

ขอขอบพระคุณผู้ช่วยศาสตราจารย์ ดร.รัตนา เวทย์ประสิทธิ์ ประธานกรรมการสอบ และผู้ช่วยศาสตราจารย์ ดร.ทรงศรี ตั้งศรีไพโรจน์ กรรมการสอบวิทยานิพนธ์ ที่ให้คำแนะนำ และแก้ไขข้อบกพร่องในวิทยานิพนธ์จนสำเร็จสมบูรณ์

ขอขอบพระคุณทุนสนับสนุนการวิจัยจากบัณฑิตวิทยาลัยและกองทุนวิจัยคณะเทคโนโลยีและสิ่งแวดล้อม มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตภูเก็ต ที่สนับสนุนทุนการศึกษาตลอดระยะเวลาทำการศึกษา

ขอขอบพระคุณอาจารย์และเจ้าหน้าที่ คณะเทคโนโลยีและสิ่งแวดล้อม มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตภูเก็ต ทุกท่าน สำหรับความช่วยเหลือในด้านต่าง ๆ ตลอดระยะเวลาทำการศึกษา

ขอขอบพระคุณเพื่อนนักศึกษาปริญญาโท คณะเทคโนโลยีและสิ่งแวดล้อม มหาวิทยาลัยสงขลานครินทร์ วิทยาเขตภูเก็ต ทุกท่าน สำหรับความช่วยเหลือในด้านต่าง ๆ ตลอดระยะเวลาการทำวิทยานิพนธ์ในครั้งนี้

สุดท้ายนี้ขอขอบพระคุณบิดา มารดา ญาติ พี่น้อง และคณาจารย์ทุกท่านที่ประสิทธิ์ประสาทวิชาความให้แก่ข้าพเจ้า รวมถึงบรรดามิตรสหายที่คอยเป็นกำลังใจและคอยช่วยเหลือตลอดระยะเวลาที่ทำการศึกษาจนข้าพเจ้าสามารถสำเร็จการศึกษาลุล่วงไปด้วยดี

อปัตตา ชัยสุตานนท์

สารบัญ

สารบัญ (ต่อ)

รายการตาราง

รายการตาราง (ต่อ)

รายการรูปภาพ

หน้า

รายการรูปภาพ (ต่อ)

หน้า

บทที่ 1

บทนำ

1.1 ความสำคัญและที่มาของการวิจัย

ในปัจจุบันซอฟต์แวร์โอเพนซอร์สได้กลายมาเป็นตัวเลือกที่สำคัญในการนำซอฟต์แวร์ไปใช้ และยังมีแนวโน้มที่จะถูกนำไปใช้งานในอัตราที่เพิ่มสูงขึ้นในอนาคต ซอฟต์แวร์โอเพนซอร์สได้เข้ามามีบทบาทในการใช้งานระบบสารสนเทศของผู้ใช้ในหลายกลุ่ม เช่น บริษัทเอกชน การทำธุรกรรมออนไลน์ งานวิจัย การศึกษา สาธารณสุข การท่องเที่ยว วิธีการพัฒนาซอฟต์แวร์โอเพนซอร์ส มีลักษณะเฉพาะที่สำคัญประการหนึ่ง คือ การเปิดโอกาสให้นักพัฒนาที่มีความสนใจเข้าร่วมการพัฒนาซอฟต์แวร์โดยไม่จำเป็นต้องรู้จักกันมาก่อน หรืออยู่ในสถานที่เดียวกัน (Aberdour, 2007) เราจึงเห็นได้ว่าโครงการซอฟต์แวร์โอเพนซอร์สในหลายโครงการมีนักพัฒนาซอฟต์แวร์จากหลายประเทศทั่วโลกเข้าร่วม การทำงานระหว่างนักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์สจะอาศัยการสื่อสารระหว่างกันในรูปแบบอิเล็กทรอนิกส์ เช่น อีเมล์ เว็บบอร์ด โปรแกรมสนทนาออนไลน์ จากคุณลักษณะข้อนี้นับว่าเป็นจุดแข็งที่ทำให้ซอฟต์แวร์โอเพนซอร์สสามารถถูกพัฒนาขึ้นมาได้โดยรวดเร็วและค่อนข้างมีความน่าเชื่อถือในแง่ของความถูกต้องของการทำงาน (Functionality) รวมถึงเทคโนโลยีใหม่ ๆ เนื่องจากได้รับการพัฒนาและปรับปรุงจากนักพัฒนาซอฟต์แวร์ที่มีมุมมอง และประสบการณ์ที่หลากหลาย

จากงานวิจัยก่อนหน้านี้ได้ระบุให้เห็นถึงข้อจำกัดเรื่องคุณภาพของซอฟต์แวร์โอเพนซอร์สในหลายมิติ เช่น เรื่องความมั่นคง (Security) ความสามารถในการบำรุงรักษา (Maintainability) (Zhou and Davis, 2005) จากข้อจำกัดดังกล่าวอาจจะเนื่องมาจากการปรับปรุงคุณภาพของซอฟต์แวร์โอเพนซอร์สโดยทั่วไปอยู่ในรูปแบบของการแก้ไขปัญหาเฉพาะหน้า คือ นักพัฒนาจะมุ่งเน้นการแก้ไขปัญหาความผิดพลาด (Bug) เฉพาะในส่วนที่ได้รับรายงานจากผู้ใช้ หรือนักพัฒนาด้วยกันเองจึงทำให้เกิดข้อจำกัดเรื่องคุณภาพของซอฟต์แวร์ ในมุมมองของวิศวกรซอฟต์แวร์จะไม่มองเพียงแค่การพัฒนาฟังก์ชันการทำงาน แต่จะมองลึกลงไปถึงโครงสร้างการทำงานของโค้ดและการบำรุงรักษา หากโค้ดมีโครงสร้างที่ดีก็จะส่งผลให้ซอฟต์แวร์นั้นมีคุณภาพด้วย

เมื่อวิวัฒนาการของซอฟต์แวร์ผ่านไปซอฟต์แวร์ก็ยิ่งมีความเสี่ยงต่อการเกิดข้อบกพร่อง เนื่องจากซอฟต์แวร์โอเพนซอร์สสามารถทำการปรับเปลี่ยนเพื่อให้เกิดความเหมาะกับการนำไปใช้งาน และให้มีประสิทธิภาพในการทำงานดีขึ้น เมื่อผู้ใช้นำซอฟต์แวร์ไปใช้งาน อาจจะมองเห็นแนวทางการเพิ่มฟังก์ชันการทำงาน หรือมีความต้องการเพิ่มเติม จึงมีความจำเป็นที่จะต้องทำการปรับปรุงระบบให้ดีขึ้นเรื่อย ๆ ซึ่งจะส่งผลให้ซอฟต์แวร์มีความซับซ้อนเพิ่มมากขึ้น หากไม่ให้ความสนใจในเรื่องคุณภาพของโค้ดหรือให้ความสนใจเฉพาะผลลัพธ์ ในอนาคตมีแนวโน้มว่าซอร์สโค้ดนั้นอาจจะก่อให้เกิดปัญหาหรือเกิดข้อบกพร่องในซอฟต์แวร์ ชุมชนวิศวกรซอฟต์แวร์ได้พยายามหาวิธีป้องกันปัญหาเหล่านั้นก่อนที่จะกลายเป็นปัญหาใหญ่จนถึงขั้นที่ทำให้ซอฟต์แวร์ทำงานไม่ได้ เพราะหากมีปัญหาเกิดขึ้นก็ย่อมจะส่งผลกระทบในหลาย ๆ ด้าน เช่น เรื่องของเวลา เรื่องของค่าใช้จ่าย ปัญหาเหล่านี้จะเกิดน้อยลงหากมีการจัดการโค้ดที่ดี วิธีการหนึ่งที่ทำให้โค้ดที่เขียนมีคุณภาพดีขึ้น คือ การตรวจทานโค้ด (Code Review) (Baysal, *et al.*, 2013) ซึ่งเป็นวิธีการหนึ่งที่ได้รับการยอมรับในงานด้านวิศวกรรมซอฟต์แวร์ การตรวจทานโค้ดจะเป็นการประเมินซอร์สโค้ดโดยนักพัฒนาซอฟต์แวร์ คือ การใช้คนทำการตรวจทานโค้ด ก่อนจะมีการนำโค้ดไปบูรณาการกับส่วนอื่น ๆ หลักการที่สำคัญของการตรวจทานโค้ด คือ การให้นักพัฒนาซอฟต์แวร์คนอื่น ๆ ช่วยระบุข้อบกพร่อง และปัญหาคุณภาพในซอร์สโค้ด รวมถึงโค้ดส่วนที่อาจจะก่อให้เกิดปัญหา ก่อนที่จะถูกนำไปใช้งานจริง (Paulson, *et al.*, 2004) ซึ่งปัญหาในโค้ดดังกล่าวนั้นรวมไปถึงตรรกะเบื้องหลังโค้ด รูปแบบของโค้ด เรื่องความมั่นคง เช่น การเข้ารหัสข้อมูล การเก็บบันทึก (Log) ต่าง ๆ และรวมไปถึงการทำให้โค้ดดีขึ้นกว่าเดิม (Scacchi, 2002) เช่น ทำให้โค้ดอ่านง่ายขึ้น เข้าใจง่ายขึ้น ซึ่งจะส่งผลให้โค้ดแก้ไขง่ายขึ้น การตรวจทานโค้ดไม่ได้ส่งผลดีเฉพาะในส่วนของการเขียนโค้ด แต่ยังรวมไปถึงการสื่อสารเกี่ยวกับคุณสมบัติของซอฟต์แวร์ที่ลูกค้าต้องการ (Requirement) การวิเคราะห์ระบบ (Analysis) การออกแบบ (Design) การทดสอบระบบ (Testing) และการนำไปติดตั้งใช้งาน (Deploy)

ในทางวิศวกรรมซอฟต์แวร์จะมีวิธีการในการปรับปรุงคุณภาพซอฟต์แวร์โดยเฉพาะในส่วนของซอร์สโค้ดอยู่หลายวิธี หนึ่งในวิธีเหล่านั้น คือ การทำรีแฟคทอริง (Refactoring) ซึ่งเป็นการเปลี่ยนแปลงโครงสร้างภายในของซอฟต์แวร์ โดยไม่เปลี่ยนแปลงการทำงานเดิมของซอฟต์แวร์ โดยมีวัตถุประสงค์เพื่อทำให้ซอฟต์แวร์นั้นง่ายต่อการทำความเข้าใจ ซึ่งจะเพิ่มความสามารถในการบำรุงรักษาซอฟต์แวร์และรองรับการเปลี่ยนของโค้ดในอนาคต (Fowler and Beck, 1999) สิ่งสำคัญก่อนจะมีการทำรีแฟคทอริงได้นั้นจะต้องทำการระบุให้ได้ก่อนว่าจะทำการรีแฟคทอริงในซอร์สโค้ดส่วนไหน โดยทั่วไปแล้วการทำรีแฟคทอริงจะถูกดำเนินการในส่วนร่องรอยโค้ดที่ไม่ดี (Code Smell) ซึ่งเป็นส่วนของโค้ดที่มีโอกาสจะก่อให้เกิดความผิดพลาด หรือข้อบกพร่อง โดยอาจจะมีสาเหตุมาจากการออกแบบที่ไม่ดี หรือเกิดจากความผิดพลาดของนักพัฒนาซอฟต์แวร์ เช่น เมื่อมีการทำงานในเมทธอดที่ซ้ำ ๆ กันมาก ๆ ก็ควรที่จะสร้างเมทธอดใหม่ขึ้นมาให้เรียกใช้งาน หรือเป็นการแก้ไขการ

ตั้งชื่อตัวแปรต่าง ๆ ให้สื่อความหมายกับหน้าที่ ที่ตัวแปรนั้นทำงาน รวมไปถึงการระบุคำอธิบายเข้าไปในโค้ดด้วย

การมีร่องรอยของโค้ดที่ไม่ดีในซอฟต์แวร์ถือเป็นปัญหาหนึ่งที่สำคัญในการส่งผลกระทบต่อคุณภาพของซอฟต์แวร์ (Halloran and William, 2002) ทั้งในด้านการนำไปใช้งานและการบำรุงรักษา ซึ่งจะรวมถึงเรื่องของการสร้างความเชื่อมั่นให้กับผู้ใช้งานซอฟต์แวร์อีกด้วย เพื่อเพิ่มความเข้าใจในเรื่องการตรวจทานโค้ดในโครงการโอเพนซอร์สให้ดียิ่งขึ้น โดยเฉพาะการให้ความสนใจของผู้ตรวจทานโค้ดต่อร่องรอยโค้ดที่ไม่ดี ผู้วิจัยจึงสนใจทำการศึกษาอิทธิผลของการทำรีแฟคทอริงที่มีต่อการตรวจทานโค้ดในโครงการซอฟต์แวร์โอเพนซอร์ส เนื่องจากข้อมูลที่ได้จากการศึกษาจะช่วยให้นักวิจัยเข้าใจถึงการเพิ่มคุณภาพของซอฟต์แวร์โอเพนซอร์ส โดยเฉพาะการลดจำนวนร่องรอยโค้ดที่ไม่ดีและมีความเข้าใจเกี่ยวกับร่องรอยโค้ดที่ไม่ดีประเภทต่าง ๆ ที่มักจะพบได้ในโครงการซอฟต์แวร์โอเพนซอร์ส สำหรับปัญหาวิจัยที่ผู้วิจัยได้กำหนดไว้ในงานวิจัยนี้ คือ

1) ผู้ตรวจทานโค้ดให้ความสำคัญกับการทำรีแฟคทอริงในโครงการซอฟต์แวร์โอเพนซอร์สหรือไม่
2) ร่องรอยของโค้ดที่ไม่ดีในโครงการซอฟต์แวร์โอเพนซอร์สแบ่งออกเป็นกี่ประเภท

เพื่อตอบคำถามงานวิจัยที่ได้ระบุไว้ ผู้วิจัยได้ทำการค้นหาประเภทของร่องรอยโค้ดที่ไม่ดี โดยการใช้กระบวนการทำเหมืองข้อความเพื่อทำการคัดกรองหาข้อเสนอแนะและนำมาจัดกลุ่มเข้าด้วยกันซึ่งจะนำข้อมูลเชิงปริมาณ (Quantitative) ที่ได้จากการค้นหานี้มาวิเคราะห์ผล โดยใช้อัลกอริทึม Latent Dirichlet Allocation (LDA) ช่วยในการจัดกลุ่มประเภทของร่องรอยโค้ดที่ไม่ดีซึ่งจะซ่อนอยู่ภายในประโยคต่าง ๆ ของข้อเสนอแนะ (Comment) จากผู้ตรวจทานโค้ดจำนวนมากในโครงการซอฟต์แวร์โอเพนซอร์สขนาดใหญ่ ในการค้นหาข้อเสนอแนะในครั้งนี้จะมุ่งเน้นทำการศึกษาไปยังข้อเสนอแนะที่มีการแนะนำให้นักพัฒนาทำการปรับปรุงโค้ดด้วยการใช้วิธีการทำรีแฟคทอริง โดยมีการนำข้อเสนอแนะเหล่านี้มาจากระบบการให้บริการการตรวจทานโค้ดที่มีชื่อว่า เกอริต (Gerrit)[1] ซึ่งเกอริตนี้เป็นระบบที่ช่วยอำนวยความสะดวกในเรื่องการตรวจทานโค้ดให้กับนักพัฒนาซอฟต์แวร์ ระบบนี้จะช่วยอำนวยความสะดวกให้กับนักพัฒนาซอฟต์แวร์ในกระบวนการการตรวจทานโค้ด ในปัจจุบันมีโครงการโอเพนซอร์สมากว่า 2,000 โครงการ ที่ใช้ระบบเกอริตในกระบวนการตรวจทานโค้ด ระบบเกอริตนั้นจะมีการทำงานร่วมกับระบบจัดการกิต (Git)[2] ซึ่งเป็นระบบซอฟต์แวร์ที่ใช้ในการจัดการเวอร์ชันของซอร์สโค้ด

[1] https://www.gerritcodereview.com/

[2] https://git-scm.com/

1.2 วัตถุประสงค์

1.2.1 ทำการศึกษาหาอิทธิพลของการทำรีแฟคทอริงต่อการตรวจทานโค้ดในโครงการซอฟต์แวร์โอเพนซอร์ส

1.2.2 พัฒนากระบวนการในการค้นหา และจัดกลุ่มประเภทของร่องรอยโค้ดที่ไม่ดีในโครงการซอฟต์แวร์โอเพนซอร์ส

1.2.3 เพื่อทำการจัดประเภท และเก็บรวบรวมข้อมูลเชิงประจักษ์ของร่องรอยโค้ดที่ไม่ดีในโครงการซอฟต์แวร์โอเพนซอร์ส

1.3 ขอบเขตของงานวิจัย

งานวิจัยนี้ผู้วิจัยใช้ข้อมูลจากโครงการซอฟต์แวร์โอเพนซอร์ส ซึ่งได้มาจากระบบเกอริตโดยทำการเลือกมา 2 โครงการ ได้แก่ OpenStack[3] และ WikiMedia[4] เนื่องจากทั้ง 2 โครงการที่เลือกมามีข้อมูลให้ทำการศึกษาอยู่เป็นจำนวนมาก เป็นโครงการที่มีการใช้งานอยู่ในปัจจุบันและมีการพัฒนาอยู่ตลอดเวลา โครงการ OpenStack เป็นระบบการประมวลผลแบบกลุ่มเมฆ (Cloud Computing) ผู้วิจัยสนใจที่จะศึกษาโครงการนี้เพราะในปัจจุบันเป็นยุคของโลกสังคมออนไลน์ สังคมดิจิทัล และในอนาคตระบบการประมวลผลข้อมูลแบบกลุ่มเมฆจะเข้ามามีบทบาทสำคัญต่อองค์กรธุรกิจ หน่วยงานราชการ และสถานศึกษา เพราะเปรียบเสมือนคลังเก็บข้อมูลออนไลน์ที่มีขนาดความจุมหาศาล บริการรับฝากไฟล์ความจุสูงและบริการด้านข้อมูลนานาชนิดบนโลกออนไลน์แบบไร้ขีดจำกัด จึงทำให้องค์กรชั้นนำหลายแห่งให้ความสำคัญกับการนำระบบการประมวลผลข้อมูลแบบกลุ่มเมฆมาใช้เพื่อเพิ่มขีดความสามารถในการแข่งขันและสร้างความแตกต่างขององค์กร และสามารถทำให้ลูกค้าสามารถเข้าถึงได้ง่ายและรวดเร็ว ในส่วนของโครงการ WikiMedia เป็นระบบที่ใช้สร้างเป็นเว็บไซต์ที่เป็นวิกิ ผู้วิจัยสนใจที่จะศึกษาโครงการนี้เพราะเป็นระบบที่มีขนาดใหญ่เปรียบเสมือนแหล่งรวบรวมความรู้จากบุคคลทั่วโลกที่เกิดจากการร่วมกันคิดร่วมกันระดมความรู้จากอาสาสมัครจากทั่วทุกมุมโลกเพื่อสร้างสารานุกรมเสรีออนไลน์[5] และเหตุผลสำคัญอีกประการหนึ่งคือ โครงการ OpenStack และ WikiMedia

[3] https://www.openstack.org/

[4] https://www.Wikimedia.org/

[5] https://www.wikipedia.org/

เป็นโครงการโอเพนซอร์สซอฟต์แวร์ที่ได้รับความนิยมนำมาศึกษาในงานวิจัยทางด้านวิศวกรรมซอฟต์แวร์ที่เกี่ยวข้องกับการตรวจทานโค้ด (Bosu and Carver, 2013; Bosu, *et al.*, 2014; Thongtanunam, *et al.*, 2014 และ Xia, *et al.*, 2015) สำหรับข้อมูลที่จะเก็บรวบรวม คือ ข้อเสนอแนะของผู้ตรวจทานโค้ดที่มีการแนะนำให้นักพัฒนาทำการแก้ไขหรือปรับปรุงซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริงในช่วงปี ค.ศ. 2011- 2015 และนำข้อมูลมาวิเคราะห์ผล ตารางที่ 1แสดงรายละเอียดของโครงการโอเพนซอร์สทั้ง 2 โครงการ

ตารางที่ 1.1 รายละเอียดของโครงการโอเพนซอร์ส

โครงการโอเพนซอร์ส	รายละเอียดของโครงการ
OpenStack	โครงการโอเพนซอร์สที่รวมเอาซอฟต์แวร์ต่าง ๆ ที่ช่วยสร้าง และจัดการแพล็ตฟอร์มของระบบการประมวลผลแบบกลุ่มเมฆ (Cloud Computing) โดยโครงการโอเพนซอร์สนี้ได้รับการสนับสนุนจากบริษัทซอฟต์แวร์ และนักพัฒนาซอฟต์แวร์ทั่วโลก ในปัจจุบันโครงการนี้ถูกบริหารโดยกลุ่มองค์กรไม่แสวงหากำไรที่เรียกว่า OpenStack Foundation ซึ่งประกอบด้วยสมาชิกไม่น้อยกว่า 28,000 คน
WikiMedia	ซอฟต์แวร์ที่ใช้สร้างเป็นเว็บไซต์ที่เป็นวิกิ (เว็บไซต์ที่เปิดให้ผู้สนใจได้เขียน แก้ไขบทความโดยอิสระ) ในปัจจุบัน WikiMedia ได้ถูกใช้ในการสร้างโครงการเว็บไซต์ที่เป็นวิกิจำนวนมาก แต่เว็บไซต์ที่เป็นที่รู้จักกันอย่างดี คือ Wikipedia นั้นเอง

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1.4.1 ผู้วิจัยและนักพัฒนาซอฟต์แวร์มีความเข้าใจการตรวจทานโค้ดในโครงการซอฟต์แวร์โอเพนซอร์สมากยิ่งขึ้น

1.4.2 เพิ่มหลักฐานเชิงประจักษ์ที่เกี่ยวกับการเพิ่มคุณภาพของซอฟต์แวร์ในโครงการซอฟต์แวร์โอเพนซอร์สให้กับนักวิจัยด้านวิศวกรรมซอฟต์แวร์

1.4.3 ทำให้นักพัฒนาได้ตระหนักและให้ความสำคัญกับการนำรีแฟคทอริงไปใช้ในการพัฒนาระบบได้

บทที่ 2

ความรู้พื้นฐาน เทคโนโลยีและวรรณกรรมที่เกี่ยวข้อง

ในบทนี้จะเป็นการอธิบายความรู้พื้นฐาน เทคโนโลยีและวรรณกรรมที่เกี่ยวข้อง โดยจะเริ่มที่พื้นฐานที่เกี่ยวข้องด้านซอฟต์แวร์โอเพนซอร์ส การตรวจทานโค้ด และการทำรีแฟคทอริง หลังจากนั้นจะเป็นการอธิบายเทคโนโลยีที่นำมาประยุกต์ใช้ในงานวิจัยครั้งนี้ และงานวิจัยก่อนหน้าที่เกี่ยวข้องกับการดำเนินงานวิจัยนี้

2.1 ความรู้พื้นฐาน

ความรู้พื้นฐานที่เกี่ยวข้องในงานวิจัย ได้แก่ ซอฟต์แวร์โอเพนซอร์ส การตรวจทานโค้ด ร่องรอยโค้ดที่ไม่ดี รีแฟคทอริง และอัลกอริทึม Latent Dirichlet Allocation (LDA)

2.1.1 ซอฟต์แวร์โอเพนซอร์ส

ซอฟต์แวร์โอเพนซอร์ส หมายถึง ซอฟต์แวร์ที่อนุญาตให้ผู้ใช้มีอิสระในการใช้งาน มีอิสระในการเผยแพร่ และแก้ไขตัวซอร์สโค้ดโปรแกรม ปัจจุบันมีนักพัฒนาซอฟต์แวร์โอเพนซอร์สจำนวนมากจากทั่วโลกช่วยกันพัฒนาซอฟต์แวร์โอเพนซอร์ส โดยเฉพาะการพัฒนาซอฟต์แวร์ผ่านเครือข่ายอินเทอร์เน็ต (Aberdour, 2007) ทำให้เกิดซอฟต์แวร์โอเพนซอร์สที่มีคุณภาพหลายโครงการ ซึ่งได้รับการยอมรับอย่างกว้างขวาง และกำลังใช้งานอยู่ในองค์กรทั่วโลก ตัวอย่างซอฟต์แวร์โอเพนซอร์สที่เป็นที่รู้จักกันอย่างแพร่หลาย ได้แก่ ระบบปฏิบัติการลีนุกซ์ และเว็บเซิร์ฟเวอร์ Apache (Baysal, *et al.*, 2013) โดยทั่วไปเราสามารถเข้าไปดาวน์โหลดใช้งานซอฟต์แวร์โอเพนซอร์สจากเว็บไซต์ที่ให้บริการจัดเก็บและบริหารโครงการโอเพนซอร์สโดยเฉพาะ ได้แก่ Sourceforge[6] GitHub[7] และ Google code[8]

[6] https://sourceforge.net/

[7] https://github.com/

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

Mark Aberdour (2013) ได้กำหนดคุณลักษณะที่สำคัญของการพัฒนาซอฟต์แวร์โอเพนซอร์ส ไว้ดังนี้

1) เป็นซอฟต์แวร์ที่มีการเผยแพร่ได้อย่างเสรี
2) เป็นซอฟต์แวร์ที่ให้ใช้ได้ฟรี
3) มีซอร์สโค้ดพร้อมที่จะใช้งาน
4) นักพัฒนาซอฟต์แวร์นิยมสื่อสารผ่านทางอินเทอร์เน็ต
5) นักพัฒนาเองจะเป็นผู้ใช้ซอฟต์แวร์โอเพนซอร์ส
6) ไม่มีค่าจ้างและมีอาสาสมัครจำนวนมาก

นอกจากนี้ Mark Aberdour ได้ระบุถึงปัจจัยที่มีผลต่อคุณภาพของโครงการซอฟต์แวร์โอเพนซอร์ส ซึ่งปัจจัยที่มีผลต่อคุณภาพของโครงการซอฟต์แวร์โอเพนซอร์สประกอบไปด้วย

1) ความยั่งยืนของนักพัฒนาซอฟต์แวร์ เป็นการพัฒนาความรู้และศักยภาพของบุคลากรในสายงานด้านการพัฒนาซอฟต์แวร์ให้มีความรู้ และความเข้าใจเรื่องกระบวนการผลิตและพัฒนาซอฟต์แวร์ที่ได้มาตรฐาน มีคุณภาพ และผลิตซอฟต์แวร์ได้ตรงตามความต้องการของผู้ใช้

2) แรงจูงใจและการมีส่วนร่วมของนักพัฒนาซอฟต์แวร์ การสร้างแรงจูงใจเป็นสิ่งที่มีอิทธิพลต่อความทุ่มเทในการทำงาน เป็นเรื่องที่เกี่ยวข้องกับอารมณ์ ความรู้สึกของคนในการปฏิบัติงานซึ่งจะส่งผลต่อการปฏิบัติงานให้มีประสิทธิภาพ และทำให้ซอฟต์แวรต์ที่พัฒนานั้นมีประสิทธิภาพที่ดีตามไปด้วย

3) การแบ่งโมดูลของโค้ด (Paulson, *et al.*, 2004) การแยกเขียนโปรแกรมออกเป็นส่วน ๆ ที่สามารถทำงานเป็นอิสระ ทำให้แก้ไขง่าย และใช้คนเขียนหลายคนได้ ประโยชน์จากการเขียนแบบโมดูลในการพัฒนา คือ สามารถคอมไพล์ได้รวดเร็วขึ้น ง่ายต่อการดูแล ง่ายต่อการทดสอบ การใช้ทรัพยากรในการเขียนโปรแกรมที่ดีขึ้น เพิ่มความสะดวกในการโอนย้ายโค้ดจากระบบปฏิบัติการอื่น ๆ ประโยชน์เหล่านี้ส่งผลโดยตรงกับการทำให้คุณภาพซอฟต์แวร์ดีขึ้น

4) การตรวจทานโค้ดโดยการใช้นักพัฒนาด้วยกันเอง (อ้างอิงจากเว็บ martinfowler.com, วันที่ 25 กรกฎาคม 2559; Oh and Choi, 2005 และ Tao, *et al.*, 2014) การตรวจทานโค้ดที่ทำได้ง่ายที่สุดจะเป็นลักษณะของการแลกเปลี่ยนกันตรวจทานโค้ดภายในทีม ผู้ที่จะทำหน้าที่ตรวจทานงานของผู้อื่นได้จำเป็นต้องมีความเชี่ยวชาญเหมือนกับผู้ที่พัฒนาระบบนั้น หรือเชี่ยวชาญกว่า โดยทั่วไปมักจะใช้ผู้ร่วมงานในระดับเดียวกันเป็นผู้ตรวจทาน การตรวจทานโค้ดนี้จะช่วยให้นักพัฒนาสามารถตรวจพบจุดผิดพลาดได้อย่างง่ายดายภายในเวลาอันรวดเร็ว

5) กระบวนการทดสอบ (Oh and Choi 2005; Tao, *et al.,* 2014 และ Zhou and Davis, 2005) กระบวนการทดสอบ เป็นกระบวนการที่จะช่วยให้ซอฟต์แวร์ที่พัฒนามีความถูกต้อง มีความสมบูรณ์ มีความมั่นคง และมีคุณภาพที่ดี โดยใช้ความรู้ทางด้านเทคนิคของการทดสอบ เช่น การทดสอบส่วนย่อย (Unit Testing) และการทดสอบระบบ (System Testing) เพื่อให้สามารถระบุหรือค้นหาความผิดพลาดของซอฟต์แวร์ที่อาจจะซ่อนอยู่ และสามารถพยากรณ์ความผิดพลาดที่อาจจะเกิดขึ้นได้

จากปัจจัยข้างต้น ในส่วนของปัจจัยที่ 4) และ 5) จะเกี่ยวข้องกับงานวิจัยนี้โดยตรง เนื่องจากหลังจากการตรวจทานโค้ด และกระบวนการทดสอบ นักพัฒนาจะทราบถึงปัญหาทางด้านคุณภาพของโค้ด และจะดำเนินการปรับปรุงซอฟต์แวร์ให้มีคุณภาพดีขึ้น

2.1.2 การตรวจทานโค้ด (Code Review)

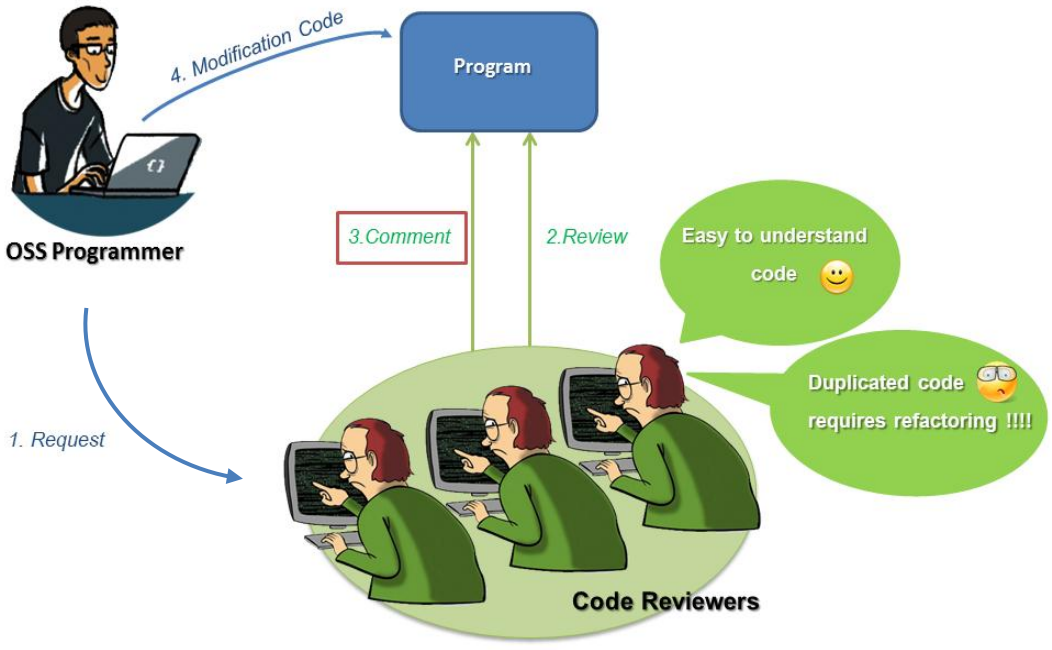
การตรวจทานโค้ด หมายถึง การตรวจสอบดูว่าซอร์สโค้ดส่วนไหนของระบบมีปัญหา หรือซอร์สโค้ดส่วนไหนมีโอกาสที่จะเกิดปัญหา โดยทั่วไปการตรวจทานโค้ดในโอเพนซอร์สจะใช้วิธีที่เรียกว่า การตราวจทานโค้ดโดยผู้รู้เสมอกัน (Peer Code Review) (Bosu and Carver, 2013) หรือที่นิยมเรียกกันว่า การตรวจทานโค้ดสมัยใหม่ (Modern Code Review) (Beller, *et al*., 2014) ซึ่งการตรวจทานโค้ดวิธีนี้จะแตกต่างกับการตรวจทานโค้ดแบบดั้งเดิมที่ใช้ในงานวิศวกรรมซอฟต์แวร์ที่เรียกว่า การตรวจสอบโค้ดอย่างละเอียด (Code Inspection) ซึ่งคิดค้นโดย Michael Fagan (1999) ในปี ค.ศ. 1976 สำหรับการตรวจสอบโค้ดอย่างละเอียดนี้จะมีกระบวนการอย่างมีระบบ และจะมีการกำหนดเกณฑ์การตรวจสอบโค้ดไว้อย่างชัดเจน ก่อนจะมีการเริ่มตรวจสอบโค้ดผู้ตรวจสอบจะต้องมีการประชุมตกลงขั้นตอนวิธีการในการตรวจสอบ และเมื่อมีการตรวจพบข้อบกพร่องจะมีการจดบันทึกไว้อย่างละเอียด โดยวิธีการนี้จะแตกต่างจากการตรวจทานโค้ดสมัยใหม่ ซึ่งจะไม่มีกระบวนการที่มีระบบ การตรวจทานโค้ดจะอาศัยประสบการณ์ของผู้ตรวจทานเป็นหลัก และเน้นความรวดเร็วในการตรวจทาน

การตรวจทานโค้ดเป็นวิธีการที่เป็นที่ยอมรับกันทางด้านวิศวกรรมซอฟต์แวร์ว่าเป็นวิธีที่ดีในการตรวจสอบคุณภาพของซอฟต์แวร์ ซึ่งจะประกอบด้วยการตรวจสอบโดยนักพัฒนาคนอื่น ๆ โดยส่วนใหญ่การตรวจทานโค้ดในลักษณะนี้มักจะมีวัตถุประสงค์เพื่อระบุข้อบกพร่องและปัญหาคุณภาพในซอร์สโค้ดก่อนที่จะมีการนำโค้ดนั้นไปใช้งานจริง (Paulson, *et al.*, 2004 และ Beller, *et al.*, 2014) เพื่อให้มั่นใจว่าซอร์สโค้ดที่พัฒนานั้นถูกต้อง ง่ายต่อการทำความเข้าใจ และตรงตามความต้องการของผู้ใช้งาน ต่อจากนั้นจะเริ่มดูในส่วนการจัดการข้อบกพร่องต่าง ๆ ทั้งจากการรับข้อมูลเข้ามาและส่วนที่เป็นผลลัพธ์ ซึ่งปัญหาดังกล่าวนั้นรวมไปถึงอัลกอริทึม รูปแบบของโค้ด ร่องรอยที่ไม่ดี ความมั่นคง เช่น

การเข้ารหัสข้อมูล โครงสร้างของข้อมูล และการทำให้โค้ดมีความสะอาด (Clean Code) ขึ้นกว่าเดิม (Scacchi, 2002) เช่น ทำให้โค้ดอ่านง่ายขึ้น เข้าใจง่ายขึ้น ง่ายต่อการแก้ไข โดยทั่วไปการตรวจทานโค้ดมี 2 แนวทาง (Beller, *et al.*, 2014) คือ

1) การตรวจทานด้วยนักพัฒนาคนอื่น ๆ โดยส่วนใหญ่นักพัฒนาที่อาวุโสกว่า หรือมีมีความชำนาญกว่าจะเป็นผู้ตรวจทานโค้ดให้ วิธีการนี้เป็นวิธีการที่ดีสำหรับการพัฒนาทักษะในการพัฒนาซอฟต์แวร์และเป็นการเพิ่มความรู้ความเข้าใจของทีมในการพัฒนาซอฟต์แวร์นั้น ๆ ด้วย แต่จะต้องใช้เวลาในการตรวจสอบโค้ด

2) การตรวจทานด้วยเครื่องมือวิเคราะห์ (Automatic code review) ในวิธีนี้นักพัฒนาซอฟต์แวร์จะใช้ซอฟต์แวร์ช่วยตรวจสอบโค้ดแบบอัตโนมัติ ตัวอย่างของซอฟต์แวร์ที่ใช้ได้แก่ Github pull request[9], Diff หรือ Wdiff[10] ซึ่งเครื่องมือที่ใช้จะทำให้ผู้ตรวจทานโค้ดสามารถทำงานได้สะดวกขึ้น เช่น การเปรียบเทียบโค้ดในเวอร์ชันก่อนหน้า การทำด้วยเครื่องมือจะช่วยกรองข้อมูล และจัดกลุ่มต่าง ๆ ของโค้ด เพื่อช่วยลดเวลาในการตรวจสอบด้วยนักพัฒนา อย่างไรก็ตามการตรวจสอบด้วยเครื่องมือไม่สามารถเข้ามาแทนการตรวจสอบด้วยนักพัฒนาได้ทั้งหมด การตรวจทานโค้ดยังจำเป็นต้องใช้ประสบการณ์ของผู้ตรวจทานโค้ดร่วมด้วย



รูปที่ 2.1 กระบวนการตรวจทานโค้ดในโครงการพัฒนาซอฟต์แวร์โอเพนซอร์ส

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

[10] https://www.diffchecker.com/

จากรูปที่ 2.1 เป็นการแสดงกระบวนการตรวจทานโค้ดในโครงการโอเพนซอร์ส ซึ่งกระบวนการตรวจสอบโค้ดจะเริ่มขึ้นเมื่อนักพัฒนาซอฟต์แวร์มีความต้องการที่จะทำการตรวจสอบซอฟต์แวร์ที่ได้พัฒนาขึ้น และมีขั้นตอนการทำงานดังนี้

1) นักพัฒนาซอฟต์แวร์จะทำการส่งคำร้องขอไปยังผู้ตรวจทานโค้ดเพื่อเป็นการขอให้ผู้ตรวจทานโค้ดทำการตรวจทานซอฟต์แวร์ที่ตนได้ทำการพัฒนาขึ้น

2) เมื่อผู้ตรวจทานโค้ดได้ยอมรับคำร้องขอก็จะเริ่มทำการตรวจทาน เพื่อหาข้อบกพร่องที่อาจจะเกิดขึ้น

3) หากผู้ตรวจทานโค้ดพบข้อบกพร่องที่ตรวจพบในซอฟต์แวร์ก็จะเพิ่มข้อเสนอแนะลงไปในซอฟต์แวร์ที่ได้ทำการตรวจเจอข้อบกพร่อง แล้วทำการส่งซอร์สโค้ดกลับไปยังนักพัฒนาเพื่อให้นักพัฒนาได้ทำการแก้ไขหรือปรับปรุงข้อบกพร่องตามข้อเสนอแนะที่ได้ทำการแนะนำไป

4) หลังจากนักพัฒนาซอฟต์แวร์ได้รับคำแนะนำจากผู้ตรวจทานโค้ดเรียบร้อยก็จะทำการแก้ไขปรับปรุงหรืออาจจะไม่ทำการแก้ไข หากนักพัฒนาได้ทำการแก้ไขก็จะเริ่มเข้าสู่ขั้นตอนในข้อที่ 1) อีกครั้ง

ในงานวิจัยนี้ผู้วิจัยทำการศึกษาในส่วนของข้อเสนอแนะของผู้ตรวจทานโค้ดเพื่อทำการหาคำตอบสำหรับคำถามของวิจัย คือ ผู้ตรวจทานโค้ดให้ความสำคัญกับการทำรีแฟคทอริงในโครงการซอฟต์แวร์โอเพนซอร์สหรือไม่

2.1.3 ร่องรอยโค้ดที่ไม่ดี (Code Smell)

ร่องรอยโค้ดที่ไม่ดี หมายถึง ลักษณะของโค้ดในซอฟต์แวร์ที่อาจจะก่อให้เกิดความผิดพลาดในการทำงานของซอฟต์แวร์ หรือลดคุณภาพของซอฟต์แวร์ซึ่งจะเพิ่มความเสี่ยงให้เกิดปัญหาต่าง ๆ ในอนาคตได้ (Fowler and Beck, 1999) โดยทางวิศวกรรมซอฟต์แวร์นักพัฒนาซอฟต์แวร์ควรจะตรวจหาร่องรอยโค้ดที่ไม่ดีก่อนที่มันจะกลายเป็นข้อบกพร่องซึ่งอาจจะทำให้เกิดการสูญเสียได้มาก Fowler and Beck (1999) ได้มีการระบุเกี่ยวกับร่องรอยโค้ดที่ไม่ดีไว้ทั้งหมด 22 ประเภท ต่อมา Matthew James Munro (2005) ได้นำร่องรอยโค้ดที่ไม่ดีทั้ง 22 ประเภท มาทำการแบ่งออกเป็น 2 กลุ่ม คือ ร่องรอยโค้ดที่ไม่ดีภายในคลาส และร่องรอยโค้ดที่ไม่ดีระหว่างคลาส โดยในแต่ละกลุ่มนั้นจะมีรายละเอียดดังนี้

2.1.3.1 ร่องรอยโค้ดที่ไม่ดีภายในคลาส

1) การเขียนคำอธิบาย (Comment) : คำอธิบายที่อยู่ในบริเวณที่มีร่องรอยโค้ดที่ไม่ดียังคงเหลืออยู่ภายหลังจากร่องรอยโค้ดในส่วนนั้นได้ถูกกำจัดไปหรือเปลี่ยนแปลงไปด้วยการรีแฟคทอริง การเขียนอธิบายนั้นก็ไม่จำเป็นอีกต่อไป นักพัฒนาซอฟต์แวร์ควรจะทำการลบคำอธิบายออกไป เพื่อไม่ให้เกิดความสับสนเมื่อจำเป็นต้องกลับมาอ่านคำอธิบายนั้นอีกในอนาคต

2) มีโค้ดซ้ำ (Duplicate Code) : เมื่อพบว่าคลาส 1 คลาส มีโค้ดที่ซ้ำกัน ส่วนใหญ่เกิดจากการคัดลอกโค้ดจากที่อื่นมาใช้ ซึ่งโค้ดที่ซ้ำกันจะมีปัญหาต่อการบำรุงรักษาซอฟต์แวร์ในอนาคต

3) เมทธอดยาวเกินไป (Long Method) : เมทธอดที่ทำงานมาก ๆ หรือมีการทำงานที่มากกว่า 1 หน้าที่จะทำให้เกิดความซับซ้อนยากต่อการทดสอบและการบำรุงรักษา

4) สวิทซ์สเตทเมนท์ (Switch Statement) : เป็นลักษณะของการมีจำนวนของเส้นทางการดำเนินการที่เป็นไปได้หลายเส้นทางจากตัวแปรตัวเดียว ซึ่งมีแนวโน้มที่จะทำให้เกิดการทำงานที่ซ้ำกัน

5) มีจำนวนพารามิเตอร์มากเกินไป (Too many Parameters) : จำนวนพารามิเตอร์ที่ส่งเข้ามายังเมทธอดเป็นจำนวนมาก จะทำให้เมทธอดนั้นยากต่อการทำความเข้าใจ และแสดงให้เห็นว่าเมทธอดนั้นจะมีการทำงานที่ซับซ้อน

6) มีเงื่อนไขซับซ้อน (Conditional Complexity) : โค้ดที่มีส่วนการตรวจสอบเงื่อนไขจำนวนมากหรือมีความซับซ้อนจะมีแนวโน้มที่ทำให้เกิดการแก้ไขเปลี่ยนแปลงอยู่บ่อยมาก

7) คลาสที่มีการทำงานมาก (Large Class) : คลาสที่มีหน้าที่การทำงานและประกอบด้วยอ็อบเจกต์จำนวนมากจะส่งผลให้ยากต่อการอ่านโค้ด และยากต่อการทำความเข้าใจ ซึ่งจะทำให้เกิดปัญหาตามมา วิธีการพิจารณาว่าคลาสมีการทำงานเยอะหรือไม่ ให้พิจารณาว่าคลาสมีหน้าที่การทำงานมากกว่าหนึ่งหน้าที่หรือไม่ และสามารถที่จะแยกหน้าที่การทำงานออกไปเป็นคลาสอื่น ๆ ได้หรือไม่

8) ชื่อไม่สื่อความหมาย (Uncommunicative Name) : ชื่อของคลาส ชื่อเมทธอดและตัวแปรต่าง ๆ ไม่สามารถบ่งบอกหน้าที่การทำงาน หรือวัตถุประสงค์ที่ต้องการใช้ได้ หรือชื่อที่กำหนดทำให้เกิดความเข้าใจที่คลาดเคลื่อนไปจากเดิม

9) โค้ดที่ไม่มีบทบาท (Dead Code) : โค้ดที่ไม่เคยถูกเรียกใช้งานเลยในโปรแกรมจำเป็นต้องถูกลบออก เพื่อป้องกันความสับสน และปัญหาในการบำรุงรักษา

10) การคาดเดาคุณลักษณะต่าง ๆ ไว้ล่วงหน้า (Speculative Generality) : เป็นลักษณะของการสร้างซับคลาส (Sub-class) หรือมีการสร้างซุปเปอร์คลาส (Super-class) เอาไว้ก่อนและสุดท้ายไม่ได้ถูกใช้ ซึ่งมันจะทำให้โปรแกรมทำงานช้าลง

11) ไลบรารีคลาสที่ไม่สมบูรณ์ (Incomplete Library Class) : การที่มีไลบรารีคลาสที่ไม่สมบรูณ์ในกรณีนี้อาจจะเกิดจากในไลบรารีนั้นไม่มีคุณลักษณะ (Features) ตามที่นักพัฒนาต้องการใช้ เมื่อซอฟต์แวร์เรียกไลบรารีคลาสมาใช้ทำให้การทำความเข้าใจคลาสเพื่อนำกลับมาใช้ใหม่ทำได้ยากขึ้น

2.1.3.2 ร่องรอยโค้ดที่ไม่ดีระหว่างคลาส

1) เกิดการปรับเปลี่ยน (Shortgun Surgery) : เมื่อมีการเพิ่มหน้าที่การทำงานเข้าไปอีกหนึ่งหน้าที่จะต้องทำการแก้ไขในคลาสอื่นหลาย ๆ คลาส

2) การเปลี่ยนแปลงที่แตกต่าง (Divergent Changes) : มีลักษณะตรงข้ามกับ Shortgun Surgery เป็นลักษณะของการเปลี่ยนแปลงหลาย ๆ อย่างที่เกิดขึ้นในคลาสเดียวกัน

3) ใช้ข้อมูลต่าง ๆ จาก Class อื่น (Feature Envy) : เป็นลักษณะของเมทธอดที่มีการเรียกใช้ข้อมูลต่าง ๆ จากคลาสอื่น แทนที่จะใช้ข้อมูลในคลาสตัวเอง

4) ข้อมูลที่มีขนาดใหญ่ (Data Clumps) : เป็นลักษณะที่มีการใช้กลุ่มข้อมูลชุดเดียวกันหลาย ๆ ที่ หรือเป็นการประกาศตัวแปร หรือชุดของตัวแปรที่มีการสร้างขึ้นมาบ่อย ๆ และใช้ร่วมกันบ่อย ๆ ซึ่งกลุ่มของข้อมูลในที่นี้ประกอบด้วยคุณลักษณะภายในคลาส หรือพารามิเตอร์ในเมทธอด

5) การครอบงำแบบเดิม (Primitive Obsession) : เป็นลักษณะของการเก็บข้อมูลสำหรับการใช้งานพื้นฐานไว้สำหรับให้คลาสอื่น ๆ นำไปใช้งาน และมีการเพิ่มตัวแปรอื่นที่มีการใช้ในลักษณะเดียวกันลงไปเรื่อย ๆ ทำให้กลายเป็นคลาสที่มีขนาดใหญ่และมีความซับซ้อนมาก

6) การสืบทอดแบบคู่ขนาน (Parallel Inheritance Hierarchies) : มีการสร้างซับคลาสที่มีความเฉพาะเจาะจง จะทำให้มีความจำเป็นที่จะต้องสร้างซับคลาสของคลาสอื่นด้วยเพื่อให้ตรงกับการเปลี่ยนแปลง

7) ตัวแปรที่มีการใช้งานชั่วคราว (Temporary Field) : ตัวแปรต่าง ๆ ในคลาสอาจจะไม่ถูกใช้งาน หรือไม่มีประโยชน์ มีการใช้งานตัวแปรชั่วคราว นักพัฒนาซอฟต์แวร์ควรตรวจสอบว่าตัวแปรถูกใช้งานจริง ๆ หรือไม่ ถ้าไม่มีการใช้งานควรจะลบออก

8) มีการส่งข้อความต่อ ๆ กัน (Message Chain) : เป็นการส่งคำขอสำหรับการใช้อ็อบเจกต์ในลักษณะมีการส่งต่อ ๆ กันแบบลูกโซ่ หากอ็อบเจกต์ที่เรียกใช้มีการเปลี่ยนแปลงจะทำให้เกิดความยุ่งยากต่อการแก้ไขในภายหลัง ควรพิจารณาว่าควรจะกำจัดออกตรงไหน โดยที่ไม่ให้ค่าเปลี่ยน

9) ไม่มีการสืบทอด (Refused Bequest) : มีคลาสลำดับชั้นไม่ถูกต้องเมื่อคลาสลูกได้รับการถ่ายทอดคุณสมบัติจากคลาสแม่มาทั้งหมด ซึ่งจริง ๆ แล้วคลาสลูกไม่ต้องการใช้ข้อมูลจากคลาสแม่

10) เป็นตัวกลาง (Middle Man) : คลาสที่ทำหน้าที่เป็นตัวกลางในการติดต่อระหว่างอ็อบเจกต์ เมื่อมีการเรียกใช้งานจะเรียกผ่านตัวกลางทุกครั้งซึ่งอาจจะทำให้คลาสที่ถูกเรียกใช้เป็นตัวกลางมีการทำงานที่หนักเกินความจำเป็นทำให้กิดความซับซ้อนในการทำงาน ควรจะทำการเรียกใช้งานคลาสที่ต้องการโดยตรง

11) ความใกล้ชิดที่ไม่เหมาะสม (Inappropriate Intimacy) : เป็นการเรียกใช้ข้อมูลจากคลาสอื่นโดยไม่จำเป็น หรือมากเกินความจำเป็น

2.1.4 รีแฟคทอริง (Refactoring)

ในงานวิศวกรรมซอฟต์แวร์นั้น รีแฟคทอริง หมายถึง การเปลี่ยนแปลงโครงสร้างภายในของซอฟต์แวร์ โดยไม่เปลี่ยนแปลงการทำงานเดิมของซอฟต์แวร์ โดยมีวัตถุประสงค์เพื่อทำให้ซอร์สโค้ดง่ายต่อการทำความเข้าใจ ซึ่งจะเพิ่มความสามารถในการบำรุงรักษาซอฟต์แวร์ (Fowler and Beck, 1999) หรือกล่าวอีกนัยหนึ่งก็ คือ การรีแฟคทอริงเป็นการกลับมาปรับปรุงโค้ดที่เคยเขียนไปแล้ว และทำงานได้อย่างถูกต้องแล้ว ให้โค้ดสั้นลง กระชับมากขึ้น มีคุณภาพมากขึ้น อ่านได้ง่ายมากขึ้น (Stamelos, *et al*., 2002; McIntosh, *et al.*, 2014; Soares, *et al.*, 2013 และ Mens and Tourwe, 2004) ลดความซับซ้อนของโค้ดเพื่อให้ง่ายต่อการเข้าใจ (Wu and Lin, 2001) ตัวอย่างของการทำรีแฟคทอริง ได้แก่ การลดจำนวนบรรทัดของโค้ด การเพิ่มซุปเปอร์คลาส (Add Super-class) การเพิ่มอินเทอเฟสคลาส (Add Interface Class)

ในกรณีที่มีการทำงานเป็นทีมหรืองานที่ไม่ได้ทำอย่างต่อเนื่อง ทำให้ยากที่จะเข้าใจซอร์สโค้ดที่เกิดขึ้นจากนักพัฒนาหลายคน การทำรีแฟคทอริงจะช่วยในการปรับปรุงการออกแบบให้ง่ายต่อการทำความเข้าใจโค้ดและง่ายต่อการแก้ไขในอนาคต โค้ดที่ได้จะมีความเป็นระเบียบเรียบร้อยจะทำให้นักพัฒนาเข้าใจโค้ดได้ง่ายขึ้น นอกจากนี้การทำรีแฟคทอริงจะช่วยให้นักพัฒนาซอฟต์แวร์ค้นหาข้อผิดพลาดที่เกิดขึ้นได้ง่าย เพราะโค้ดที่ซ้ำกันจะมีน้อยลงช่วยให้โปรแกรมทำงานได้ดีขึ้น (Fagan, 1999) ซึ่งเป็นการทำให้คุณภาพของโค้ดดีขึ้นโดยไม่เปลี่ยนแปลงการทำงานเดิมของซอฟต์แวร์ Fowler and Beck (1999) ได้ระบุให้เห็นถึงประโยชน์ของการทำรีแฟคทอริงไว้ดังนี้

1) รีแฟคทอริงจะทำให้แก้ไขจุดบกพร่องได้ง่าย ทำให้จำนวนความผิดพลาดในซอฟต์แวร์ลดน้อยลง เนื่องจากซอร์สโค้ดที่อ่านง่ายขึ้น

2) การลดขั้นตอนการทำงานในโค้ดให้สั้น หรือกระชับมากขึ้น จะมีแนวโน้มให้ระบบซอฟต์แวร์ทำงานได้เร็วมากขึ้น

3) รีแฟคทอริงจะเพิ่มความยืดหยุ่นของโปรแกรมและปรับปรุงการออกแบบให้ดีขึ้น ดังนั้นจึงทำให้นักพัฒนาซอฟต์แวร์สามารถเพิ่มโค้ดเข้าไปในระบบ หรือการรวมโค้ดเข้าด้วยกันในอนาคตทำได้ง่ายขึ้น

โดยทั่วไปเราสามารถทำรีแฟคทอริงได้ตลอดเวลาในระหว่างการพัฒนาซอฟต์แวร์ แต่อาจจะทำรีแฟคทอริงในกรณีเฉพาะต่อไปนี้

1) เมื่อพบว่าโค้ดหรือโครงสร้างจะต้องมีการเปลี่ยนในอนาคต : ในกรณีนี้จะทำรีแฟคทอริงก่อนเพื่อให้โค้ดง่ายต่อการเพิ่มหรือขยาย หากต้องการเพิ่มฟังก์ชันให้พิจารณาโค้ดภายในที่มีอยู่ว่าเหมาะสมที่จะทำรีแฟคทอริงหรือไม่ หากจำเป็นจะต้องมีการทำรีแฟคทอริง ก็ควรจะทำการรีแฟคทอริงก่อน แล้วค่อยเพิ่มฟังก์ชันเข้าไป

2) หลังการทดสอบย่อย : ทุกครั้งที่มีการทดสอบย่อยควรจะมีการพิจารณาเพื่อทำรีแฟคทอริง

3) พบร่องรอยโค้ดที่ไม่ดี : ร่องรอยโค้ดที่ไม่ดีอาจจะก่อปัญหากับระบบซอฟต์แวร์ในอนาคต การทำรีแฟคทอริงจะช่วยลดหรือกำจัดร่องรอยโค้ดที่ไม่ดี

สำหรับกระบวนการทำรีแฟคทอริงนั้น Fowler and Beck (1999) ได้ระบุขั้นตอนในการทำรีแฟคทอริงไว้ดังนี้

1) ระบุส่วนที่ต้องการจะทำรีแฟคทอริง : นักพัฒนาซอฟต์แวร์ควรจะระบุส่วนของโค้ดที่ต้องการทำรีแฟคทอริง โดยทั่วไปนักพัฒนาจะมองหาส่วนที่มีร่องรอยโค้ดที่ไม่ดี

2) เลือกวิธีการรีแฟคทอริงให้เหมาะสม : หลังจากระบุส่วนที่ต้องการทำรีแฟคทอริงได้เรียบร้อย นักพัฒนาจะต้องพิจารณาหาวิธีการทำรีแฟคทอริงที่เหมาะสม และช่วยกำจัดปัญหาที่อยู่ในส่วนนั้น

3) ประเมินผลว่ามีอะไรดีขึ้นจากเดิม : นอกจากมีการทดสอบการทำงานของฟังก์ชันหลังจากทำรีแฟคทอริง นักพัฒนาซอฟต์แวร์จะต้องประเมินถึงข้อดี หรือสิ่งที่ปรับปรุงขึ้นมาจากวิธีรีแฟคทอริงที่เลือกใช้ว่าทำให้โค้ดมีคุณภาพดีขึ้นกว่าเดิมหรือไม่

4) ทำการปรับปรุงเอกสาร : การทำรีแฟคทอริงจะทำให้โค้ดเกิดการปลี่ยนแปลงจึงมีความจำเป็นที่จะต้องปรับปรุงเอกสารที่เกี่ยวข้องกับโค้ดให้เป็นปัจจุบัน

5) ทดสอบการทำงานของระบบ : เมื่อทำการรีแฟคทอริงเสร็จสิ้น นักพัฒนาซอฟต์แวร์จะต้องมีการทดสอบดูว่าระบบซอฟต์แวร์หรือฟังก์ชันยังทำงานได้เหมือนเดิมผลลัพธ์ที่ได้ยังเหมือนก่อนที่จะทำรีแฟคทอริง ดังนั้นเอกสารต่าง ๆ ที่เกี่ยวกับโค้ด หรือคำอธิบายต่าง ๆ ภายในโค้ดจำเป็นต้องมีการปรับปรุงให้เป็นปัจจุบัน

ในปัจจุบันนี้มีวิธีการรีแฟคทอริงอยู่เป็นจำนวนมาก ซึ่งการเลือกว่าจะทำรีแฟคทอริงในจุดไหน การเลือกประเภทวิธีการ รวมไปถึงการระบุโครงสร้างภายในที่ควรทำการรีแฟคทอริงสามารถทำได้จากการพิจารณาโดยใช้วิจารณญาณส่วนตัว และพิจารณาจากร่องรอยโค้ดที่ไม่ดี ซึ่งวิธีการเหล่านี้เกิดจากประสบการณ์ของนักพัฒนาซอฟต์แวร์ ทฤษฎีทางด้านการพัฒนาซอฟต์แวร์ และงานวิจัยในที่นี้ผู้วิจัยขอยกตัวอย่างวิธีการรีแฟคทอริงที่ได้รับความนิยมบางวิธี เช่น

1) การเพิ่มตัวแปร ฟังก์ชัน คลาส (Add Variable, Add Function, Add Class)
2) การเอาตัวแปร ฟังก์ชัน คลาส ที่ไม่ได้ใช้ออก (Remove Variable, Remove Function, Remove Classes)
3) การเปลี่ยนชื่อตัวแปร (Change Name)
4) การสร้างคลาสแม่ (Create Super-class)
5) การลดจำนวนพารามิเตอร์ (Reduce number of parameters)

```
1    private void printCard() {
2            printBanner();
3            //print detail
4            System.out.println(“Name: “+name”);
5            System.out.println(“Address: “+detail”);
6    }
```

รูปที่ 2.2 ตัวอย่างของโค้ดที่มีการใช้เมทธอดร่วมกัน

```
1    private void printCard() {
2            printBanner();
3            printDetail();
4    }
5    private void printDetail(){
6            System.out.println(“Name: “+name”);
7            System.out.println(“Address: “+detail”);
8    }
     }
```

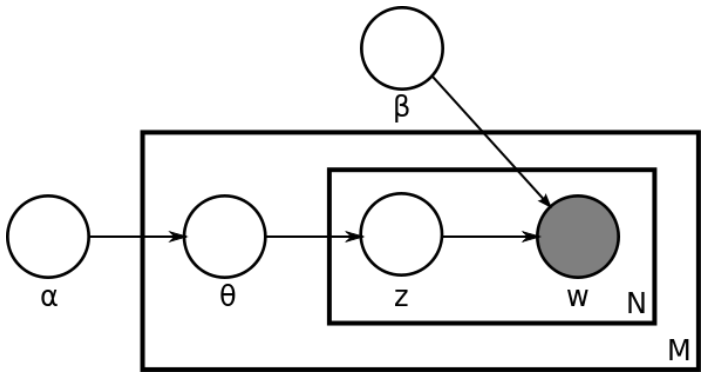
รูปที่ 2.3 วิธีการรีแฟคทอริงที่เรียกว่า Extract Method

รูปที่ 2.2 และ 2.3 แสดงตัวอย่างขั้นตอนการทำรีแฟคทอริง จากโค้ดในรูปที่ 2.2 จะเห็นได้ว่าโค้ดในบรรทัดที่ 4 และ 5 มีการเรียกใช้เมทธอด System.out.println() ในการแสดงรายละเอียดของชื่อและที่อยู่ออกทางหน้าจอคอมพิวเตอร์ ในกรณีนี้ ชื่อและที่อยู่ จัดเป็นรายละเอียดของคลาสที่คล้ายคลึงกัน ควรแยกเอาการทำงานในส่วนนี้ไว้เป็นอีกเมทธอดหนึ่ง เพื่อให้ง่ายในการแก้ไขและบำรุงรักษาในอนาคต ในรูปที่ 2.3 จะใช้วิธีการรีแฟคทอริงที่เรียกว่า Extract Method โดยการสร้างเมทธอดใหม่ที่ชื่อว่า printDetail() และมีการทำงานภายในเมทธอด

2.1.5 อัลกอริทึม Latent Dirichlet Allocation (LDA)

Latent Dirichlet Allocation (LDA) เป็นอัลกอริทึมที่ใช้ในการสร้างแบบจำลองหัวข้อ ซึ่งเป็นแบบจำลองความน่าจะเป็นสำหรับข้อมูลที่ไม่ต่อเนื่อง เช่น คลังข้อมูล (Corpus) ซึ่งจะทำการจัดกลุ่มของหัวข้อในเอกสารที่เกี่ยวข้องกันให้มาอยู่ในกลุ่มเดียวกัน โดยคำนวณคำที่ปรากฏในเอกสารโดยใช้ค่าความน่าจะเป็น ซึ่งข้อจำกัดของวิธีการนี้ คือ จะแสดงเฉพาะข้อมูลที่มีความเกี่ยวข้องกันเท่านั้น (Blei, *et al.*, 2003) โดยจะมีหลักการทำงานดังนี้

1) อัลกอริทึมจะทำการสุ่มเลือกการกระจายในหัวข้อต่างๆ
2) สำหรับแต่ละคำในเอกสาร
   - อัลกอริทึมจะทำการสุ่มเลือกหัวข้อจากการกระจายผ่านหัวข้อต่างๆ
   - อัลกอริทึมจะทำการสุ่มเลือกคำจากหัวข้อที่เกี่ยวข้อง (การกระจายในคำศัพท์ต่าง ๆ)



รูปที่ 2.4 แบบจำลองของ LDA

จากรูปที่ 2.4 แสดงแบบจำลองของ LDA ซึ่งจะเป็นลักษณะของข้อมูลที่มีการพึ่งพากันระหว่างตัวแปรต่าง ๆ โดยกล่องสี่เหลี่ยมด้านนอกจะเป็นตัวแทนของเอกสาร และกล่องสี่เหลี่ยม

ด้านในจะเป็นตัวเลือกและคำพูดที่เกิดซ้ำกันภายในเอกสาร M หมายถึง จำนวนเอกสาร และ N เป็นจำนวนคำในเอกสาร โดยที่

α : เป็นพารามิเตอร์ของการสุ่มที่มีการแฝงตัว (Dirichlet) ก่อนการกระจายหัวข้อของแต่ละเอกสาร

β : เป็นพารามิเตอร์ของการสุ่มที่มีการแฝงตัวก่อนการกระจายคำของแต่ละหัวข้อ

θ_m : การกระจายหัวข้อสำหรับเอกสาร m

φ_{k} : การกระจายคำสำหรับหัวข้อ k

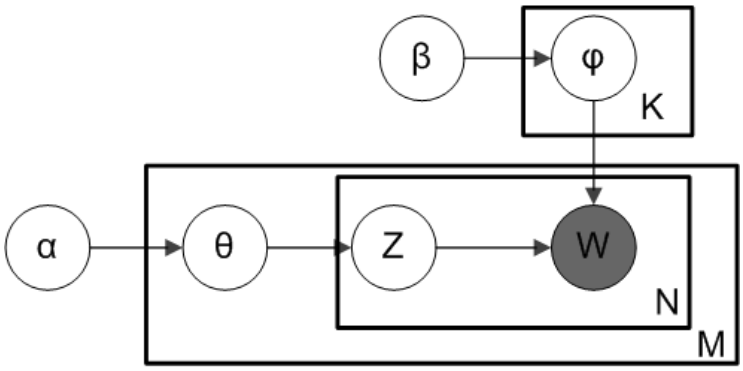
z_{mn} : เป็นหัวข้อสำหรับคำ n ในเอกสาร m

w_{mn} : เป็นคำเฉพาะเจาะจง

เอกสารจะแสดงในรูปแบบการสุ่มที่มีการแฝงตัวอยู่ในหัวข้อต่าง ๆ ซึ่งแต่ละหัวข้อจะมีการอธิบายลักษณะของหัวข้อโดยผ่านการกระจายของคำ LDA จะมีการสร้างเนื้อหาต่อไปนี้สำหรับคลังข้อมูล D ประกอบด้วยเอกสาร M และความยาวของแต่ละเอกสาร คือ N_i สำหรับขั้นตอนการทำงานของ LDA มีดังต่อไปนี้

1) เลือก $\theta_{\mathrm{i}} \sim \mathrm{Dir}(\alpha)$ ที่ $\mathrm{i} \in \{1,\ldots,M\}$ และ $\sim \mathrm{Dir}(\alpha)$ คือการกระจายการสุ่มที่มีการแฝงตัวที่มีพารามิเตอร์ α ซึ่งจะมีการกระจายตัวอยู่น้อย
2) เลือก $\varphi_{\mathrm{k}} \sim \mathrm{Dir}(\beta)$ ที่ $k \in \{1,\ldots, K\}$ และ β จะมีการกระจายตัวอยู่น้อย
3) สำหรับแต่ละตำแหน่งของคำ i, j ที่ $\mathrm{j} \in \{1,\ldots,N_i\}$ และ $\mathrm{i} \in \{1,\ldots,M\}$
   - เลือกหัวข้อ $\mathrm{z}_{\mathrm{i,j}} \sim \mathrm{Multinomial}(\theta_{\mathrm{i}})$
   - เลือกคำ $\mathrm{w}_{\mathrm{ij}} \sim \mathrm{Multinomial}(\varphi_{\mathrm{z,i,j}})$

ขั้นตอนที่อธิบายมาจะเป็นรูปแบบของการกระจายพหุนาม (Multinomial) กับการทดลองเพียงครั้งเดียว ซึ่งจะเรียกว่า การแจกแจงแบบแบ่งหมวดหมู่ ความยาว N_i จะถือว่าเป็นอิสระจากตัวแปรที่ใช้ในการสร้างข้อมูลทั้งหมด (w และ z) ตำแหน่งของคำ มักจะลดลงเช่นเดียวกับรูปที่ได้แสดงไว้ โดย w_{ij} เป็นเพียงตัวแปรสังเกตและตัวแปรอื่น ๆ ที่มีตัวแปรแฝงตามที่ได้กำหนดไว้ในเอกสาร Dirichlet ก่อนที่จะนำไปใช้ผ่านคำที่มีการกระจายในหัวข้อต่าง ๆ จากลักษณะดังกล่าวจะเป็นลักษณะของหัวข้อที่มุ่งเน้นไปยังชุดเล็ก ๆ ของคำ



รูปที่ 2.5 แบบจำลองของ LDA ที่ใช้กันอยู่ในปัจจุบัน

จากรูปที่ 2.5 เป็นรูปแบบของ LDA ที่มีการนำไปใช้งานกันปัจจุบันจะมีกล่อง K เพิ่มเข้าไปทางด้านขวาโดยที่ K หมายถึง จำนวนของหัวข้อและ $\varphi_1,\ldots,\ \varphi_{\mathrm{k}}$ มีการจัดเก็บแบบเวคเตอร์แบบสองมิติทำหน้าที่ในการเก็บพารามิเตอร์ของคำที่มีการกระจายอยู่ในหัวข้อ (V คือ จำนวนคำในคำศัพท์ต่าง ๆ) สูตรที่ใช้สำหรับการกระจายตัวร่วม (ของตัวแปรซ่อนอยู่และตัวแปรที่สังเกต) คือ

$$p(\beta_{1:K},\ \theta_{1:D},\ z_{1:D}, w_{1:D}) \quad = \prod_{i=1}^{K} p(\beta i) \prod_{d=1}^{D} p(\theta d) \prod_{n=1}^{N} p(z_{d,n}|\theta_d) p(w_{d,n}|\beta_{1:K}, z_{d,n})$$

ตัวแปรต่าง ๆ ที่เกิดขึ้นภายในอัลกอลิทึม LDA มีรายละเอียดดังนี้

$\beta_{1:\mathrm{K}}$: เป็นหัวข้อที่แต่ละ β_k (การแจกแจงคำศัพท์)

θ_{d} : เป็นสัดส่วนหัวข้อสำหรับเอกสาร d

$\theta_{\mathrm{d,k}}$: เป็นสัดส่วนหัวข้อสำหรับหัวข้อ k ในเอกสาร d

$\mathbf{z}_{\mathrm{d}}$: การกำหนดหัวข้อสำหรับเอกสาร d

$\mathbf{z}_{\mathrm{d,n}}$: การกำหนดหัวข้อสำหรับคำ n ในเอกสาร d

$\mathbf{w}_{\mathrm{d}}$: เป็นคำที่ได้จากเอกสาร d

โดย LDA ได้รับความนิยมในการนำมาใช้กันอย่างกว้างขวางในกระบวนการจัดกลุ่มของข้อมูลในงานวิจัยทางด้านวิศวกรรมซอฟต์แวร์ (Maskeri, *et al.*, 2008; Lukins, *et al.*, 2010 และ Andrzejewski, *et al.*, 2011) และการกรองข้อมูลแบบพึ่งพาผู้ใช้ร่วม (Collaborative Filtering)

ในงานวิจัยนี้ได้นำอัลกอริทึม LDA มาช่วยในการจัดกลุ่มประเภทของร่องรอยโค้ดที่ไม่ดี โดยนำข้อเสนอแนะจำนวนมากของผู้ตรวจทานโค้ดที่ได้จากกระบวนการตรวจทานโค้ดในโครงการโอเพนซอร์สมาผ่านกระบวนการ LDA เพื่อนำมาใช้ในการจำแนกคำแนะนำของผู้ตรวจทาน

โค้ดที่เกี่ยวกับร่องรอยโค้ดที่ไม่ดี มาทำการแยกประเภทและค้นหาคำที่สื่อถึงร่องรอยโค้ดที่ไม่ดีที่แฝงอยู่ภายในประโยคต่าง ๆ ของคำแนะนำ

2.2 เทคโนโลยีที่เกี่ยวข้อง

เทคโนโลยีที่เกี่ยวข้องในงานวิจัย ได้แก่ กิต (Git) เกอริต (Gerrit) เวิร์ดเน็ต (WordNet) และโปรแกรมอาร์ (R) ซึ่งเป็นเทคโนโลยีที่นำเข้ามาช่วยดำเนินการเริ่มตั้งแต่การรวบรวมและการประมวลผลข้อมูล

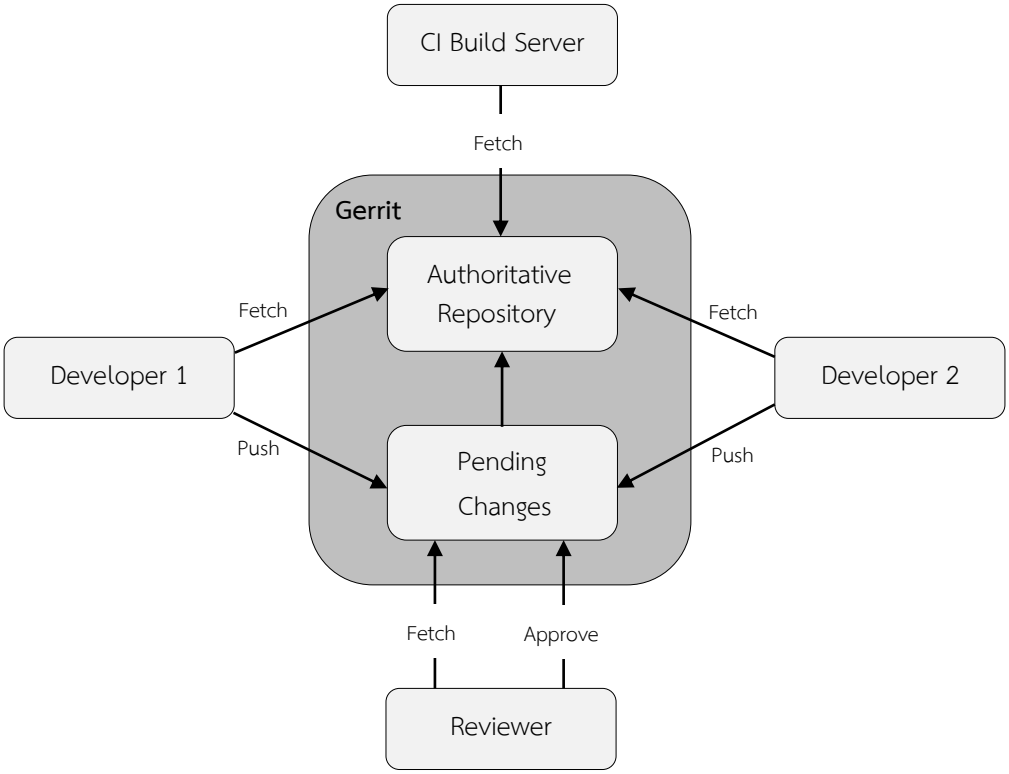
2.2.1 กิต (Git)

กิต เป็นระบบซอฟต์แวร์ที่ช่วยควบคุมการเปลี่ยนแปลงของซอร์สโค้ด (Version Control) และไฟล์ทั่วไป ซึ่งเป็นระบบที่มีหน้าที่ในการจัดเก็บประวัติการเปลี่ยนแปลงของข้อมูลในไฟล์ข้อมูล ทำให้เราสามารถย้อนการกระทำใด ๆ ได้เสมอ เช่น การแก้ไขไฟล์ทำให้เกิดความเสียหายหรือเกิดข้อผิดพลาด นักพัฒนาสามารถย้อนกลับไปใช้ไฟล์ในเวอร์ชันก่อนหน้าได้ หรือแม้แต่ดูว่าใครเป็นผู้แก้ไขไฟล์ ระบบกิตจะช่วยให้นักพัฒนาที่ทำงานเป็นทีมสามารถแก้ไขโค้ดไปพร้อม ๆ กันได้ โดยไม่จำเป็นจะต้องรอให้อีกคนทำงานเสร็จ ระบบกิตยังทำหน้าที่รวมโค้ด (Merge) จากคนสองคนให้อย่างอัตโนมัติ ในกรณีที่กิตไม่สามารถตัดสินใจไม่ได้ว่าควรจะรวมโค้ดยังไง ก็จะให้ตัวผู้แก้ไขตัดสินใจเองว่าจะนำโค้ดไปรวมกันอยู่ในตำแหน่งใด ระบบกิตเป็นเครื่องมือที่ช่วยให้การพัฒนาซอฟต์แวร์เป็นทีมมีประสิทธิภาพ (Brose, 1965)

2.2.2 เกอริต (Gerrit)

เกอริต เป็นระบบที่ช่วยอำนวยความสะดวกในเรื่องการตรวจทานโค้ดให้กับกลุ่มนักพัฒนาซอฟต์แวร์ในกระบวนการการตรวจทานโค้ด ซึ่งจะมีการทำงานร่วมกับระบบจัดการกิต ทำให้นักพัฒนาสามารถใช้งานเกอริตผ่านโปรโตคอล Hypertext Transfer Protocol (HTTP) ในการตรวจสอบการเปลี่ยนแปลงของซอร์สโค้ด และเป็นพื้นที่ในการแสดงความคิดเห็นเกี่ยวกับการปรับปรุงแก้ไขซอร์สโค้ดเพื่อให้บุคคลในทีมพัฒนาสามารถสนทนาแลกเปลี่ยนความรู้กันได้โดยไม่จำเป็นจะต้องอยู่ในสถานที่เดียวกัน ลักษณะของผู้ตรวจทานโค้ดในเกอริตจะมี 3 แบบ คือ 1) ได้รับการเชิญจากเจ้าของซอร์สโค้ดให้มาเป็นผู้ตรวจทานโค้ด 2) ได้รับการแต่งตั้งให้เป็นผู้ตรวจทานโค้ดโดยอัตโนมัติตามความเชี่ยวชาญ 3) อาสาสมัครในการเป็นผู้ตรวจทานโค้ดโดยการส่งอีเมลไปยังเจ้าของซอร์สโค้ด

เพื่อแจ้งความประสงค์ที่ต้องการจะตรวจทานโค้ดชุดนั้น ในทำนองเดียวกันนักพัฒนาท่านอื่น ๆ ที่ไม่ได้มีส่วนเกี่ยวข้องกับซอฟต์แวร์ชุดนั้นก็สามารถที่จะตรวจสอบดูการเปลี่ยนแปลงของซอร์สโค้ดและสามารถให้ข้อเสนอแนะในการการปรับปรุงซอร์สโค้ดได้เช่นกัน ทุก ๆ ครั้งที่ซอร์สโค้ดมีการเปลี่ยนแปลง จะต้องได้รับการยืนยันก่อนที่จะมีการอนุมัติเพื่อนำไปเก็บในโค้ดฉบับสมบูรณ์ (Code Base) การเปลี่ยนแปลงจะถูกอัปโหลดไปยังเกอริต แต่ไม่ได้กลายเป็นส่วนหนึ่งของโครงการจนกว่าจะได้รับการตรวจสอบและได้รับการยอมรับ ระบบเกอริตเป็นหนึ่งในเครื่องมือที่สนับสนุนกระบวนการพัฒนาซอฟต์แวร์โอเพนซอร์ส เป็นการทำให้ผู้ใช้งานทุกคนได้เห็นถึงความเปลี่ยนแปลงของซอร์สโค้ด ในการส่งแพทช์ (Patches) มาทำการตรวจสอบก่อนที่จะนำไปเก็บเป็นโค้ดฉบับสมบูรณ์เพื่อให้มั่นใจว่าการเปลี่ยนแปลงนั้นสามารถนำไปใช้งานได้จริง ด้วยเหตุนี้เกอริตทำให้ผู้ใช้งานทุกคนมีสิทธิเท่าเทียมกันในการเข้ามามีส่วนร่วม เช่น ในกรณีของการพัฒนาซอฟต์แวร์กรรมสิทธิ์จะไม่มีการเปิดเผยซอร์สโค้ดให้นักพัฒนาคนอื่นได้เห็น จะมีเพียงบุคคลที่ทำการพัฒนาเท่านั้นที่จะเห็นและเข้าใจซอร์สโค้ดชุดดังกล่าว เมื่อถึงเวลาที่จะต้องทำการปรับปรุงอาจจะทำได้ยากถ้าคนกลุ่มนั้นหายไป (McIntosh, *et al.*, 2014) รูปที่ 2.6 แสดงการทำงานของระบบเกอริตในภาพรวมโดยมีรายละเอียดการทำงานดังนี้



รูปที่ 2.6 การทำงานของระบบเกอริต[11]

[11] https://gerrit-review.googlesource.com

ทีมพัฒนาจำเป็นจะต้องมีพื้นที่ที่ใช้สำหรับเก็บข้อมูลที่ได้รับการอนุมัติ (Authoritative Repository) เพื่อเป็นศูนย์รวมให้นักพัฒนาในทีมสามารถเข้าถึงข้อมูลได้ โดยปกติแล้วกิตจะสามารถทำงานได้โดยไม่ต้องอาศัยพื้นที่ในการเก็บข้อมูล แต่ในทางปฏิบัติมักมีพื้นที่ที่ใช้สำหรับเก็บข้อมูลเพื่อทำการสร้างสำเนาของการเปลี่ยนแปลงต่าง ๆ ที่เกิดขึ้นในการพัฒนาซอฟต์แวร์ของแต่ละโครงการ ซึ่งจะเป็นสิ่งที่ทุกคนในทีมพัฒนาสามารถทำการดึงข้อมูล (Fetches) และการผลักข้อมูล (Push) ไปยังตำแหน่งของสร้างเซิร์ฟเวอร์ที่ได้สร้างไว้ (CI Build Server)

เกอริตถูกนำไปใช้แทนพื้นที่ของการเก็บข้อมูลส่วนกลางและมีการเพิ่มเติมในส่วนของการจัดเก็บข้อมูลที่รออนุมัติการเปลี่ยนแปลง นักพัฒนาทุกคนยังทำการดึงข้อมูลที่ได้รับการอนุมัติแต่แทนที่จะทำการส่งข้อมูลกลับไป (Pushing Back) ที่เดิม พวกเขาก็จะส่งข้อมูลไปยังตำแหน่งการเปลี่ยนแปลงที่รอดำเนินการ เพื่อรออนุมัติการเปลี่ยนแปลง เมื่อการเปลี่ยนแปลงชุดนั้นได้รับการอนุมัติจากผู้ตรวจทานเรียบร้อยแล้วก็จะถูกส่งไปจัดเก็บไว้ในข้อมูลที่ได้รับการอนุมัติและกลายเป็นส่วนหนึ่งของโครงการ เช่นเดียวกับการแก้ปัญหาพื้นที่เก็บข้อมูลโฮสติ้ง เกอริตมีรูปแบบการควบคุมการเข้าถึงที่มีประสิทธิภาพ ผู้ใช้งานสามารถจะเข้าถึงพื้นที่เก็บข้อมูลส่วนกลางได้โดยไม่ต้องผ่านกระบวนการตรวจทานโค้ดทั้งหมด สถานะในการตรวจทานโค้ดในระบบเกอริตมี 2 ประเภท คือ

1) Code-Review เป็นสถานะการตรวจทานที่จะถูกนำมาใช้ในการลงคะแนนในเรื่องคุณภาพของการนำไปใช้งาน รูปแบบของโค้ด ความสอดคล้องของโค้ด และการออกแบบโดยรวมของโค้ดที่เป็นมาตรฐานตามลักษณะของโครงการนั้น ๆ ซึ่งบุคคลที่มีสิทธิในการลงคะแนน คือ ทีมพัฒนาหลัก (Committers) และ ผู้ร่วมพัฒนา (Contributors) ซึ่งเกณฑ์ในการให้คะแนนของสถานะการตรวจทาน มีดังนี้

- -2 โค้ดจะไม่ถูกนำมาตรวจสอบ เป็นเพราะโค้ดชุดนี้ไม่ถูกต้องหรือเกิดข้อผิดพลาดไม่สามารถทำงานได้ ผู้ตรวจทานโค้ดจะไม่นำคำร้องขอนี้มาพิจารณาไม่ว่าจะในกรณีใดก็ตาม
- -1 โค้ดจะไม่ถูกนำไปรวมกับโครงการทันที เป็นเพราะโค้ดไม่ถูกต้องหรืออาจจะมีการทำอะไรที่แตกต่างไปจากรูปแบบของโค้ดในโครงการ แต่ผู้ตรวจทานโค้ดก็ยินดีที่จะให้ข้อเสนอแนะเพราะซอร์สโค้ดชุดนี้อาจจะดีกว่าสิ่งที่อยู่ในโครงการ บ่อยครั้งที่ผู้ร่วมพัฒนาไม่ชอบการเปลี่ยนแปลงของซอร์สโค้ดนี้ แต่จะไม่มีส่วนรับผิดชอบต่อโครงการในระยะยาวและจะไม่ได้พูดถึงการเปลี่ยนแปลงในขั้นสุดท้าย
- 0 ไม่มีคะแนน ผู้ตรวจทานโค้ดไม่ได้พยายามที่จะทำการตรวจสอบโค้ด
- +1 โค้ดดูดี ทีมพัฒนาเห็นชอบที่จะอนุมัติซอร์สโค้ด แต่การอนุมัติจะต้องได้รับความเห็นชอบจากผู้ร่วมพัฒนาด้วย โค้ดมีลักษณะเหมาะสมกับผู้ตรวจสอบรายนี้แต่ผู้ตรวจสอบไม่สามารถให้คะแนน +2 สำหรับหมวดหมู่นี้ได้ บ่อยครั้งที่มีการใช้งานโดยผู้ร่วมพัฒนาให้

ข้อมูลกับโครงการที่สามารถตรวจสอบการเปลี่ยนแปลงและชอบสิ่งที่ทำ แต่ไม่ได้รับการอนุมัติในขั้นสุดท้าย

- +2 โค้ดดูดีและได้รับการอนุมัติ

2) Verified เป็นสถานะที่ได้รับการยืนยันในการตรวจสอบแล้ว หมายถึง ซอร์สโค้ดชุดนั้นได้ผ่านการทดสอบย่อยเรียบร้อยแล้ว ซึ่งเกณฑ์ในการให้คะแนนของสถานะการการยืนยัน มีดังนี้

- -1 ล้มเหลว พยายามจะประมวลผลแล้วแต่มีข้อผิดพลาดในการทดสอบ
- ไม่มีคะแนน ไม่ได้พยายามที่จะดำเนินการตรวจสอบ
- +1 ได้รับการยืนยัน ประมวลผลการทดสอบเรียบร้อยแล้ว

2.2.3 เวิร์ดเน็ต (WordNet)

เวิร์ดเน็ต เป็นซอฟต์แวร์ที่มีฐานข้อมูลคำศัพท์ภาษาอังกฤษขนาดใหญ่ของคำนาม คำกริยา คำคุณศัพท์ และกริยาวิเศษณ์ มีการจัดกลุ่มเป็นชุดของคำพ้องความรู้ความเข้าใจจะเชื่อมโยงกันโดยวิธีการของความสัมพันธ์ของแนวคิดและความหมายคำศัพท์ที่ออกแบบตามทฤษฎีทางภาษาศาสตร์ โดยมีวัตถุประสงค์เพื่อผสมผสานระหว่างพจนานุกรมคำศัพท์ (Dictionary) กับพจนานุกรมอภิธานศัพท์หรือคำพ้องความหมายหรือคำตรงข้าม (Thesaurus) เข้าด้วยกันเป็นเครื่องมือที่มีประโยชน์สำหรับภาษาศาสตร์ คอมพิวเตอร์ และการประมวลผลภาษาธรรมชาติ ซึ่งเป็นซอฟต์แวร์ที่ได้รับความนิยมนำไปใช้ในงานวิจัยที่มีลักษณะเดียวกันกับงานวิจัยนี้ (Moha, *et al.*, 2009; Kuhn, *et al.*, 2007; Ouni, *et al.*, 2015 และ Abebe, *et al.*, 2011)

งานวิจัยครั้งนี้มีการนำเวิร์ดเน็ตมาเป็นตัวช่วยค้นคืนสารสนเทศ ช่วยทำการระบุคำที่มีความหมายสื่อถึงคำหลัก เช่น ในกรณีที่เราต้องการจะค้นหาข้อเสนอแนะของผู้ตรวจทานโค้ดที่มีการให้คำแนะนำเกี่ยวกับร่องรอยโค้ดที่ไม่ดีประเภท Duplicate Code เมื่อค้นหาด้วยคำว่า “Duplicate” ก็จะแสดงเฉพาะข้อเสนอแนะที่มีคำว่า “Duplicate” ออกมาเท่านั้น แต่หากในการค้นหานั้นมีฐานข้อมูลของเวิร์ดเน็ต ในการค้นหาก็จะรู้ว่า “Duplicate” “Copy” “Clone” “Double” และ “Repeatedly” นั้นอ้างถึงร่องรอยโค้ดที่ไม่ดีประเภท Duplicate Code เวิร์ดเน็ตมี 2 ประเภท คือ แบบออนไลน์ สามารถใช้งานผ่านเครือข่ายอินเทอร์เน็ตได้และแบบโปรแกรมสำเร็จรูปซึ่งสามารถใช้งานผ่านเครื่องที่ติดตั้งโปรแกรมได้เลย สามารถรองรับได้หลายระบบปฏิบัติการไม่ว่าจะเป็นระบบปฏิบัติการวินโดวส์ (Windows) ระบบปฏิบัติการยูนิกซ์ (Unix) และระบบปฏิบัติการลีนุกซ์ (Linux) นอกจากนี้ยังมีเวิร์ดเน็ตในรูปแบบของโปรแกรมเสริม (Plug-in) ที่จะเพิ่มความสามารถพิเศษให้กับหลาย ๆ โปรแกรม เช่น โปรแกรมอาร์ (R) โปรแกรมไพธอน (Python) เป็นต้น ประโยชน์ของเวิร์ดเน็ต (Fellbaum, 1998) มีดังนี้

1) สามารถระบุความสัมพันธ์ทางความหมายแบบต่าง ๆ ได้
2) ใช้เป็นแหล่งข้อมูล (Resource) สำหรับงานการประมวลผลภาษาธรรมชาติ (Natural Language Processing) ต่าง ๆ
3) ช่วยแก้ปัญหาเรื่องความกำกวมความหมายของคำ (Word Sense Disambiguation)
4) ช่วยค้นคืนสารสนเทศ (Information retrieval)
5) ช่วยในการสร้างระบบคำถาม-คำตอบ (Question Answering)
6) ช่วยการสรุปใจความสำคัญ (Text Summarization)
7) วิศวกรรมความรู้ (Knowledge Engineering)

2.2.4 โปรแกรมอาร์ (R)

โปรแกรม R เป็นซอฟต์แวร์ที่รวมเอาคุณสมบัติด้านการจัดการข้อมูล การคำนวณและการแสดงผลทางกราฟิกไว้ด้วยกันอย่างดี โปรแกรม R มีคำสั่งสำหรับการวิเคราะห์ข้อมูลทางสถิติ รวมถึงการแสดงผลทางด้านกราฟฟิก และมีความสามารถในการทำเหมืองข้อมูล (Data Mining) ซึ่งได้รับความนิยมใช้กันในกลุ่มของนักวิเคราะห์ข้อมูลและนักวิชาการ โปรแกรม R เป็นโปรแกรมที่สามารถดาวน์โหลดมาใช้ได้ฟรี[12] ซึ่งสามารถทำงานได้บนหลายระบบปฏิบัติการ เช่น วินโดวส์ แมคโอเอส (Mac OS) และลีนุกซ์

ในงานวิจัยนี้ได้มีการนำโปรแกรม R มาช่วยดำเนินการในส่วนของการทำเหมืองข้อความ (Text Mining) โดยการเรียกใช้งาน package tm ในการค้นหาคำที่มีความหมายที่สื่อถึงคำหลักจะมีการเรียกใช้งาน package wordnet และในส่วนของค้นหาประเภทร่องรอยโค้ดที่ไม่ดีเพิ่มเติมจะมีการเรียกใช้งาน package lda รวมไปถึงการนำมาใช้ในการคำนวณหาค่าสถิติและนำเสนอข้อมูลในรูปแบบของกราฟต่าง ๆ

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

2.3 วรรณกรรมที่เกี่ยวข้อง

จากการทบทวนวรรณกรรมที่เกี่ยวข้อง ผู้วิจัยได้ทำการจัดกลุ่มของงานวิจัยที่เกี่ยวข้องออกเป็น 4 กลุ่ม คือ 1) งานวิจัยที่มีการศึกษาเกี่ยวกับข้อเสนอแนะของผู้ตรวจทานโค้ด 2) งานวิจัยที่มีการศึกษาเกี่ยวกับการทำรีแฟคทอริงและร่องรอยโค้ดที่ไม่ดี 3) งานวิจัยที่มีการศึกษาเกี่ยวกับการปรับปรุงคุณภาพของซอฟต์แวร์ในโครงการโอเพนซอร์ส 4) งานวิจัยที่มีการศึกษาเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์ส

1) งานวิจัยที่มีการศึกษาเกี่ยวกับข้อเสนอแนะของผู้ตรวจทานโค้ด มีดังนี้

งานวิจัยของ Tao และคณะ ได้นำเสนอแนวทางที่จะช่วยให้นักพัฒนาเขียนโปรแกรมที่ใช้แก้ไขข้อบกพร่องของซอฟต์แวร์ (Patch) ให้ได้รับการยอมรับโดยผู้ตรวจทานโค้ด และเพื่อหลีกเลี่ยงไม่ให้มีการส่งโค้ดไปตรวจสอบซ้ำอีกครั้ง คณะผู้วิจัยได้ทำการตรวจสอบสาเหตุที่ทำให้โปรแกรมแก้ไขซอฟต์แวร์จะถูกปฏิเสธ โดยคณะผู้วิจัยได้ทำการรวบรวมข้อมูลการปฏิเสธโปรแกรมแก้ไขในโครงการพัฒนาซอฟต์แวร์ Eclipse และ Mozilla คณะผู้วิจัยได้จัดทำแบบสำรวจออนไลน์เพื่อเก็บข้อมูลจากนักพัฒนาจำนวน 246 คน ที่มีส่วนร่วมกับโครงการ Eclipse และ Mozilla และได้ทำการสำรวจเพิ่มเติมจากงานวิจัยที่ผ่านมา มีการรวบรวมรายการที่ครอบคลุมถึงเหตุผลในการปฏิเสธโปรแกรมแก้ไข และค้นหาความแตกต่างระหว่างผู้พัฒนาโปรแกรมแก้ไขและการแสดงความคิดเห็นในแง่ของเกณฑ์การประเมินโปรแกรมแก้ไข (Tao, *et al.*, 2014) ผลลัพธ์ที่ได้จากการสำรวจมีดังนี้

1) วิธีการที่ไม่สมบูรณ์ของการแก้ไขจะเป็นเหตุผลที่ถูกปฏิเสธบ่อยที่สุด
2) ผู้ตรวจทานโค้ดตัดสินใจยากในการยอมรับโปรแกรมแก้ไขที่เกิดขึ้นใหม่ครั้งแรก
3) ผู้ตรวจทานโค้ดจะทำการปฏิเสธโปรแกรมแก้ไขที่มีขนาดใหญ่ เพราะสาเหตุส่วนใหญ่ที่ทำให้โปรแกรมแก้ไขมีขนาดใหญ่ขึ้น คือ มีการเปลี่ยนแปลงที่ไม่จำเป็น
4) ผู้ตรวจทานโค้ดจะพิจารณาถึงปัญหาที่อาจจะเกิดขึ้นได้ดีกว่านักพัฒนา
5) การกำหนดเวลาที่ไม่ดีของการส่งโปรแกรมแก้ไขไปให้ตรวจทาน และการขาดการสื่อสารกับสมาชิกในทีม

งานวิจัยนี้มีความเกี่ยวข้องกับงานวิจัยที่ได้ศึกษา คือ งานที่ผู้วิจัยทำการศึกษาเป็นการนำข้อเสนอแนะที่เกิดขึ้นในกระบวนการตรวจทานโค้ดมาทำการค้นหาข้อเสนอแนะที่มีการแนะนำให้นักพัฒนาทำการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง และทำการแยกประเภทของร่องรอยโค้ดที่ไม่ดี ดังนั้นก่อนจะทำการรวบรวมข้อเสนอแนะจำเป็นต้องมีความเข้าใจถึงกระบวนการ

ตรวจทานโค้ด และศึกษารูปแบบของข้อเสนอแนะที่เกิดจากผู้ตรวจทานโค้ด เพื่อสร้างความเข้าใจและง่ายต่อการค้นหาและรวบรวมข้อเสนอแนะที่เกิดขึ้น

ขณะที่งานวิจัยของ Bosu และคณะ ได้ทำการวิเคราะห์การตรวจทานโค้ด การจำแนกคุณลักษณะของโค้ด การเปลี่ยนแปลง และทำการจำแนกลักษณะการแนะนำของนักพัฒนาที่จะไม่ก่อให้เกิดความบกพร่องในด้านความมั่นคงในโครงการซอฟต์แวร์โอเพนซอร์ส คณะผู้วิจัยได้ทำการวิเคราะห์คำร้องขอในการตรวจทานโค้ดจำนวน 267,046 คำร้องขอจาก 10 โครงการของซอฟต์แวร์โอเพนซอร์ส ผลการค้นหาที่สำคัญของงานวิจัยนี้ (Bosu, *et al.*, 2014) ได้แก่

1) การตรวจทานโค้ดสามารถระบุชนิดของความบกพร่องที่พบได้
2) การเปลี่ยนแปลงโค้ดจะขึ้นอยู่กับประสบการณ์ของผู้ตรวจทาน
3) มีความเป็นไปได้ที่จะเกิดความบกพร่องเพิ่มขึ้นเมื่อจำนวนบรรทัดมีการเปลี่ยนแปลง
4) ไฟล์ที่มีการแก้ไข มีแนวโน้มที่จะพบข้อพกพร่องมากกว่าไฟล์ที่สร้างขึ้นมาใหม่

งานวิจัยนี้มีความเกี่ยวข้องกับงานวิจัยที่กำลังทำการศึกษา คือ ในงานวิจัยนี้ได้ทำการจำแนกลักษณะของโค้ดที่ได้รับการตรวจทานโค้ดในโครงการซอฟต์แวร์โอเพนซอร์ส ผู้วิจัยจึงเล็งเห็นว่าสามารถนำการจัดกลุ่มคำสำคัญจากงานวิจัยนี้มาเป็นแนวทางในการจัดกลุ่มของร่องรอยโค้ดที่ไม่ดีได้

งานวิจัยของ Bosu และคณะ อีกหนึ่งงานวิจัยได้นำเสนอการค้นหาลักษณะของคำแนะนำที่เป็นประโยชน์ต่อตรวจทานโค้ดในบริษัทไมโครซอฟต์ โดยมีขั้นตอนในการศึกษาดังนี้

1) ทำการศึกษาลักษณะของข้อเสนอแนะที่เป็นประโยชน์จากมุมมองของนักพัฒนา
2) สร้างกระบวนการแยกความแตกต่างระหว่างข้อเสนอแนะที่เป็นประโยชน์และไม่มีประโยชน์
3) นำข้อมูลที่ได้จากข้อที่ 1) มาแยกความแตกต่างระหว่างข้อเสนอแนะที่เป็นประโยชน์และไม่มีประโยชน์ แต่ยังคงมีข้อจำกัด คือ มีการตรวจสอบโค้ดเฉพาะในบริษัทไมโครซอฟต์เท่านั้น ซึ่งยังไม่มีการเปรียบเทียบกับโครงการซอฟต์แวร์โอเพนซอร์สทั่วไป

จากงานวิจัยพบว่านักพัฒนาใช้เวลาและความพยายามอย่างมากในการตรวจทานโค้ด คณะผู้วิจัยได้ทำการระบุปัจจัยที่นำไปสู่การตรวจสอบโค้ดที่เป็นประโยชน์ โดยทำการแยกประเภทข้อเสนอแนะออกเป็น 2 ประเภท คือ ข้อเสนอแนะที่เป็นประโยชน์ และข้อเสนอแนะที่ไม่เป็นประโยชน์ (Bosu, *et al.*, 2015)

งานวิจัยนี้มีความเกี่ยวข้องกับงานวิจัยที่กำลังทำการศึกษา คือ มีลักษณะของการจัดประเภทของร่องรอยโค้ดที่ไม่ดีโดยทำการศึกษาจากข้อเสนอแนะของผู้ตรวจทานโค้ดในโครงการโอเพนซอร์ส ซึ่งมีความคล้ายคลึงกับการแยกประเภทของข้อเสนอแนะที่เป็นประโยชน์ และข้อเสนอแนะที่ไม่เป็นประโยชน์ แต่จะมีความแตกต่างกันในเรื่องของวิธีการที่นำมาใช้ทำการแยกประเภท โดยในงานวิจัยนี้มีการนำข้อเสนอแนะมาทำการค้นหาร่องรอยโค้ดที่ไม่ดีที่ซ่อนอยู่ในคำแนะนำและทำการจัดกลุ่มของร่องรอยที่ไม่ดีเหล่านั้นด้วย

งานวิจัยของ Baum และคณะ ได้ทำการศึกษาหาปัจจัยที่มีอิทธิพลต่อการตรวจทานโค้ดในอุตสาหกรรมซอฟต์แวร์ เพื่อให้ได้ข้อมูลในเชิงลึกพวกเขาได้ทำการสัมภาษณ์ผู้เชี่ยวชาญด้านการพัฒนาซอฟต์แวร์จำนวน 24 คน จาก 19 บริษัท งานวิจัยนี้แสดงให้เห็นถึงความสำคัญที่เกี่ยวกับประเด็นทางสังคมและประเด็นทางวัฒนธรรมขององค์กรสำหรับการนำกระบวนการตรวจทานโค้ดไปใช้งาน งานวิจัยนี้ได้ระบุให้เห็นถึงปัจจัยหลายบริบทที่มีอิทธิพลต่อการนำกระบวนการตรวจทานโค้ดมาใช้ คณะผู้วิจัยได้จัดกลุ่มของอิทธิพลดังกล่าวออกเป็น 5 ประเภท ได้แก่ วัฒนธรรมองค์กร ทีมพัฒนา ผลิตภัณฑ์ กระบวนการพัฒนา และเครื่องมือที่นำมาใช้ในการพัฒนา (Baum, *et al.*, 2016)

งานวิจัยนี้มีความเกี่ยวข้องกับงานวิจัยที่กำลังทำการศึกษา เนื่องจากงานวิจัยของ Baum, *et al*. (2016) ได้ทำการหาปัจจัยที่มีอิทธิพลต่อการตรวจทานโค้ดในอุตสาหกรรมซอฟต์แวร์ ทำให้เห็นว่ามีปัจจัยหลายอย่างที่ส่งผลต่อการปรับคุณภาพของซอฟต์แวร์ ซึ่งงานวิจัยที่ผู้วิจัยได้ทำการศึกษาจะมีส่วนเกี่ยวข้องอิทธิพลในด้านของกระบวนการที่ใช้ในการปรับปรุงคุณภาพของซอฟต์แวร์ ด้วยวิธีการทำรีแฟคทอริ่ง เป็นการมุ่งเน้นให้นักพัฒนาเห็นถึงความสำคัญเกี่ยวกับการปรับปรุงคุณภาพของซอฟต์แวร์ให้ดียิ่งขึ้น

จากงานวิจัยข้างต้น จะเห็นได้ว่าคำแนะนำของผู้ตรวจทานโค้ด ได้มีการนำมาศึกษาในหลาย ๆ มุมมอง ทั้งในมุมของการช่วยเหลือในการแก้ปัญหาของนักพัฒนาซอฟต์แวร์ให้สามารถเขียนโปรแกรมแก้ไขให้เร็วขึ้น และทำการจัดประเภทของคำอธิบายในการตรวจทานโค้ด เพื่อให้ง่ายต่อการทำความเข้าใจและมีความระมัดระวังในการพัฒนาซอฟต์แวร์ให้มีคุณภาพ ผู้วิจัยได้เล็งเห็นถึงความจำเป็นในการค้นหาข้อมูลเพื่อทำการจัดประเภทของร่องรอยโค้ดที่ไม่ดีในโครงการโอเพนซอร์สเป็นการเพิ่มหลักฐานเชิงประจักษ์ที่เกี่ยวกับการเพิ่มคุณภาพของซอฟต์แวร์ในโครงการซอฟต์แวร์โอเพนซอร์สให้กับนักวิจัยด้านวิศวกรรมซอฟต์แวร์ได้ตระหนักและให้ความสำคัญกับการนำรีแฟคทอริงไปใช้ในการพัฒนาระบบได้

2) งานวิจัยที่มีการศึกษาเกี่ยวกับการทำรีแฟคทอริงและร่องรอยโค้ดที่ไม่ดี มีดังนี้

งานวิจัยของ Yamashita และคณะ ได้ทำการตรวจสอบว่านักวิจัยและนักพัฒนามีความกังวลหรือให้ความสำคัญเกี่ยวกับร่องรอยโค้ดที่ไม่ดีเหมือนกันหรือไม่เพราะที่ผ่านมามีหลายงานวิจัยได้นำเสนอวิธีการที่ดี ๆ เกี่ยวกับการป้องกันและการกำจัดร่องรอยโค้ดที่ไม่ดี แต่ร่องรอยโค้ดที่ไม่ดีก็ยังส่งผลกระทบที่ก่อให้เกิดความเสียหายอยู่ในปัจจุบัน ผู้วิจัยได้ทำการสำรวจแบบออนไลน์จากนักพัฒนาซอฟต์แวร์มืออาชีพ ทั้งหมดจำนวน 85 คน มีนักพัฒนาจำนวน 73 คนที่ตอบแบบสอบถามเสร็จสมบูรณ์ (อัตราการของการตอบสนอง 86%) มาจาก 29 ประเทศ (เช่น อาร์เจนตินา โครเอเชีย ลัตเวีย ไนจีเรีย โปรตุเกส ฯลฯ) ในแบบสำรวจจะมีทั้งคำถามปลายเปิดและคำถามปลายปิดเป้าหมายเพื่อดูระดับของความเข้าใจเกี่ยวกับร่องรอยโค้ดที่ไม่ดี ดูวิธี แนวคิด และเครื่องมือที่ใช้ในการป้องกันเกี่ยวกับร่องรอยโค้ดที่ไม่ดีที่เหล่านักพัฒนาใช้กันอยู่ในปัจจุบัน งานวิจัยนี้เป็นการสะท้อนให้เห็นว่ามีนักพัฒนาส่วนใหญ่ไม่มีความเข้าใจเกี่ยวกับร่องรอยโค้ดที่ไม่ดี มีนักพัฒนาเพียง 4 % เท่านั้นที่มีความเข้าใจเกี่ยวกับร่องรอยโค้ดที่ไม่ดี มีความสนใจหรือให้ความสำคัญเกี่ยวกับร่องรอยโค้ดที่ไม่ดีอยู่ในระดับปานกลาง สาเหตุหลัก คือ ต้องการที่จะทำให้ซอฟต์แวร์มีคุณภาพ โดยพวกเขาต้องการเครื่องมือที่ช่วยสนับสนุนในการค้นหาร่องรอยโค้ดที่ไม่ดี ที่มีลักษณะการทำงานที่สามารถแสดงผลทันที (Real Time) เมื่อเจอร่องรอยโค้ดที่ไม่ดี จะทำให้เกิดประโยชน์สูงสุดที่จะช่วยให้ระบุพื้นที่ที่มีปัญหาได้ (Yamashita, *et al.*, 2013)

งานวิจัยนี้มีความเกี่ยวข้องกับงานวิจัยที่ได้ศึกษา คือ ในงานวิจัยนี้ได้แสดงให้เห็นว่านักพัฒนาให้ความสำคัญกับการเกิดร่องรอยโค้ดที่ไม่ดีอย่างไร นักพัฒนามีความเข้าใจเกี่ยวกับร่องรอยโค้ดที่ไม่ดีมากน้อยแค่ไหน เป็นการสะท้อนให้เห็นถึงเรื่องคุณภาพและความเสี่ยงในการเกิดข้อบกพร่องซึ่งจะทำให้ยากต่อการบำรุงรักษาซอฟต์แวร์ในอนาคต

งานวิจัยของ Yoshida และคณะ ได้ศึกษาความสัมพันธ์ระหว่างการทำรีแฟคทอริงกับร่องรอยโค้ดที่ไม่ดี เป็นการตรวจสอบว่านักพัฒนาจะมีการเลือกใช้การทำรีแฟคทอริงรูปแบบใดที่เหมาะสมในการแก้ไขร่องรอยโค้ดที่ไม่ดี ซึ่งในงานวิจัยนี้มีวัตถุประสงค์เพื่อสนับสนุนการเขียนโปรแกรมในการหารูปแบบการทำรีแฟคทอริงที่เหมาะสม สามารถช่วยกำจัดร่องรอยโค้ดที่ไม่ดีออกจากระบบ โดยการกำหนดชื่อรูปแบบของรีแฟคทอริงให้สอดคล้องกับประเภทของร่องรอยโค้ดที่ไม่ดี โดยทำการเลือกโครงการซอฟต์แวร์โอเพนซอรส์ที่มีการพัฒนาด้วยภาษาจาวา (Java) มา 3 โครงการ คือ Apache Ant (Ant), ArgoUML (Argo), และ Xerces-J (Xerces) มาทำการตรวจสอบว่ามีความสอดคล้องกับรูปแบบการทำรีแฟคทอริงที่นำไปใช้กับร่องรอยโค้ดที่ไม่ดีหรือไม่ จากผลการตรวจสอบ

พบว่า ในการทำรีแฟคทอริงไม่ค่อยจะนำประเภทของร่องรอยโค้ดที่ไม่ดี มาใช้ในการทำรีแฟคทอริง เนื่องจากรูปแบบของร่องรอยโค้ดที่ไม่ดีไม่สอดคล้องกับการทำรีแฟคทอริง (Yoshida, *et al.*, 2016)

งานวิจัยนี้มีความเกี่ยวข้องกับงานวิจัยที่กำลังทำการศึกษา คือ ในงานวิจัยนี้ได้แสดงให้เห็นถึงความความสัมพันธ์ระหว่างประเภทของร่องรอยโค้ดที่ไม่ดีกับการทำรีแฟคทอริง สามารถนำมาเป็นแนวทางในการศึกษาถึงวิธีการทำรีแฟคทอริงเมื่อเจอร่องรอยโค้ดที่ไม่ดีในแต่ละประเภทควรจะทำการแก้ไขในลักษณะใด

งานวิจัยของ Silva และคณะ ได้ศึกษาถึงแรงจูงใจในการทำรีแฟคทอริงของนักพัฒนาเพื่อหาสาเหตุที่แท้จริงที่เป็นตัวการให้นักพัฒนาตัดสินใจทำรีแฟคทอริง โดยให้นักพัฒนาอธิบายถึงสาเหตุที่ทำให้พวกเขาต้องนำวิธีการทำรีแฟคทอริงมาใช้ในการปรับปรุงซอร์สโค้ด โดยจะนำข้อมูลที่ได้มาทำการวิเคราะห์ผล จากการรวบรวมแรงจูงใจที่ได้จากนักพัฒนาเกี่ยวกับการทำรีแฟคทอริงมีทั้งหมด 44 รายการ สำหรับวิธีการทำรีแฟคทอริงที่รู้จักกันเป็นอย่างดี 12 ชนิด พวกเขาพบว่ากิจกรรมในการทำรีแฟคทอริงส่วนใหญ่จะมีสาเหตุหลักมาจากการเปลี่ยนแปลงความต้องการ (Requirements) และการทำรีแฟคทอริงจะเกิดขึ้นเมื่อพบเจอร่องรอยโค้ดที่ไม่ดี การศึกษาของพวกเขายืนยันว่า การแยกเมธอด (Extract Method) เป็น “สุดยอดเครื่องมือในการทำรีแฟคทอริง” ซึ่งเป็นการทำรีแฟคทอริงที่มีแรงจูงใจมากที่สุด และยังทำให้พวกเขาพบหลักฐานชิ้นสำคัญ คือ เครื่องมือที่ช่วยในการพัฒนา (IDE) ที่นักพัฒนาใช้งานอยู่มีส่วนในการช่วยทำรีแฟคทอริงแบบอัตโนมัติ (Silva, *et al.,* 2016)

งานวิจัยนี้มีความเกี่ยวข้องกับงานวิจัยที่กำลังทำการศึกษา คือ ในงานวิจัยนี้ได้แสดงให้เห็นถึงแรงจูงใจของนักพัฒนาเกี่ยวกับการทำรีแฟคทอริง ทำให้เห็นว่านักพัฒนามีความพยายามที่จะทำการแก้ไขปรับปรุงซอฟต์แวร์ด้วยวิธีการทำรีแฟคทอริง เป็นการสะท้อนให้เห็นถึงแนวโน้มเกี่ยวกับเรื่องคุณภาพของซอฟต์แวร์ในอนาคต ซึ่งงานที่ผู้วิจัยได้ทำการวิจัยอยู่นั้นจะเป็นการตรวจสอบในมุมมองของผู้ตรวจทานโค้ดถึงความสนใจในการนำรีแฟคทอริงมาใช้ในการปรังปรุงโค้ด

3) งานวิจัยที่มีการศึกษาเกี่ยวกับการปรับปรุงคุณภาพของซอฟต์แวร์ในโครงการโอเพนซอร์ส

ซอฟต์แวร์โอเพนซอร์สเป็นโครงการที่มีการจัดการข้อมูลที่ดี ทำให้ง่ายต่อการนำข้อมูลที่เก็บมาใช้ในการตรวจสอบคุณภาพของซอฟต์แวร์ (Yu, *et al.*, 2005) เช่น การติดตามข้อบกพร่องของซอฟต์แวร์ จากงานวิจัยก่อนหน้านี้ (Zhou and Davis, 2005) ได้มีการแสดงให้เห็นว่าการพัฒนาของ

โครงการโอเพนซอร์สมีรูปแบบการเติบโตและมีความน่าเชื่อถือที่คล้ายคลึงกับซอฟต์แวร์ลิขสิทธิ์ มีการตรวจสอบคุณภาพของซอฟต์แวร์โดยทำการบันทึกการเปลี่ยนแปลงของซอร์สโค้ด (Badri, *et al.*, 2012)

งานวิจัยที่มีการตรวจสอบคุณภาพของซอฟต์แวร์โอเพนซอร์สส่วนใหญ่จะมีการนำซอฟต์แวร์มาทำการเปรียบเทียบกันในเรื่องคุณภาพของซอร์สโค้ดและการทำงาน เช่น ทำการเปรียบเทียบระหว่างซอฟต์แวร์ในโครงการโอเพนซอร์สที่ต่างกัน งานวิจัยของ Stamelos และคณะ ได้ทำการตรวจสอบคุณภาพโครงสร้างของโค้ด ในซอฟต์แวร์ที่มีการใช้งานในระบบปฏิบัติการลีนุกซ์จำนวน 100 โปรแกรมมาทำการเปรียบเทียบกัน โดยใช้ Logiscope® มาเป็นมาตรฐานในการวัด ทำการสังเกตและประเมินความสัมพันธ์ระหว่างขนาดของส่วนประกอบและการนำไปประยุกต์ใช้ เป็นการวัดความพึงพอใจของผู้ใช้งานซอฟต์แวร์ จากการตรวจสอบพบว่าคุณภาพโครงสร้างของโค้ดค่อนข้างจะต่ำกว่ามาตรฐานที่กำหนดไว้ (Stamelos, *et al.*, 2002) ในทำนองเดียวกันงานวิจัยของ Jeong และคณะ ได้ทำการเปรียบเทียบคุณภาพของกระบวนการตรวจทานโค้ดระหว่าง Firefox และ Mozilla เพื่อทำการตรวจสอบดูว่าในขั้นตอนการพัฒนาซอฟต์แวร์โครงการใดมีอัตราที่จะได้รับการยอมรับหรือมีการแก้ไขน้อยที่สุดจากผู้ตรวจทานโค้ด พบว่าคุณภาพของซอฟต์แวร์ส่วนหนึ่งจะขึ้นอยู่กับความรู้ความสามารถของผู้ตรวจทานโค้ด ซึ่งในบางโครงการอาจจะมีการระบุผู้ตรวจทานโค้ดไว้ตั้งแต่ต้น เพื่อคาดหวังว่าจะได้รับข้อเสนอแนะหรือวิธีการปรับปรุงที่จะทำให้ซอฟต์แวร์มีคุณภาพมากขึ้น (Jeong, *et al.*, 2009) งานวิจัยของ Bakar และ Arsat ได้ทำการศึกษาปัจจัยที่มีผลต่อคุณภาพของซอฟต์แวร์โอเพนซอร์ส ปัจจัยดังกล่าวประกอบด้วยการบำรุงรักษา ความถูกต้องน่าเชื่อถือ ประสิทธิภาพ และการใช้งาน (Bakar and Arsat, 2014) การทำให้ซอฟต์แวร์โอเพนซอร์สมีประสิทธิภาพที่สูงขึ้นจะต้องมีคุณสมบัติที่สามารถปรับได้ตามระดับของความต้องการของผู้ใช้งาน การพัฒนาคุณภาพของซอฟต์แวร์จะวิเคราะห์จากการสังเกตวิวัฒนาการของระบบซอฟต์แวร์โอเพนซอร์ส (Soares, *et al.*, 2013 และ Ouni, *et al.*, 2013)

งานวิจัยในกลุ่มนี้มีความเกี่ยวข้องกับงานวิจัยที่ได้ศึกษา คือ งานที่ผู้วิจัยทำการศึกษาจะเป็นการศึกษาเกี่ยวกับคุณภาพของซอร์สโค้ดที่เกิดขึ้นในกระบวนการตรวจทานโค้ด โดยจะศึกษาในมุมมองของผู้ตรวจทานโค้ด และต้องการที่จะเพิ่มหลักฐานเชิงประจักษ์ที่เกี่ยวกับการเพิ่มคุณภาพของซอฟต์แวร์ในโครงการโอเพนซอร์สให้กับนักวิจัยด้านวิศวกรรมซอฟต์แวร์

4) งานวิจัยที่มีการศึกษาเกี่ยวกับการบำรุงรักษาซอฟต์แวร์ในโครงการโอเพนซอร์ส

การบำรุงรักษาซอฟต์แวร์เป็นสิ่งสำคัญในการใช้งานระบบซอฟต์แวร์ หลังจากที่ทำการติดตั้งระบบเสร็จเรียบร้อยแล้ว เมื่อถึงระยะเวลาหนึ่งระบบอาจต้องมีการปรับปรุงแก้ไขให้ทันต่อยุคสมัย หรือปรับปรุงข้อผิดพลาดให้ถูกต้อง การบำรุงรักษาซอฟต์แวร์เป็นการยืดอายุการงานให้ใช้

ต่อไปอย่างมีประสิทธิภาพ (Ouni, *et al.*, 2013 และ Yu, *et al.*, 2005) ได้มีงานวิจัยหลายชิ้นให้ความสำคัญกับกระบวนการ และขั้นตอนการบำรุงรักษาซอฟต์แวร์โอเพนซอร์ส (Koponen and Hotti, 2005; Raza, *et al.*, 2013; Midha and Bhattacherjee, 2012 และ Xiong, *et al.*, 2009)

งานวิจัยที่มีการบำรุงรักษาซอฟต์แวร์ด้วยวิธีการทำรีแฟคทอริงเพื่อเป็นการยืนยันว่าการทำรีแฟคทอริงเป็นวิธีที่สามารถช่วยปิดช่องโหว่ของการเกิดร่องรอยโค้ดที่ไม่ดีหรือป้องกันการถูกโจมตีในอนาคต และเป็นการกระตุ้นให้นักพัฒนาให้ความสำคัญกับการทำรีเฟคทอริง เช่น งานของ Cinnéide และคณะ เป็นลักษณะของการประเมินประโยชน์ที่ได้จากการทำรีเฟคทอริงในสถานการณ์จริง (Cinnéide, *et al.,* 2016)

บทที่ 3

วิธีการวิจัย

การศึกษาวิจัยนี้เป็นการค้นหาร่องรอยโค้ดที่ไม่ดีจากคำแนะนำของผู้ตรวจทานโค้ดที่มีการเสนอแนะให้นักพัฒนาทำการปรับปรุงโค้ดด้วยวิธีการทำรีแฟคทอริงในโครงการซอฟต์แวร์โอเพนซอร์ส เพื่อนำมาวิเคราะห์ในเชิงปริมาณ ในการดำเนินงานวิจัยมีขั้นตอนการวิจัยดังรูปที่ 3.1 สำหรับรายละเอียดการดำเนินการในแต่ละขั้นตอนมีรายละเอียดดังนี้

รูปที่ 3.1 ขั้นตอนการวิจัย

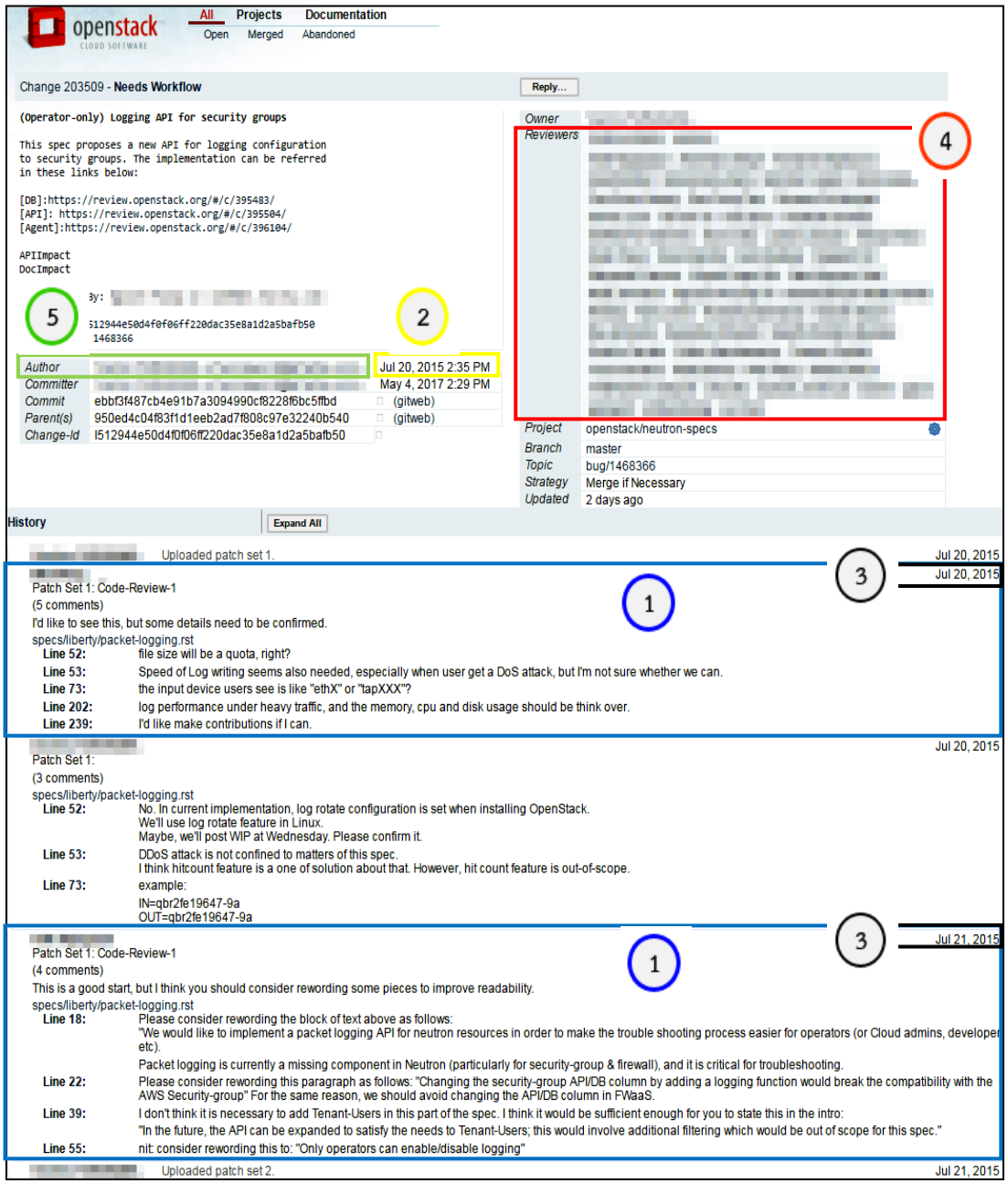
3.1 ศึกษาทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ผู้วิจัยเริ่มศึกษาข้อมูลและงานวิจัยที่เกี่ยวข้องกับการศึกษาข้อเสนอแนะของผู้ตรวจทานจากการตรวจทานโค้ด งานวิจัยที่มีการศึกษาเกี่ยวกับการทำรีแฟคทอริงและร่องรอยโค้ดที่ไม่ดี และงานวิจัยอื่น ๆ ที่เกี่ยวข้องกับการการปรับปรุงคุณภาพซอฟต์แวร์โอเพนซอร์ส ผู้วิจัยได้รวบรวม แนวความคิด ทฤษฎีและผลงานวิจัยที่เกี่ยวข้อง เพื่อนำมาเป็นแนวทางในการศึกษา

3.2 กำหนดข้อมูลของโครงการโอนเพนซอร์ส

ผู้วิจัยทำการกำหนดข้อมูลของโครงการโอนเพนซอร์สที่จำเป็นในการนำไปใช้วิเคราะห์เพื่อให้ได้คำตอบของงานวิจัย โดยพิจารณาจากโครงการโอเพนซอร์สที่ได้รับความนิยมในการนำมาวิเคราะห์ข้อเสนอแนะที่เกิดขึ้นในกระบวนการตรวจทานโค้ดจากงานวิจัยที่ผ่านมาโดยมีหลักเกณฑ์ในการพิจารณาดังนี้ 1) เป็นโครงการที่มีการใช้งานอยู่ในปัจจุบันและมีการพัฒนาอยู่ตลอดเวลา และ 2) มีความเป็นไปได้ในการเข้าถึงแหล่งข้อมูลที่จะนำมาวิเคราะห์ สำหรับงานวิจัยนี้ผู้วิจัยได้ทำการเลือกโครงการซอฟต์แวร์โอเพนซอร์สมา 2 โครงการ ได้แก่ OpenStack และ WikiMedia ซึ่งข้อมูลที่ใช้ศึกษาข้อเสนอแนะของผู้ตรวจทานโค้ดในอยู่ในช่วงปี ค.ศ. 2011- 2015 โดยข้อมูลจากทั้งสองโครงการนี้ได้ถูกจัดเก็บอยู่ในระบบเกอริต แสดงดังรูปที่ 3.2 เป็นการแสดงตัวอย่างข้อมูลของโครงการ OpenStack ที่จัดเก็บอยู่ในระบบเกอริต สำหรับข้อมูลที่นำมาศึกษามีดังนี้

1) ข้อเสนอแนะที่เกิดขึ้นในกระบวนการตรวจทานโค้ด
2) รายละเอียดวันเวลาของการส่งคำร้องขอ
3) รายละเอียดวันเวลาของการตรวจทาน
4) ข้อมูลชื่อของผู้ตรวจทานโค้ด
5) ข้อมูลชื่อของเจ้าของซอร์สโค้ดที่มีการร้องขอให้ผู้ตรวจทานโค้ดช่วยทำการตรวจทานโค้ด

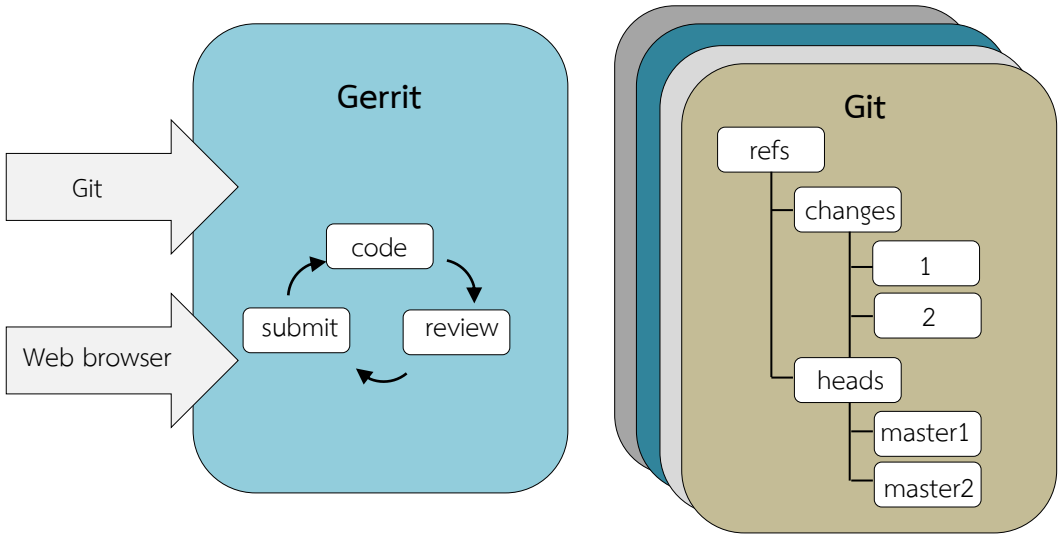


**หมายเหตุ: ผู้วิจัยต้องปกปิดชื่อของ Reviewer และ Author เพื่อเป็นการรักษาสิทธิส่วนบุคคล*

รูปที่ 3.2 ตัวอย่างข้อมูลของโครงการ OpenStack ที่จัดเก็บอยู่ในระบบเกอริต

3.3 ศึกษาโครงสร้างการเก็บข้อมูลของระบบเกอริต

ผู้วิจัยได้ทำการศึกษาถึงโครงสร้างการเก็บข้อมูลของระบบเกอริต เพื่อนำมาใช้ในการออกแบบและพัฒนาซอฟต์แวร์สืบค้นข้อมูลและจัดเก็บข้อมูล ระบบเกอริตนั้นจะมีการทำงานร่วมกับระบบจัดการกิต ซึ่งเป็นระบบซอฟต์แวร์ที่ใช้ในการจัดการเวอร์ชันของซอร์สโค้ดแสดงดังรูปที่ 3.3



รูปที่ 3.3 กระบวนการทำงานของเกอริต
อ้างอิงภาพมาจาก http://git-scm.com, วันที่ 1 พฤษภาคม 2560

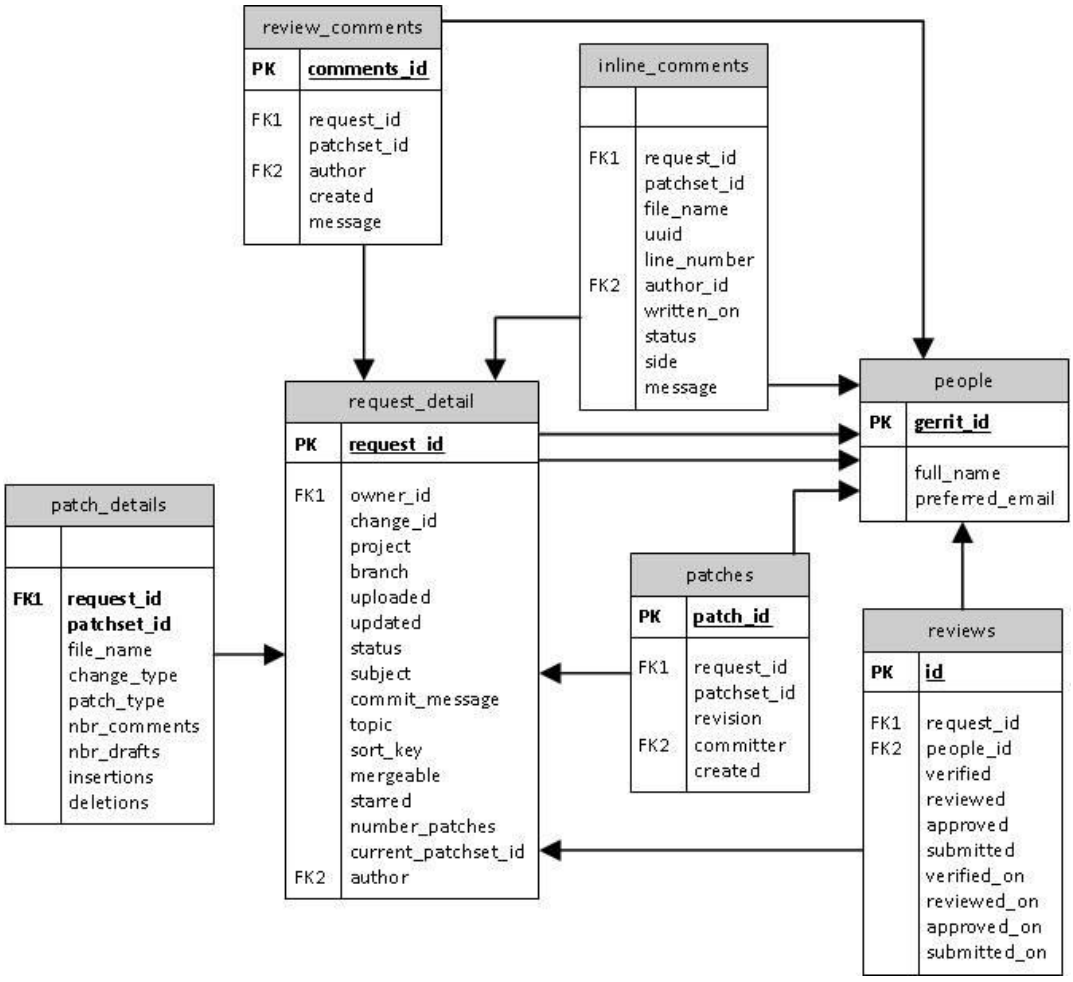
จากรูปที่ 3.3 จะเห็นว่าการใช้งานเกอริต มี 2 ส่วน คือ

1) เริ่มจากผู้ที่มีส่วนเกี่ยวข้องอาจจะเป็นผู้ตรวจทานโค้ด ผู้ที่ส่งคำร้องขอ หรือนักพัฒนาซอฟต์แวร์ได้ทำการเปลี่ยนแปลงซอร์สโค้ด การเปลี่ยนแปลงเหล่านี้จะถูกอัปโหลดไปยังเกอริตและประวัติการเปลี่ยนแปลงของซอร์สโค้ดจะถูกเก็บไว้ในกิต

2) นักพัฒนาจะสามารถทำการตรวจทานโค้ดผ่านเว็บเบราว์เซอร์ เกอริตสามารถจัดเก็บข้อมูลได้หลายโครงการแต่ละโครงการสามารถแยก (branch) ออกจากโปรแกรมหลัก (main program) ไปทำงานที่โปรแกรมย่อยได้ ในกรณีที่นักพัฒนาทำการยืนยัน (commit) การเปลี่ยนแปลงของซอร์สโค้ดก็จะมีการอ้างอิงไปยัง refs/changes/ เส้นทางหรือตำแหน่งที่ซอร์สโค้ดนั้นถูกเก็บอยู่

(1 หรือ 2) เมื่อนักพัฒนาทำการผลักข้อมูล (push) การเปลี่ยนแปลงจะมีเส้นทางไปยัง refs/changes /heads/ เส้นทางหรือตำแหน่งที่ต้องการ (master1 หรือ master2) ตัวอย่างเช่น เมื่อนักพัฒนาทำการอัปโหลดไฟล์ไปที่แยก dev การอ้างอิงเป้าหมายจะเป็น refs/for/ dev

เกอริต เป็นระบบที่ช่วยอำนวยความสะดวกในเรื่องการตรวจทานโค้ดให้กับนักพัฒนาซอฟต์แวร์ในกระบวนการตรวจทานโค้ด ในปัจจุบันมีโครงการโอเพนซอร์สมากกว่า 2,000 โครงการที่ใช้ระบบเกอริตในกระบวนการตรวจทานโค้ด ลักษณะของโครงสร้างการเก็บข้อมูลของเกอริต แสดงดังรูปที่ 3.4 (Bosu and Carver, 2014) สำหรับรายละเอียดของโครงสร้างการเก็บข้อมูลในเกอริตจะแสดงอยู่ในภาคผนวก ก



รูปที่ 3.4 โครงสร้างการเก็บข้อมูลในเกอริต

3.4 พัฒนาซอฟต์แวร์ที่ช่วยในการจัดการข้อมูล

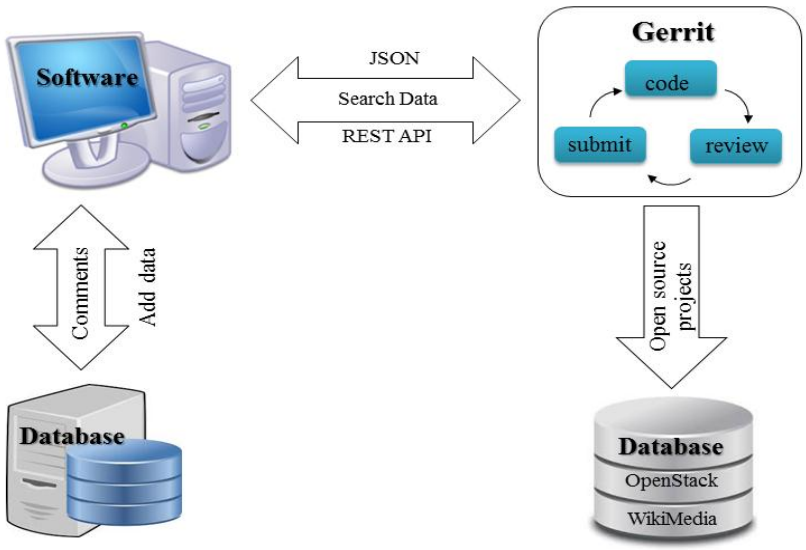
ผู้วิจัยทำการพัฒนาซอฟต์แวร์ที่ช่วยในการสืบค้นข้อมูลและจัดเก็บข้อมูล ซึ่งข้อมูลนี้เป็นข้อมูลที่เกี่ยวข้องกับการตรวจทานโค้ดในโครงการซอฟต์แวร์โอเพนซอร์สจากระบบเกอริต โดยข้อมูลที่ได้จากซอฟต์แวร์นี้จะถูกจัดเก็บในระบบฐานข้อมูล สำหรับประโยชน์ของการจัดเก็บข้อมูลในระบบฐานข้อมูลจะช่วยให้ผู้วิจัยสามารถจัดการ และสืบค้นข้อมูลได้ง่าย นอกจากนี้การจัดเก็บข้อมูลในระบบฐานข้อมูลจะเอื้อให้ผู้วิจัยแปลงข้อมูลให้อยู่ในรูปแบบต่าง ๆ ได้ง่าย สำหรับในงานวิจัยนี้ผู้วิจัยได้เลือกซอฟต์แวร์โอเพนซอร์สที่จะนำมาทำการศึกษาจำนวน 2 โครงการ ได้แก่ OpenStack และ WikiMedia

ในการพัฒนาซอฟต์แวร์นั้น ผู้วิจัยได้ทำการศึกษาโครงสร้างของข้อมูลในระบบเกอริต โดยในระบบเกอริตนั้นจะมีเว็บเซอร์วิสให้บริการในการสืบค้นข้อมูล และดาวน์โหลดข้อมูลในรูปแบบ REST API[13] ซึ่งข้อมูลที่ดาวน์โหลดได้นั้นจะอยู่ในรูปแบบ JavaScript Object Notation (JSON) สำหรับซอฟต์แวร์ที่พัฒนาขึ้นมานั้นถูกพัฒนาด้วยภาษาจาวา เนื่องจากในภาษาจาวาจะมีไลบรารีในการทำงานกับ REST API และ JSON ซึ่งจะทำให้การพัฒนาซอฟต์แวร์สะดวกขึ้น

สำหรับฟังก์ชันหลักของซอฟต์แวร์ที่พัฒนามีดังนี้

- สืบค้นข้อมูลจากระบบเกอริตโดยสามารถกำหนดเงื่อนไขในการค้นหาได้
- แปลงข้อมูลที่อยู่ในรูปแบบ JSON ให้อยู่ในรูปแบบข้อมูลที่มีโครงสร้างซึ่งสามารถจัดเก็บอยู่ในระบบฐานข้อมูล
- บันทึกข้อมูลลงระบบฐานข้อมูล ซึ่งระบบฐานข้อมูลนี้จะถูกออกแบบให้รองรับข้อมูลที่จำเป็นต้องใช้ในขั้นตอนที่ 3

[13] http://www.Review.typo3.org

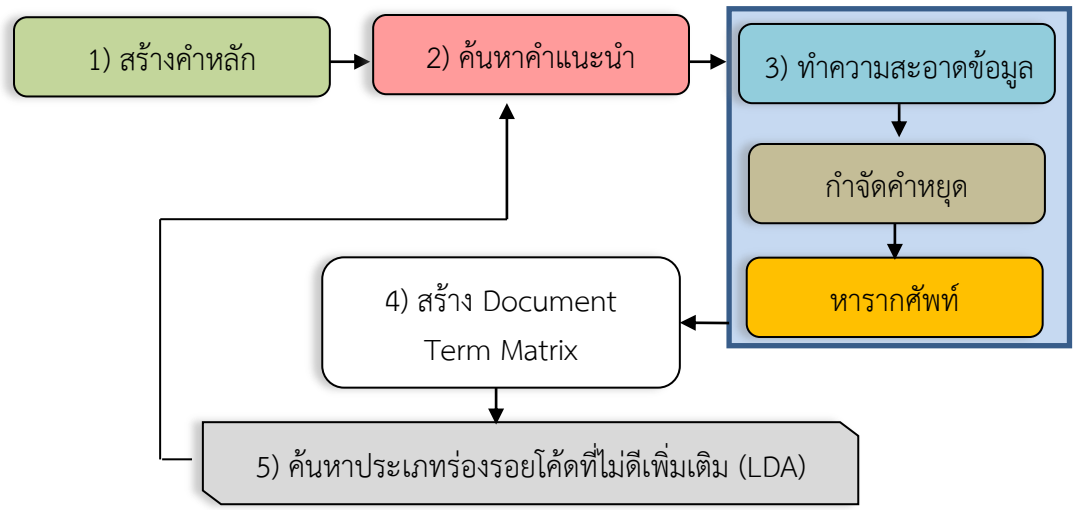


รูปที่ 3.5 การทำงานของซอฟต์แวร์ที่ทำการพัฒนา

จากรูปที่ 3.5 กระบวนการทำงานของซอฟต์แวร์จะเริ่มต้นจากการค้นหาข้อเสนอแนะที่เกิดขึ้นในกระบวนการตรวจทานโค้ดโดยการใช้ REST API ในการดึงข้อมูลจากระบบฐานข้อมูลของเกอริต ซึ่งระบบเกอริตจะมีการจัดเก็บข้อมูลของข้อเสนอแนะที่เกิดขึ้นจากกระบวนการตรวจทานโค้ดของโครงการซอฟต์แวร์โอเพนซอร์สไว้ ในงานวิจัยนี้ต้องการจะรวบรวมข้อเสนอแนะของซอฟต์แวร์โอเพนซอร์ส 2 โครงการ คือ โครงการ OpenStack และ โครงการ WikiMedia โดยข้อมูลที่ได้จากการค้นหาโดย REST API จะถูกส่งกลับมาในรูปแบบของ JSON และข้อมูลนี้จะถูกแปลงให้อยู่ในรูปแบบที่สามารถจัดเก็บในระบบฐานข้อมูลเชิงสัมพันธ์ สำหรับโครงสร้างฐานข้อมูลที่ใช้ในการจัดเก็บข้อมูลสำหรับงานวิจัยนี้มีรายละเอียดซึ่งจะแสดงอยู่ในภาคผนวก ข

3.5 จัดกลุ่มประเภทร่องรอยโค้ดที่ไม่ดี

ในขั้นตอนนี้ผู้วิจัยได้ทำการจัดกลุ่มประเภทของร่องรอยโค้ดที่ไม่ดีที่ปรากฏอยู่ภายในประโยคต่าง ๆ ของข้อเสนอแนะจากผู้ตรวจทานโค้ดในโครงการซอฟต์แวร์โอเพนซอร์ส สำหรับขั้นตอนในการจัดกลุ่มประเภทร่องรอยโค้ดที่ไม่ดีแสดงดังรูปที่ 3.6



รูปที่ 3.6 ขั้นตอนในการจัดกลุ่มประเภทร่องรอยโค้ดที่ไม่ดี

จากรูปที่ 3.6 สามารถอธิบายรายละเอียดในขั้นตอนการจัดกลุ่มประเภทร่องรอยโค้ดที่ไม่ดีได้ดังนี้

1) สร้างคำหลัก (Keyword)

การสร้างคำหลักที่เกี่ยวข้องกับประเภทของร่องรอยโค้ดที่ไม่ดี เช่น Duplicate, Long, Complexity ซึ่งคำหลักนี้ได้นำมาจากหนังสือ Refactoring: Improving the Design of Existing Code (Fowler and Beck, 1999) ซึ่งร่องรอยโค้ดที่ไม่ดีในหนังสือเล่มนี้ได้รับการศึกษาและได้รับการยอมรับโดยทั่วไปในกลุ่มของวิศวกรรมซอฟต์แวร์ โดยปกติแล้วในคำแนะนำของผู้ตรวจทานโค้ดจะไม่ได้มีการใช้คำที่ตรงตามคำหลักที่ได้ระบุไว้ ผู้วิจัยจึงทำการระบุคำที่มีความหมายสื่อถึงคำหลัก (Synonyms) ผู้วิจัยได้ทำการค้นหาคำเหล่านี้โดยการใช้ โปรแกรม R[14] ซอฟต์แวร์เวิร์ดเน็ต[15] (Fellbaum, 1998) ซึ่งเป็นซอฟต์แวร์ที่ได้รับความนิยมนำไปใช้ในงานวิจัยที่มีลักษณะเดียวกันกับงานวิจัยนี้ และพจนานุกรม เป็นตัวช่วยในการค้นหา เพื่อให้ได้คำหลักมากที่สุดเท่าที่จะเป็นไปได้

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

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

2) ค้นหาคำแนะนำ (Comment Retriveval)

ในขั้นตอนนี้ผู้วิจัยทำการค้นหาคำแนะนำของผู้ตรวจทานโค้ดที่มีการกล่าวถึงร่องรอยของที่ไม่ดีอยู่อย่างน้อย 1 คำจากฐานข้อมูล โดยใช้คำหลักที่ได้มาจากในข้อ 1) ในการค้นหาข้อมูลนี้ผู้วิจัยจะทำการพัฒนาคำสั่ง SQL ที่มีคำหลักอยู่อย่างน้อย 1 คำ เพื่อป้องกันปัญหาการใช้เวลานานในการสืบค้นข้อมูลแต่ละครั้ง ผู้วิจัยได้พัฒนาชุดคำสั่ง SQL ออกเป็นชุดคำสั่งย่อย ๆ โดยใช้เงื่อนไขของเวลาที่ปรากฏในคำแนะนำที่จัดเก็บอยู่ในระบบเกอริต หลังจากที่ได้ผลลัพธ์จากการสืบค้นข้อมูลแล้ว ผู้วิจัยจะนำผลลัพธ์ที่ได้ไปจัดเก็บไว้ในรูปแบบของฐานข้อมูล

3) ทำความสะอาดข้อมูล (Data Cleaning)

ในขั้นตอนนี้ผู้วิจัยทำการลบช่องว่าง เครื่องหมายวรรคตอน และตัวเลขออกจากข้อความที่ได้จากการค้นหา เพื่อเป็นการทำให้ดัชนีของข้อมูลลดลงและลดเวลาในการประมวลผล หลังจากนั้นจะทำการแยกชื่อตัวแปร หรือเมทธอดที่มีคำติดกันออกจากกัน (Identifier Splitting) ซึ่งชื่อเหล่านี้อาจจะปรากฏอยู่ในคำแนะนำ เช่น “isBufferFull” จะแยกออกมาเป็น “is buffer full” หรือ “read_string” จะแยกออกมาเป็น “read string”

3.1) การกำจัดคำหยุด (Stop-Word List Removal)

เป็นการนำคำที่ไม่มีนัยสำคัญออกโดยที่ไม่ทำให้ความหมายของคำ หรือข้อความเปลี่ยนแปลง คำหยุดมักจะปรากฏอยู่ในข้อความทุกข้อความ คำหยุดเป็นคุณลักษณะที่ไม่เกี่ยวข้องหรือไม่มีประโยชน์ในการค้นคืน หรือการจำแนกหมวดหมู่ การกำจัดคำหยุดเป็นกระบวนการที่จะช่วยให้ขนาดของดัชนีลดลง และยังลดเวลาในการประมวลผล ตัวอย่างคำหยุดในภาษาอังกฤษ ได้แก่ a, an, the

3.2) ทำการหารากศัพท์ (Stemming)

การหาคำที่ยังไม่เปลี่ยนรูป ยังไม่เติมคำอุปสรรค (Prefixes) หรือยังไม่เติมคำปัจจัย (Suffixes) เป็นการหารูปแบบเดิมของคำที่แสดงความหมายหลัก เช่น คำว่า connection, connections, connective, connected และ connecting รากศัพท์ คือ connect ในการดำเนินงานขั้นตอนนี้ทางผู้วิจัยเลือกใช้อัลกอริทึม Porter’s stemming (Willett, 2006) ในการหารากศัพท์

4) สร้าง Document Term Matrix

ในขั้นตอนนี้ผู้วิจัยนำคำที่ผ่านการหารากศัพท์จากข้อที่ 3.2) มาสร้าง Document Term Matrix โดยมีโครงสร้างที่เป็นรูปแบบ สดมภ์ และแถว โดยข้อมูลในแถวจะเป็นรากศัพท์ และสดมภ์จะเป็นหมายเลขของเอกสาร จำนวนความถี่ของการปรากฏคำในแต่ละเอกสารจะแสดงอยู่ในช่องตาราง แสดงดังตัวอย่างในตารางที่ 3.2 คำว่า duplicate, complex, long, large, และ mean

จะเป็นรากศัพท์ และ Y1, Y2, และ Y3 แสดงจำนวนความถี่ของคำที่ปรากฏสำหรับร่องรอยโค้ดที่ไม่ดีในแต่ละปีจะแสดงอยู่ในช่องตาราง

ตารางที่ 3.2 ตัวอย่างของดัชนีที่ใช้ในงานวิจัยนี้

	Y1	Y2	Y3
duplicate	1	2	0
complex	1	0	0
long	1	1	1
large	2	1	1
mean	1	1	1

5) ค้นหาประเภทร่องรอยโค้ดที่ไม่ดีเพิ่มเติม

ในขั้นตอนนี้ผู้วิจัยทำการค้นหาประเภทของร่องรอยโค้ดที่ไม่ดีประเภทอื่น ๆ ที่ซ่อนอยู่ในข้อเสนอแนะของผู้ตรวจทานโค้ด เพื่อเป็นการค้นหาประเภทของร่องรอยโค้ดที่ไม่ดีที่พบในโครงการโอเพนซอร์สเพิ่มเติมโดยอาศัยอัลกอริทึม LDA ในการหาประเภทร่องรอยโค้ดที่ไม่ดีเพิ่มเติม ผู้วิจัยจะดำเนินการในขั้นตอนที่ 2) ถึง 5) ซ้ำจนกว่าจะไม่พบร่องรอยโค้ดที่ไม่ดีเพิ่มเติมอีก

สำหรับในขั้นตอนที่ 3) ถึง 7) นั้นทางผู้วิจัยได้มีการนำโปรแกรม R มาช่วยดำเนินการในส่วนของการทำเหมืองข้อความ (Text Mining) โดยมีการเลือกใช้งาน package tm และในส่วนของค้นหาประเภทร่องรอยโค้ดที่ไม่ดีเพิ่มเติมจะมีการเรียกใช้งาน package lda

3.6 วิเคราะห์ข้อมูล

ในการวิเคราะห์ข้อมูลเพื่อหาอิทธิพลของการทำรีแฟคทอริงต่อการตรวจทานโค้ดในโครงการซอฟต์แวร์โอเพนซอร์ส ผู้วิจัยจะทำการค้นหา และจัดกลุ่มประเภทของร่องรอยโค้ดที่ไม่ดีเข้าด้วยกัน ซึ่งจะนำข้อมูลเชิงปริมาณ (Quantitative) ที่ได้จากการค้นหานี้มาวิเคราะห์ผล โดยผู้วิจัยจะใช้อัลกอริทึม LDA ช่วยในการจัดกลุ่มประเภทของร่องรอยโค้ดที่ไม่ดีที่มีปรากฏอยู่ภายในประโยคต่าง ๆ ของข้อแนะนำจากผู้ตรวจทานโค้ดจำนวนมากในโครงการซอฟต์แวร์โอเพนซอร์สขนาดใหญ่ สำหรับวิธีการทางสถิติที่ผู้วิจัยนำมาช่วยในการวิเคราะห์ข้อมูลในครั้งนี้ มีการใช้สถิติการถดถอย (Regression analysis) เพื่อใช้ในการพยากรณ์แนวโน้มของการเกิดร่องรอยโค้ดที่ไม่ดีในอนาคต และสถิติสหสัมพันธ์ (Correlation) (Navidi, 2010) มาช่วยในการวิเคราะห์หาความสัมพันธ์ระหว่างบทบาทของการเป็นผู้ตรวจทานโค้ดและบทบาทของการเป็นผู้ส่งคำร้องขอว่ามีความสัมพันธ์กันหรือไม่ ประกอบกับการใช้เครื่องมือทางสถิติต่าง ๆ ที่เหมาะสม โดยจะทำการกำหนดค่าของสถิติสหสัมพันธ์ให้มีระดับความเชื่อมั่นอยู่ที่ r = 0.05 ในการแปลผลค่าระดับความเชื่อมั่นมีรายละเอียดดังนี้ (Navidi, 2010)

ค่า r	ระดับของความสัมพันธ์
0.90 – 1.00	มีความสัมพันธ์กันสูงมาก
0.70 – 0.90	มีความสัมพันธ์กันในระดับสูง
0.50 – 0.70	มีความสัมพันธ์กันในระดับปานกลาง
0.30 – 0.50	มีความสัมพันธ์กันในระดับต่ำ
0.0 – 0.30	มีความสัมพันธ์กันในระดับต่ำมาก

บทที่ 4

ผลการวิจัย

ในบทนี้ผู้วิจัยจะนำเสนอผลการวิเคราะห์ข้อมูลซึ่งผู้วิจัยได้ทำการรวบรวมข้อเสนอแนะของผู้ตรวจทานโค้ดในโครงการ OpenStack และโครงการ WikiMedia ที่มีการแนะนำให้นักพัฒนาปรับปรุงหรือแก้ไขซอร์สโค้ดด้วยวิธีการรีแฟคทอริง ซึ่งข้อมูลของข้อเสนอแนะเหล่านี้ได้มาจากกระบวนการตรวจทานโค้ดที่มีการเก็บข้อมูลไว้ในระบบเกอริต ซึ่งได้ดำเนินการตามขั้นตอนของงานวิจัย และผลที่ได้จากการวิจัยในครั้งนี้ได้แบ่งออกเป็น 4 ส่วน คือ 1) ผลที่ได้จากการกำหนดคำหลักและคำที่มีความหมายสื่อถึงคำหลัก 2) ผลที่ได้จากการค้นหาประเภทร่องรอยโค้ดที่ไม่ดีเพิ่มเติม 3) ผลที่ได้จากการค้นหาข้อเสนอแนะที่ผู้ตรวจทานโค้ดได้ให้คำแนะนำเกี่ยวกับการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง และ 4) ผลของการวิเคราะห์ข้อเสนอแนะของผู้ตรวจทานโค้ดทั้งสองโครงการ โดยมีรายละเอียดดังนี้

4.1 ผลที่ได้จากการกำหนดคำหลักและคำที่มีความหมายสื่อถึงคำหลัก

4.1.1 การกำหนดคำหลัก

ก่อนที่จะทำการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริงในขั้นตอนแรกจะต้องทำการระบุตำแหน่งของโค้ดที่มีความเสี่ยงที่จะเกิดข้อบกพร่องหรือทำให้ซอฟต์แวร์เกิดความเสียหายต่อการทำงานในอนาคต ข้อบกพร่องที่เกิดขึ้น มีชื่อเรียกว่า “ร่องรอยโค้ดที่ไม่ดี” จากการศึกษาเกี่ยวกับลักษณะของร่องรอยโค้ดที่ไม่ดีในหนังสือของ Fowler and Beck (1999) ได้มีการระบุเกี่ยวกับร่องรอยโค้ดที่ไม่ดีไว้ทั้งหมด 22 ประเภท พร้อมทั้งนำเสนอวิธีการทำรีแฟคทอริงที่เหมาะสมที่จะใช้ในการกำจัดร่องรอยโค้ดที่ไม่ดีในแต่ละประเภทไว้ด้วย ร่องรอยโค้ดที่ไม่ดีทั้ง 22 ประเภทนี้ได้รับการศึกษาและได้รับการยอมรับโดยทั่วไปในกลุ่มของวิศวกรรมซอฟต์แวร์ ในงานวิจัยนี้ผู้วิจัยได้นำประเภทของร่องรอยโค้ดที่ไม่ดีทั้ง 22 ประเภท มาเป็นคำหลักเพื่อใช้ในการค้นหาคำแนะนำของผู้ตรวจทานโค้ด ซึ่งร่องรอยโค้ดที่ไม่ดีทั้ง 22 ประเภทเหล่านี้ เป็นลักษณะของโค้ดที่ทำ

การพัฒนาขึ้นมาแล้วมีแนวโน้มก่อให้เกิดปัญหาหรือข้อผิดพลาดที่อาจจะเกิดขึ้นได้ ทำให้เกิดการพัฒนาที่ล่าช้าและสามารถเพิ่มความเสี่ยงให้เกิดปัญหาต่าง ๆ ในอนาคตได้ ซึ่งจะส่งผลให้ซอฟต์แวร์ทำงานอย่างไม่มีประสิทธิภาพ และทำให้คุณภาพของซอร์สโค้ดอาจจะไม่ดีเท่าที่ควรจึงเป็นสาเหตุที่ทำให้เกิดปัญหาในด้านคุณภาพซอฟต์แวร์ ดังนั้นเพื่อให้ซอฟต์แวร์มีคุณภาพเพิ่มมากขึ้นจะต้องลดจำนวนร่องรอยโค้ดที่ไม่ดีที่เกิดขึ้นให้น้อยลง จึงทำให้ต้องแก้ไขโดยการทำรีแฟคทอริง ร่องรอยโค้ดที่ไม่ดีในแต่ละประเภทจะมีวิธีการทำรีแฟคทอริงที่เหมาะสมในการกำจัดร่องรอยโค้ดที่ไม่ดีสำหรับการพัฒนาโปรแกรม

ร่องรอยโค้ดที่ไม่ดีแต่ละประเภทจะประกอบด้วยคำอย่างน้อย 2 คำ ซึ่งในการกำหนดคำหลักนั้นมีวัตถุประสงค์เพื่อต้องการให้ข้อมูลของข้อเสนอแนะที่ได้จากการค้นหามีความสมบูรณ์และลดการเกิดข้อมูลที่ไม่มีส่วนเกี่ยวข้องกับการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง โดยผู้วิจัยได้กำหนดหลักเกณฑ์ที่ใช้ในการพิจารณาเลือกร่องรอยโค้ดที่ไม่ดีมาสร้างเป็นคำหลักดังนี้

1) ไม่นำคำศัพท์ที่เป็นคำหลักของภาษาโปรแกรมมาเป็นคำหลัก ตัวอย่าง เช่น Attribute, Class, Code, Conditional, Feature, Field, Function, Method, Object, Parameter, Value เป็นต้น

2) เลือกคำที่แสดงอาการ สภาพ หรือการกระทำต่าง ๆ ตัวอย่าง เช่น ร่องรอยโค้ดที่ไม่ดีประเภท Dead Code เกิดจากการรวมกันของคำ Dead และ Code จากตัวอย่างนี้จะทำการเลือกคำว่า Dead มาเป็นคำหลักเนื่องจากเป็นคำที่แสดงให้เห็นถึงอาการหรือสภาพที่บ่งบอกถึงการหยุดนิ่ง อย่างไรก็ตามหลังจากที่ได้ทำการค้นหาข้อมูลเรียบร้อยแล้วผู้วิจัยยังคงนำคำว่า Code ที่ปรากฏอยู่ในประเภทของร่องรอยโค้ดที่ไม่ดีมาเป็นตัวช่วยการตัดสินใจในขั้นตอนสุดท้ายของการทำความสะอาดข้อมูลเพื่อตรวจสอบดูว่าข้อเสนอแนะที่เกิดขึ้นสามารถนำมาทำการวิเคราะห์ได้ไหรือไม่

3) ในกรณีที่ประเภทของร่องรอยโค้ดที่ไม่ดีบางประเภทไม่สามารถที่จะใช้เกณฑ์การพิจารณาตามที่กำหนดไว้ใน ข้อที่ 1) และ ข้อที่ 2) ให้ทำการเลือกคำที่มีความเหมาะสมในการนำไปใช้ค้นหาข้อเสนอแนะ เช่น ร่องรอยโค้ดที่ไม่ดีประเภท Too Many Parameters ในกรณีนี้จะไม่สามารถเลือกคำว่า Parameters มาเป็นคำหลักได้เนื่องจากเป็นคำศัพท์ที่ใช้ในการพัฒนาซอฟต์แวร์ และไม่สามารถนำคำว่า Many หรือคำว่า Too มาเป็นคำหลักได้ เนื่องจากทั้ง 2 คำจะเป็นคำพื้นฐานที่มักจะพบเจอในประโยคทั่วไป ผู้วิจัยจึงทำการนำคำว่า Many และคำว่า Parameters มารวมกัน

หลังจากนำประเภทร่องรอยโค้ดที่ไม่ดีมาผ่านเกณฑ์การพิจารณาเรียบร้อยแล้ว ผู้วิจัยได้ทำการกำหนดคำหลักดังตารางที่ 4.1 ซึ่งแสดงให้เห็นถึงที่มาของคำหลักแต่ละคำว่าผู้วิจัยได้นำร่องรอยโค้ดที่ไม่ดีประเภทไหนบ้างมาใช้เป็นคำหลัก

ตารางที่ 4.1 คำหลักที่จะนำไปใช้ในการค้นหาข้อเสนอแนะของผู้ตรวจทานโค้ด

ลำดับที่	ชนิดของร่องรอยโค้ดที่ไม่ดี	คำหลัก
1	Message Chains	Chains
2	Data Clump	Clump
3	Conditional Complexity	Complex
4	Dead Code	Dead
5	Divergent Changes	Divergent
6	Duplicate Code	Duplicate
7	Feature Envy	Envy
8	Inappropriate intimacy and Generality	Inappropriate
9	Incomplete Library Class	Incomplete
10	Parallel Inheritance Hierarchies	Inheritance
11	Large Class	Large
12	Long Parameter List and Method	Long
13	Too Many Parameters	Many Parameters
14	Comments Meaning	Meaning
15	Middle Man	Middle
16	Primitive Obsession	Primitive
17	Refused Bequest	Refused
18	Speculative Generality	Speculative
19	Shotgun Surgery	Surgery
20	Switch Statement	Switch
21	Temporary Field	Temporary
22	Uncommunicative Name	Understand

4.1.2 ผลของการค้นหาคำที่มีความหมายสื่อถึงคำหลัก

โดยปกติแล้วคำแนะนำของผู้ตรวจทานโค้ดจะไม่ได้มีการใช้คำที่ตรงตามคำหลักที่ได้ระบุไว้ในตารางที่ 4.1 ดังนั้นผู้วิจัยจึงทำการค้นหาคำที่มีความหมายที่สื่อถึงคำหลักโดยการใช้โปรแกรม R ซอฟต์แวร์เวิร์ดเน็ต และพจนานุกรม มาเป็นตัวช่วยในการค้นหา เพื่อให้ข้อมูลมีความครบถ้วนสมบูรณ์ ยกตัวอย่างเช่น เมื่อต้องการที่จะทำการค้นหาคำที่มีความหมายสื่อถึงคำหลัก “Chains”

ผลลัพธ์ที่ได้จากเครื่องมือแต่ละตัว

โปรแกรม R	=	{ chain, connect, connected, link }
ซอฟต์แวร์เวิร์ดเน็ต	=	{ chain, connect, sequence connected, continuity, progression, concatenation }
พจนานุกรม	=	{ chain, connect, connected, continual, continue, sequence, successive, unremitted }
คำที่มีความหมายสื่อถึงคำหลัก	=	{ chain, connect, connected, link, concatenation, connection, progression, successive, unremitted, continual, continuity, continue }

ลักษณะของผลลัพธ์ที่ได้จากการนำเครื่องมือเหล่านี้มาช่วยค้นหาคำที่มีความหมายสื่อถึงคำหลักจะมี 2 ลักษณะ คือ

1) คำที่มีความหมายสื่อถึงคำหลักซ้ำกัน (จากตัวอย่างจะแสดงคำที่ซ้ำกันโดยใช้สัญลักษณ์การขีดเส้นใต้) : ในกรณีนี้ผู้วิจัยจะเลือกคำที่มีความหมายสื่อถึงคำหลักที่ซ้ำกันจากเครื่องมือทั้ง 3 มาเพียง 1 คำ เพื่อนำมาเป็นตัวแทนในการระบุให้เป็นคำที่มีความหมายสื่อถึงคำหลัก

2) คำที่มีความหมายสื่อถึงคำหลักที่แตกต่าง : ในกรณีนี้ผู้วิจัยนำคำที่ได้จากทั้ง 3 เครื่องมือมาเพื่อระบุเป็นคำที่มีความหมายสื่อถึงคำหลัก

ในตารางที่ 4.2 จะแสดงผลที่ได้จากการค้นหาคำที่มีความหมายสื่อถึงคำหลักที่ได้ทำการรวบรวมมาจากโปรแกรม R ซอฟต์แวร์เวิร์ดเน็ต และพจนานุกรม

ตารางที่ 4.2 ผลของการค้นหาคำที่มีความหมายสื่อถึงคำหลัก

ลำดับที่	คำหลัก	คำที่มีความหมายสื่อถึงคำหลัก
1	Chains	chain, continue, connected, continual, connect, unremitted, successive, concatenation, link, continuity, progression, sequence, connection,
2	Clump	group, cluster, collection, bloc, congestion, lot, suite, chunk, hunk, lump, gob, set, clump
3	Complex	complex, complicated, sophisticated, multiple, labyrinthine, convoluted, confused, entangled, mixed, multifarious, multiform, discursive, disordered, muddled, paradoxical, knotty, mingled, undecipherable, coordination, tangled, intricate, bewildering, hidden, inscrutable, puzzling, cryptic
4	Dead	dead, deceased, defunct, lifeless, inflexible, lost
5	Divergent	distinct, novel, unlike, variant, different, unlike, disparate, divergent, dissimilar
6	Duplicate	duplicate, repeatedly, repetitive, doubly, alike, overlapping, identical, duplication, replication, ditto, double, likeness, repetition, clone, copy, imitate, replicate, redo, rework, act like, do again, make like, repeat
7	Envy	envy, use, apply, rivalry, coveting, desire, hanker
8	Inappropriate	inappropriate, intimacy, impolitic, improper, inadvisable, inappropriate, inapt, indiscreet, inapplicable, incorrect, infelicitous, unworthy, irrelevant, unseemly, unsuitable, ill suited, inept, undue, unmeet

ตารางที่ 4.2 ผลของการค้นหาคำที่มีความหมายสื่อถึงคำหลัก (ต่อ)

ลำดับที่	คำหลัก	คำที่มีความหมายสื่อถึงคำหลัก
9	Incomplete	imperfect, incomplete, flawed, partially, unfit, defective, sketchy, unfinished, unwholesome, deficient, inadequate, insufficient, lacking, partial, garbled, incoherent, uncompleted, uncompleted
10	Inheritance	inherit, relay, hierarchies, hierarchy, heritage, inheritance, legacy, estate, bequest, patrimony
11	Large	big, large, great, huge, bulky, much, roomy, grand, jumbo, considerable, gigantic, massive, sizable, colossal, copious, exorbitant, giant, immeasurable,voluminous, excessive, enormous
12	Long	lengthily, long, continued, expanded, prolix, lengthened, stretch, gangling, longish, lengthy, boundless, long-winded, overlong, unending
13	Many parameters	much, numerous, many, great, considerable, massive
14	Meaning	meaning, implication, signification, purport, gist, connotation, content, definition, explanation, hint, interpretation, significance, understanding, suggestion, significant, idea, intent, purpose, significant
15	Middle	middle, intermediary, between, middleman, intermediate, center, betwixt, centermost, core, medial, middlemost, midmost, midst, midpoint

ตารางที่ 4.2 ผลของการค้นหาคำที่มีความหมายสื่อถึงคำหลัก (ต่อ)

ลำดับที่	คำหลัก	คำที่มีความหมายสื่อถึงคำหลัก
16	Primitive	obsession, original, primitive, orthodox, primal, aboriginal, pristine
17	Refused	heritage, legacy, bequest, patrimony, refused
18	Speculative	speculative, conjecture, speculate, guess, predict, anticipate, conjectural, speculative, judgmental, risky, hazardous, venture, assumed, logical, notional, uncertain, high risk
19	Surgery	adjust, change, alter, vary, convert, transform
20	Switch	diversification, exchanging, passage, switch, conversion, many routes, swap, transformation, alteration, shift
21	Temporary	temporary, transient, transitory, ephemeral, provisional, tentative, provisory, momentary, stopgap, substitute, evanescent, fleeting, pro-tempore, unstable
22	Understand	understand, comprehend, accept, explain, fathom, grasp, know, perceive

ผู้วิจัยได้นำคำหลักและคำที่มีความหมายสื่อถึงคำหลักมาจัดเก็บให้อยู่ในรูปแบบของฐานข้อมูล โดยจะกำหนดให้ข้อมูลชุดนี้เป็น “datadict” เพื่อให้ง่ายต่อการนำไปช่วยในการค้นหาข้อเสนอแนะที่มีการแนะนำให้นักพัฒนาทำการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง ในตารางที่ 4.3 จะแสดงโครงสร้างการเก็บข้อมูลของ datadict และในตารางที่ 4.4 จะแสดงข้อมูลการจัดเก็บบางส่วนของ datadict

ตารางที่ 4.3 โครงสร้างการเก็บข้อมูลของ datadict

datadict	
PK	id
	keywords
	synonyms

ตารางที่ 4.4 ตัวอย่างบางส่วนของข้อมูลที่จัดเก็บใน datadict

id	keywords	synonyms
1	Chains	chain
2	Chains	continue
3	Chains	unremitted
4	Chains	successive
5	Chains	connected
6	Chains	continual
7	Chains	concatenation
8	Chains	continuity
9	Chains	progression
10	Chains	sequence
11	Chains	connection
12	Chains	link
13	Chains	connect
14	Clump	clump
15	Clump	cluster
16	Clump	collection
17	Clump	bloc
18	Clump	congestion
19	Clump	suite
20	Clump	chunk

4.2 ผลที่ได้จากการค้นหาประเภทร่องรอยโค้ดที่ไม่ดีเพิ่มเติม

ผู้วิจัยได้ทำการค้นหาประเภทร่องรอยโค้ดที่ไม่ดี ที่เกิดขึ้นในโครงการโอเพนซอร์สทั้ง 2 โครงการ เพื่อนำมาเป็นร่องรอยโค้ดที่ไม่ดีประเภทใหม่เพิ่มจากเดิมที่มีอยู่แล้ว 22 ประเภท โดยการใช้อัลกอริทึม LDA ผ่านการเรียกใช้งาน package lda ในโปรแกรม R ในรูปที่ 4.1 จะแสดงรายละเอียดการใช้อัลกอริทึม LDA ในโปรแกรม R

```
1    > require(lda)
2    Loading required package: lda
3    > corpusLDA <- lexicalize(docs)
4    > ldaModel<-lda.collapsed.gibbs.sampler(corpusLDA$documents,K=5,
5      vocab=corpusLDA$vocab,burnin=9999,num.iterations=10000,alpha=1
       ,eta=0.1)
6    > top.words <- top.topic.words(ldaModel$topics, 10, by.score=TRUE)
7    > print(top.words)
8           [,1]          [,2]          [,3]          [,4]          [,5]
9    [1,]   "apply"       "code"        "transform"   "object"      "copy"
10   [2,]   "original"    "loop"        "switch"      "core"        "duplicate"
11   [3,]   "parameter"   "function"    "adjust"      "connect"     "break"
12   [4,]   "name"        "lost"        "main"        "follow"      "delete"
13   [5,]   "confuse"     "variable"    "subclass"    "dead"        "move"
14   [6,]   "default"     "set"         "different"   "test"        "accept"
15   [7,]   "attribute"   "process"     "mean"        "call"        "temporary"
16   [8,]   "know"        "library"     "incorrect"   "comment"     "clear"
17   [9,]   "format"      "maintain"    "cluster      "class"       "scope"
18   [10,]  "repeat"      "data"        "size"        "detail"      "great"
```

รูปที่ 4.1 รายละเอียดการใช้อัลกอริทึม LDA ในโปรแกรม R

จากชุดคำสั่งในรูปที่ 4.1 ประกอบด้วยคำสั่งที่ใช้ดังรายละเอียดต่อไปนี้

บรรทัดที่ 1 : จะเป็นจุดเริ่มต้นในการเรียกใช้งาน package lda

บรรทัดที่ 2 : เป็นการแสดงสถานะให้เห็นว่ากำลังทำการเรียกใช้ package lda

บรรทัดที่ 3 : เป็นการเลือกเอกสารที่ต้องการโดยใช้ฟังก์ชัน “`lexicalize(docs)`” เอกสารในการวิจัยครั้งนี้จะเป็นข้อมูลเกี่ยวกับข้อเสนอแนะของผู้ตรวจทานที่ให้คำแนะนำเกี่ยวกับการปรับปรุงหรือแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง จากรูปที่ 4.1 ผู้วิจัยได้ทำการกำหนดให้ข้อมูลที่จะนำมาผ่านกระบวนการของอัลกอริทึม LDA อยู่ในตัวแปรที่มีชื่อว่า “`corpusLDA`”

บรรทัดที่ 4-5 : จะเข้าสู่ขั้นตอนการกำหนดค่าของพารามิเตอร์ต่าง ๆ ในการค้นหาคำหลักที่ซ่อนอยู่ในข้อเสนอแนะ จะทำการจัดกลุ่มของหัวข้อที่พบในข้อเสนอแนะที่มีความเกี่ยวข้องกันให้มาอยู่ในกลุ่มเดียวกัน โดยจะทำการแบ่งเป็น 5 กลุ่ม (`K=5`) มีการกำหนดให้ทำการสุ่มตัวอย่างมาทั้งหมด 9,999 ตัวอย่าง (`burnin=9999` โดยที่ค่าของ `burnin` จะไม่มีการนับซ้ำกับค่าของ

num.iterations) และได้กำหนดการค้นหาเป็น 10,000 ครั้ง (num.iterations=10000) เพื่อทำการสุ่มเก็บข้อมูลทั้งหมดในเอกสาร เนื่องจากข้อมูลของข้อเสนอแนะทั้งหมดมี 11,808 ข้อเสนอแนะ จึงต้องการที่จะให้มีการค้นหาอย่างละเอียด ยิ่งมีการกำหนดจำนวนของการค้นหาเยอะก็จะยิ่งผ่านการตรวจสอบหลาย ๆ รอบ จะส่งผลให้แสดงเฉพาะข้อมูลที่มีความเกี่ยวข้องกันเท่านั้น นอกจากนี้ยังมีการกำหนดค่าของเอกสารก่อนที่จะมีการกระจายตัวไปยังคำหลัก (alpha=1) และค่าของคำหลักก่อนที่จะมีการกระจายตัวไปยังคำต่าง ๆ ที่ปรากฏอยู่ในข้อเสนอแนะ (eta=0.1)

บรรทัดที่ 6 : เข้าสู่การคำนวนในการค้นหาคำหลักที่ซ่อนอยู่ของอัลกอริทึม LDA คือ $K \times V$ (K = จำนวนของคำหลัก V = จำนวนของคำในคำศัพท์) ที่แต่ละรายการจะเป็นตัวเลขความน่าจะเป็นที่เกิดขึ้น โดยในขั้นตอนนี้ได้มีการกำหนดให้แสดง 10 อันดับของคำหลักที่เจอ โดยเรียงลำดับตามคะแนนที่เกิดขึ้นจากอัลกอริทึม LDA

บรรทัดที่ 7 : เป็นขั้นตอนในการแสดงผลลัพธ์

บรรทัดที่ 8–18 : เป็นผลลัพธ์ที่ได้จากการค้นหาร่องรอยโค้ดที่ไม่ดีประเภทใหม่

จากการใช้อัลกอริทึม LDA จะสังเกตเห็นว่ามีคำที่มีความหมายสื่อถึงคำหลักปรากฏอยู่ในผลลัพธ์ครั้งนี้ด้วย นั่นเป็นการยืนยันอย่างหนึ่งว่าข้อมูลที่อยู่ในตารางที่ 4.2 มีความถูกต้อง ผู้วิจัยจะไม่นำคำที่มีความหมายสื่อถึงคำหลักและคำที่ไม่ได้แสดงอาการ สภาพ หรือการกระทำต่าง ๆ เช่น Data, Class, Object,...etc. มาเป็นร่องรอยโค้ดที่ไม่ดีประเภทใหม่ ในรูปที่ 4.2 จะแสดงประเภทร่องรอยโค้ดที่ไม่ดีประเภทใหม่ที่ค้นพบจากการใช้อัลกอริทึม LDA ซึ่งผ่านการพิจารณาโดยใช้หลักการข้างต้น

```
1   > require(lda)
2   Loading required package: lda
3   > corpusLDA <- lexicalize(docs)
4   > ldaModel<-lda.collapsed.gibbs.sampler(corpusLDA$documents,K=5,
5     vocab=corpusLDA$vocab,burnin=9000,num.iterations=10000,alpha=1,
      eta=0.1)
6   > top.words <- top.topic.words(ldaModel$topics, 10, by.score=TRUE)
7   > print(top.words)
8          [,1]          [,2]          [,3]          [,4]          [,5]
9   [1,]   "apply"       "code"        "transform"   "object"      "copy"
10  [2,]   "original"    "loop"        "switch"      "core"        "duplicate"
11  [3,]   "parameter"   "function"    "adjust"      "connect"     "break"
12  [4,]   "name"        "lost"        "main"        "follow"      "delete"
13  [5,]   "confuse"     "variable"    "subclass"    "dead"        "move"
14  [6,]   "default"     "set"         "different"   "test"        "accept"
15  [7,]   "attribute"   "process"     "mean"        "call"        "temporary"
16  [8,]   "know"        "library"     "incorrect"   "comment"     "clear"
17  [9,]   "format"      "maintain"    "cluster      "class"       "scope"
18  [10,]  "repeat"      "data"        "size"        "detail"      "great"
```

รูปที่ 4.2 ร่องรอยโค้ดที่ไม่ดีที่ค้นพบจากการใช้อัลกอริทึม LDA

ประเภทร่องรอยโค้ดที่ไม่ดีที่ค้นพบจากการใช้อัลกอริทึม LDA ในการวิจัยในครั้งนี้มี 3 ประเภท ได้แก่ Break, Confuse และ Loop เพื่อให้ได้มาซึ่งคำนิยามหรือคำกำจัดความของประเภทร่องรอยโค้ดที่ไม่ดีที่เกิดขึ้นใหม่ ผู้วิจัยได้นำข้อมูลของข้อเสนอแนะที่ผู้ตรวจทานโค้ดได้ให้แนะนำแก่นักพัฒนาให้ทำการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริงมาทำการประมวลผลอีกครั้งโดยใช้คำสั่ง

```
> findFreqTerms(docs.tdm, lowfreq=100)
```

ซึ่งคำสั่งนี้จะช่วยทำการวิเคราะห์รายการของคำร่วมที่มักจะเกิดขึ้นกับร่องรอยโค้ดที่ไม่ดีทั้ง 3 ประเภท โดยการตรวจสอบจากความถี่ที่พบจากชุดของคำสั่งผู้วิจัยได้ทำการกำหนดให้แสดงคำที่เกิดขึ้นร่วมกับคำหลักอย่างน้อย 100 ครั้ง ในตารางที่ 4.5 จะแสดงผลของคำร่วมที่มักจะเกิดขึ้นกับร่องรอยโค้ดที่ไม่ดีทั้ง 3 ประเภท

ตารางที่ 4.5 คำร่วมที่มักจะเกิดขึ้นกับร่องรอยโค้ดที่ไม่ดีประเภทใหม่ที่ค้นพบ

ร่องรอยโค้ดที่ไม่ดีที่ค้นพบ	คำร่วมที่มักจะเกิดขึ้นกับร่องรอยโค้ดที่ไม่ดีที่ค้นพบ
Break	Compatibility, Code, Component, Structure
Confuse	Attribute ,Value, Reduce, Stable liberty
Loop	Flow, Function, Method, Schedule, Resource, Test

หลังจากที่ได้คำร่วมที่เกิดขึ้นกับคำหลักทั้ง 3 เรียบร้อยแล้ว ผู้วิจัยจะนำข้อเสนอแนะที่ผู้ตรวจทานโค้ดได้ระบุไว้เกี่ยวกับคำหลักของทั้ง 3 ประเภท มาพิจารณาร่วมกับผลที่ได้จากการค้นหาคำร่วมที่มักจะเกิดขึ้นกับคำหลักอีกครั้ง ผู้วิจัยได้ให้คำจำกัดความหรือนิยามของร่องรอยโค้ดที่ไม่ดีทั้ง 3 ประเภทดังนี้

- Break code เป็นลักษณะของโค้ดที่ผ่านการแก้ไข หรือทำการปรับเปลี่ยนส่วนประกอบต่าง ๆ ภายในซอร์สโค้ด แล้วทำให้เกิดช่องว่างจำนวนมากในซอร์สโค้ด อาจจะส่งผลให้เกิดการหยุดทำงาน หรือลดประสิทธิภาพในการรันผลลัพธ์
- Attribute confuse เป็นลักษณะการทำงานที่มีการระบุค่าต่าง ๆ ไว้เพื่อให้ครอบคลุมในทุก ๆ ฟังก์ชันที่ต้องการใช้งาน แต่อาจจะไม่ได้นำไปใช้งานทำให้เกิดความสับสนในการเรียกใช้งานควรจะประกาศหรือกำหนดค่าเหล่านั้นเมื่อมีความจำเป็นที่จะต้องใช้งานจริง ๆ

- Function loop เป็นลักษณะของการเรียกใช้งานฟังก์ชันที่มีการทำงานแบบวนลูปอาจจะทำให้เกิดเงื่อนไขที่ซับซ้อนหากมีการเรียกใช้ฟังก์ชันที่มีลูปเยอะอาจจะทำให้เกิดปัญหาในอนาคต ยากต่อการแก้ไขและยังทำให้ยุ่งยากต่อการทดสอบอีกด้วย

ร่องรอยโค้ดที่ไม่ดี ที่ผู้วิจัยได้ทำการค้นพบทั้ง 3 ประเภทนี้ จะนำไปใช้ในการค้นหาข้อเสนอแนะของผู้ตรวจทาน เพื่อให้ข้อมูลที่มีความครอบคลุมเกี่ยวกับคำหลักทั้ง 3 ประเภท ผู้วิจัยได้นำโปรแกรม R ซอฟต์แวร์เวิร์ดเน็ตและพจนานุกรม มาช่วยในการค้นหาคำที่มีความหมายสื่อถึงคำหลักให้กับร่องรอยโค้ดที่ไม่ดีทั้ง 3 ประเภท แสดงดังตารางที่ 4.6

ตารางที่ 4.6 ผลของการค้นหาคำที่มีความหมายสื่อถึงคำหลักของร่องรอยโค้ดที่ไม่ดีประเภทใหม่ที่ค้นพบ

ลำดับที่	คำหลัก	คำที่มีความหมายสื่อถึงคำหลัก
1	Break	break, destroy, ruin, disrupt, demolish
2	Confuse	confuse, baffle, confound, disorient, mystify
3	Loop	loop, coil, convolution, curl, knot, noose, twist

4.3 ผลที่ได้จากการค้นหาข้อเสนอแนะที่ผู้ตรวจทานโค้ดได้ให้คำแนะนำ

ผู้วิจัยได้ทำการพัฒนาคำสั่ง SQL ที่มีคำหลักอยู่อย่างน้อย 1 คำที่มีการกล่าวถึงร่องรอยของโค้ดที่ไม่ดีอยู่อย่างน้อย 1 คำจากฐานข้อมูล และได้ทำการพัฒนาชุดคำสั่ง SQL ออกเป็นชุดคำสั่งย่อยๆ เพื่อทำการค้นหาคำแนะนำของผู้ตรวจทานโค้ดโดยใช้เงื่อนไขของเวลาที่ผู้ตรวจทานโค้ดได้ให้คำแนะนำเกี่ยวกับการปรับปรุงซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง ชุดคำสั่งที่จะยกตัวอย่างต่อไปนี้ เป็นคำสั่งที่สามารถนำไปใช้ได้กับทุก ๆ คำหลักที่ต้องการจะค้นหาคำแนะนำของผู้ตรวจทานโค้ด เพียงทำการเปลี่ยนชื่อตัวแปรในบางตำแหน่งก็สามารถทำการค้นหาคำหลัก คำอื่น ๆ ได้ในรูปที่ 4.3 – 4.4 จะเป็นตัวอย่างของคำสั่ง SQL ที่ใช้กับระบบฐานข้อมูล MySQL ที่ได้ทำการพัฒนาในการค้นหาคำแนะนำของผู้ตรวจทานโค้ดในโครงการ OpenStack

```
1  CREATE DEFINER =`apatta`@`%`PROCEDURE `extractOpenStack`(IN var1
   varchar(100))
2         DETERMINISTIC
3  BEGIN
4         DECLARE temp_synonyms varchar(255);
5
6         DECLARE v_not_found BOOL default FALSE;
7         DECLARE data_cursor CURSOR FOR
8         SELECT synonyms FROM datadict WHERE keywords = var1;
9
10        DECLARE continue handler for not found set v_not_found
          := TRUE;
11
12        OPEN data_cursor;
13                    cursor_loop: loop
14              FETCH data_cursor INTO temp_synonyms;
15              if v_not_found then
16                 leave cursor_loop;
17              end if;
18               SET@sql_text1 = CONCAT('''%',temp_synonyms,'%''');
19              SET@sql_text2 = CONCAT('INSERT INTO tempOpenStack
                SELECT id,''',
20              var1,''',message,''OpenStack'',''',temp_synonyms,''',
                created_on,
21              reviewer,author FROM comment_detail_Openstack where
                message like
22              ',@sql_text1,';');
23
24              SELECT @sql_text2;
25                    PREPARE stmt1 FROM @sql_text2;
26
27                    EXECUTE stmt1;
28              DEALLOCATE PREPARE stmt1;
29
30
31        end loop;
32
33        CLOSE data_cursor;
34 #Routine body goes here...
35
36 END;
```

รูปที่ 4.3 คำสั่ง SQL ที่ได้ทำการพัฒนาที่มีคำหลักอยู่อย่างน้อย 1 คำ ที่มีการกล่าวถึงร่องรอยของที่ไม่ดีอยู่อย่างน้อย 1 คำจากฐานข้อมูล

หลักการทำงานของชุดคำสั่งในรูปที่ 4.3 คือ ทำการค้นหาคำแนะนำของผู้ตรวจทานโค้ดที่ได้รวบรวมมาจากระบบเกอริต โดยจะนำข้อมูลของคำที่มีความหมายสื่อถึงคำหลัก “synonyms” ที่จัดเก็บอยู่ในตารางข้อมูล “datadict” มาค้นหาคำแนะนำของผู้ตรวจทานโค้ด

ในการค้นหาแต่ละครั้งจะทำการกำหนดคำหลักมาช่วยในการค้นหาซึ่งในคำหลักเหล่านั้นก็จะประกอบไปด้วยคำที่มีความหมายสื่อถึงคำหลักหลาย ๆ คำที่ถูกจัดเก็บไว้ใน datadict จากคำสั่งในรูปที่ 4.3 จะทำการอธิบายรายละเอียดของคำสั่งที่ใช้ในแต่ละบรรทัดดังนี้

บรรทัดที่ 2-1 : จะเป็นรายละเอียดการสร้างไฟล์ชุดคำสั่ง

บรรทัดที่ 3 : จะเป็นจุดเริ่มต้นของพื้นที่ในการประกาศค่าการใช้งานต่าง ๆ

บรรทัดที่ 5-4 : เป็นการประกาศตัวแปรที่มีชื่อว่า “temp_synonyms” จะทำหน้าที่ในการเก็บรวบรวมข้อมูลของคำที่มีความหมายสื่อถึงคำหลักของคำหลักแต่ละคำ ที่อยู่ในตาราง “datadict” เพื่อนำไปค้นหาคำแนะนำของผู้ตรวจทานโค้ดที่มีการแนะนำให้นักพัฒนาทำการแก้ไขหรือปรับปรุงซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง

บรรทัดที่ 6 : ประกาศตัวแปรที่มีชื่อว่า “v_not_found” เพื่อทำหน้าที่ในการตรวจสอบการบันทึกข้อมูล

บรรทัดที่ 7 : ประกาศตัวแปรชนิด Cursor ที่มีชื่อว่า “data_cursor” จะทำหน้าที่ในการเก็บค่าที่ได้จากการ SELECT ทั้งหมดมาจัดเก็บไว้ใน “data_cursor”

บรรทัดที่ 8 : เป็นการค้นหาคำที่มีความหมายสื่อถึงคำหลักที่อยู่ในคำหลักแต่ละคำที่จัดเก็บอยู่ในตารางข้อมูล datadict ซึ่งในบรรทัดนี้จะต้องทำการระบุคำหลักที่ต้องการจะค้นหาในตำแหน่งของตัวแปร “var1” ให้กลายเป็นคำหลักที่ต้องการจะค้นหา

บรรทัดที่ 10 : ทำการตรวจสอบการบันทึกข้อมูลแบบต่อเนื่องโดยกำหนดให้ “v_not_found:= TRUE” เมื่อไม่มีข้อมูลที่จะทำการบันทึกแล้ว

บรรทัดที่ 12-14 : เปิด “data_cursor” เพื่อนำค่าตัวแปร Cursor ที่ได้ไปทำการค้นหาข้อมูลวนไปเรื่อยๆ และนำค่าที่ได้ไปใส่ไว้ใน “temp_synonyms”

บรรทัดที่ 15-17 : เมื่อทำการบันทึกข้อมูลเสร็จเรียบร้อยแล้วให้ออกจาก loop ทันที

บรรทัดที่ 18 : จะทำหน้าที่ในการเก็บรวบรวมข้อมูลของ “temp_synonyms” ไว้ในตัวแปร “@sql_text1” เพื่อนำข้อมูลชุดนี้ไปรวมกับข้อมูลจากตารางอื่น ๆ

บรรทัดที่ 22-19 : ในขั้นตอนนี้จะเป็นการค้นหาข้อมูลในตาราง “comment_detail_Openstack” ซึ่งข้อมูลในตารางนี้เป็นการรวบรวมคำแนะนำที่ได้มาจากเกอริต โดยจะทำการนำข้อมูล id คำหลัก คำแนะนำของผู้ตรวจทานโค้ด โครงการที่ต้องการ คำที่มีความหมายสื่อถึงคำหลักที่ใช้ ช่วงเวลาที่ผู้ตรวจทานโค้ดได้ให้คำแนะนำ ชื่อผู้ตรวจทานโค้ด ชื่อเจ้าของซอร์สโค้ด โดยจะทำการเลือกเฉพาะคำแนะนำที่มีการใช้คำที่มีความหมายสื่อถึงคำหลักตามที่ได้ระบุไว้ใน “temp_synonyms” โดยเรียกใช้งานผ่านตัวแปร “@sql_text1” และนำข้อมูลทั้งหมดมา

รวบรวมไว้ในตาราง “tempOpenStack” ซึ่งจะถูกจัดเก็บไว้ในตัวแปร “@sql_text2” เพื่อให้ง่ายต่อการเรียกใช้ในลำดับต่อไป

บรรทัดที่ 28-24 : ทำการตรวจสอบความถูกต้องของคำสั่ง SQL ที่อยู่ในตัวแปร “@sql_text2” เพื่อให้แน่ใจว่าคำสั่งที่ใช้มีความถูกต้องก่อนจะทำการประมวลผล

บรรทัดที่ 31 : จบการทำงานของคำสั่ง loop

บรรทัดที่ 33 : ปิดการใช้งาน Cursor

บรรทัดที่ 36 : จบการทำงานของคำสั่งชุดนี้

นอกจากนี้ผู้วิจัยได้ทำการพัฒนาชุดคำสั่ง SQL ออกเป็นชุดคำสั่งย่อย ๆ เพื่อทำการค้นหาคำแนะนำของผู้ตรวจทานโค้ด โดยใช้เงื่อนไขของเวลาที่ผู้ตรวจทานโค้ดได้ให้คำแนะนำเกี่ยวกับการปรับปรุงซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง แสดงดังรูปที่ 4.4

```
1   SELECT keywords, synonyms, message, reviewer, author, created_on
2   FROM   tempOpenStack
3   WHERE  keywords = 'Duplicate'
4   AND    created_on Like '2014%'
```

รูปที่ 4.4 ชุดคำสั่ง SQL ย่อย ๆ โดยใช้เงื่อนไขของเวลา

คำสั่งในรูปที่ 4.4 จะเป็นตัวอย่างของการค้นหาคำแนะนำของผู้ตรวจทานโค้ดที่มีการกล่าวถึงร่องรอยโค้ดที่ไม่ดีในกลุ่มของคำหลัก Duplicate ที่เกิดขึ้นในปี ค.ศ. 2014 จากตัวอย่างหากต้องการจะค้นหาคำหลัก คำอื่น ๆ สามารถทำการเปลี่ยนคำหลักที่ต้องการจะค้นหาได้ในบรรทัดที่ 3 และในกรณีที่ต้องการจะค้นหาคำแนะนำในช่วงปีอื่น ๆ ก็สามารถทำการเปลี่ยนข้อมูลของปีได้ในบรรทัดที่ 4

หลังจากที่ได้ใช้ชุดคำสั่งที่พัฒนาเหล่านี้มาทำการรวบรวมคำแนะนำของผู้ตรวจทานโค้ดที่มีการแนะนำให้นักพัฒนาทำการแก้ไขปรับปรุงซอร์สโค้ดโดยผ่านการใช้คำหลักทั้ง 25 ประเภท และเป็นคำแนะนำที่อยู่ในช่วงปี ค.ศ. 2011- 2015 มาจัดเก็บไว้ในรูปแบบของฐานข้อมูล ในรูปที่ 4.5 จะแสดงข้อมูลบางส่วนที่ได้จากการค้นหาคำแนะนำ

id	keywords	message	project	synonyms	created_on	reviewer	author
e449f58c	Duplicate	L55-L59 should are duplicated across all meth	OpenStack	Duplicate	2014-11-18		
259ac8cc	Duplicate	Armando, we created the infra manual since n	OpenStack	Duplicate	2014-12-13		
fdd16352	Duplicate	These methods seem to have a lot of duplicat	OpenStack	Duplicate	2014-12-20		
f653ac2f	Duplicate	type_vxlan and type_gre first versions were sir	OpenStack	Duplicate	2014-10-25		
5e5bd853	Duplicate	this method is duplicated with the one in Poli	OpenStack	Duplicate	2014-10-06		
1d93ab59	Duplicate	@Alistair, Thanks for the review. Please see th	OpenStack	Duplicate	2014-12-16		
abd49dd	Duplicate	I created duplicate flavor `ram64' and run scen	OpenStack	Duplicate	2014-12-23		
3f19c0b9	Duplicate	What about other kinds of exceptions that can	OpenStack	Duplicate	2014-12-26		
fc78c6f0	Duplicate	All of the content on this page is duplicated el	OpenStack	Duplicate	2014-12-18		
6aceb8d	Duplicate	hard-coding the filename to heat.yaml will m	OpenStack	Repeatedly	2014-12-03		
ac2fec59	Duplicate	Repeatedly calling 'shutdown' in the body of t	OpenStack	Repeatedly	2014-01-29		
0dd1d0a	Duplicate	After reading the credentials snippet several ti	OpenStack	Repeatedly	2014-04-09		
ee54d311	Duplicate	I didn't trace where tmp_path came from, but	OpenStack	Repeatedly	2013-08-03		
656b83b	Duplicate	If it can be run repeatedly w/o causing proble	OpenStack	Repeatedly	2013-03-19		
7137b0d	Duplicate	Consider defining the pattern and compiling t	OpenStack	Repeatedly	2012-03-08		
735df9a1	Duplicate	let's actually call lower() on both sides of the c	OpenStack	Repeatedly	2012-02-13		
4859d68f	Duplicate	Missing boiler plate license headerBeing that t	OpenStack	Repeatedly	2015-08-31		
7caccacf	Duplicate	Now that this is a class method, it's not using	OpenStack	Repeatedly	2015-07-30		
cff0c1a6	Duplicate	this is necessary only for the first iteration, rigl	OpenStack	Repeatedly	2015-08-12		
68f75237	Duplicate	Agree. At least don't do split() on the same str	OpenStack	Repeatedly	2015-08-12		
6bd9f54f	Duplicate	You're repeating the same thing repeatedly wi	OpenStack	Repeatedly	2015-07-01		
b1470766	Duplicate	This pattern keeps showing up repeatedly. Ma	OpenStack	Repeatedly	2015-07-08		
92130774	Duplicate	It' better to define a constant if we use it repea	OpenStack	Repeatedly	2015-06-30		
ee449d98	Duplicate	My concern is not with actually booting from	OpenStack	Repeatedly	2015-02-17		
d8129456	Duplicate	Ah, nice. The file caching prevents this from b	OpenStack	Repeatedly	2015-02-16		
61d74451	Duplicate	We can make this less repetitive by just addinç	OpenStack	Repetitive	2014-10-09		
7ad5045a	Duplicate	Yeah, this is pretty repetitive and maybe could	OpenStack	Repetitive	2014-05-13		
f5cdd945	Duplicate	Seems like these functions will all be repetitive	OpenStack	Repetitive	2014-01-14		

**หมายเหตุ: ผู้วิจัยต้องปกปิดชื่อของ Reviewer และ Author เพื่อเป็นการรักษาสิทธิส่วนบุคคล*

รูปที่ 4.5 ตัวอย่างของข้อมูลที่ได้จากการค้นหาคำแนะนำ

ซึ่งข้อมูลที่ได้จากการค้นหาคำแนะนำจะประกอบด้วยชุดข้อมูลดังนี้

- Id หมายเลขลำดับข้อมูล
- Keywords เป็นคำหลักที่ใช้ในการค้นหา
- Message เป็นรายละเอียดของคำแนะนำหรือข้อเสนอแนะที่ผู้ตรวจทานได้ให้คำแนะนำแก่นักพัฒนา
- Project เป็นข้อมูลชื่อของโครงการที่ได้ทำการค้นหา
- Synonyms เป็นการแสดงคำที่มีความหมายสื่อถึงคำหลักที่มีการค้นพบในคำแนะนำ
- Created on เป็นรายละเอียดวันเวลาของการส่งคำร้องขอหรือการตรวจทาน
- Reviewer เป็นข้อมูลชื่อของผู้ตรวจทานโค้ด
- Author เป็นข้อมูลชื่อของเจ้าของซอร์สโค้ดที่มีการร้องขอให้ผู้ตรวจทานโค้ดช่วยทำการตรวจทานโค้ด

สำหรับข้อมูลที่ได้จากการเก็บรวบรวมคำแนะนำของผู้ตรวจทานโค้ดเพื่อนำมาวิเคราะห์ผลในตารางที่ 4.7 จะแสดงรายละเอียดข้อมูลทั้งหมดของโครงการโอเพนซอร์ส ทั้ง 2 โครงการที่ได้ทำการรวบรวมมาจากระบบเกอริต และในตารางที่ 4.8 จะแสดงรายละเอียดของข้อมูลที่ได้ผ่านกระบวนการทำเหมืองข้อความเพื่อทำการคัดกรองหาข้อเสนอแนะที่ผู้ตรวจทานโค้ดได้ให้คำแนะนำของโครงการโอเพนซอร์ส ทั้ง 2 โครงการ

ตารางที่ 4.7 รายละเอียดข้อมูลของโครงการโอเพนซอร์สทั้ง 2 โครงการตลอดช่วงปี ค.ศ. 2011 - 2015

	โครงการ OpenStack	โครงการ WikiMedia
จำนวนผู้ตรวจทานโค้ด	5,694	988
จำนวนคำร้องขอให้ตรวจทานโค้ด	179,507	208,610
จำนวนข้อเสนอแนะทั้งหมด	822,845	26,947

ตารางที่ 4.8 รายละเอียดข้อมูลของโครงการโอเพนซอร์สทั้ง 2 โครงการ ที่มีการแนะนำเกี่ยวกับการปรับปรุงแก้ไขซอร์สโค้ดตลอดช่วงปี ค.ศ. 2011-2015

	โครงการ OpenStack	โครงการ WikiMedia
จำนวนผู้ตรวจทานโค้ด	1,233	149
จำนวนคำร้องขอให้ตรวจทานโค้ด	1,683	210
จำนวนข้อเสนอแนะทั้งหมด	13,461	1,622

จากตารางที่ 4.7 และตารางที่ 4.8 ทำให้เห็นว่าจากจำนวนของผู้ตรวจทานโค้ดทั้งหมดมีผู้ตรวจทานโค้ดที่ให้ความสำคัญเกี่ยวกับการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริงในโครงการ OpenStack คิดเป็น 21.65% และโครงการ WikiMedia คิดเป็น 15.08% ถือว่ายังเป็นจำนวนที่น้อยแต่เมื่อพิจารณาจำนวนข้อเสนอแนะที่เกี่ยวกับการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง จะเห็นว่าผู้ตรวจทาน 1 คน จะมีการให้ข้อเสนอแนะเกี่ยวกับการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริงมากกว่า 1 ข้อเสนอแนะ จากข้อมูลดังกล่าวจะสามารถบ่งบอกได้ว่าข้อเสนอแนะมักจะเกิดจากคำแนะนำของผู้ตรวจทานโค้ดคนเดิมที่คอยให้คำแนะนำเกี่ยวกับการนำรีแฟคทอริงไปช่วยในการพัฒนาซอฟต์แวร์

4.3.1 ผลของการค้นหาคำแนะนำ

ผลที่ได้จากการค้นหาข้อเสนอแนะที่ผู้ตรวจทานโค้ดได้ให้คำแนะนำเกี่ยวกับการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง ผู้วิจัยได้นำข้อมูลดังกล่าวไปประมวลผลในโปรแกรม R เพื่อทำการนับความถี่ที่เกิดขึ้นและได้ทำการจัดเก็บในรูปแบบของ Document Term Matrix โดยมีโครงสร้างที่เป็นรูปแบบสดมภ์และแถว โดยข้อมูลในแถวจะเป็นคำหลักที่ใช้ในการค้นหาข้อเสนอแนะที่ได้จากกระบวนการตรวจทานโค้ด และสดมภ์จะเป็นช่วงระยะเวลาของแต่ละปีที่ทำการค้นหาข้อเสนอแนะ จำนวนความถี่ของคำหลักที่ปรากฏสำหรับร่องรอยโค้ดที่ไม่ดีในแต่ละปีจะแสดงอยู่ในช่องตาราง ยิ่งมีความถี่มากก็แสดงว่าผู้ตรวจทานโค้ดได้ให้ความสนใจหรือให้ความสำคัญเกี่ยวกับการมุ่งเน้นให้นักพัฒนาทำการปรับปรุงแก้ไขซอร์สโค้ดให้มีคุณภาพด้วยวิธีการทำรีแฟคทอริง ในกรณีที่ความถี่ที่มีค่าเป็น 0 คือ ไม่มีการกล่าวถึงการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง โดยในตารางที่ 4.9 จะเป็นการแสดงจำนวนความถี่ของคำหลักที่ปรากฏทั้งหมดในข้อเสนอแนะของโครงการ OpenStack และในตารางที่ 4.10 จะเป็นการแสดงจำนวนความถี่ของคำหลักที่ปรากฏทั้งหมดในข้อเสนอแนะของโครงการ WikiMedia

ตารางที่ 4.9 จำนวนความถี่ของคำหลักที่ปรากฏทั้งหมดในข้อเสนอแนะของโครงการ OpenStack

ลำดับที่	คำหลัก	ปี				
		2011	2012	2013	2014	2015
1	Chain	0	26	114	228	270
2	Clump	2	37	189	398	483
3	Complex	1	11	77	180	231
4	Dead	0	6	40	104	98
5	Divergent	1	21	78	159	249
6	Duplicate	4	56	363	667	975
7	Envy	4	10	32	68	88
8	Inappropriate	1	8	72	138	193
9	Incomplete	1	1	10	23	22
10	Inheritance	0	0	15	54	50
11	Large	7	38	230	426	663
12	Long	1	32	118	219	320
13	Many Parameters	3	23	164	340	474
14	Meaning	2	47	159	308	456
15	Middle	1	13	52	144	153

ตารางที่ 4.9 จำนวนความถี่ของคำหลักที่ปรากฏทั้งหมดในข้อเสนอแนะของโครงการ OpenStack (ต่อ)

ลำดับที่	คำหลัก	ปี				
		2011	2012	2013	2014	2015
16	Primitive	1	12	38	86	61
17	Refused	0	0	4	2	14
18	Speculative	1	26	73	143	222
19	Surgery	0	6	24	46	48
20	Switch	0	3	36	69	82
21	Temporary	0	12	47	106	128
22	Understand	6	39	113	144	273
23	Break	3	17	47	98	316
24	Confuse	0	3	13	9	25
25	Loop	4	43	168	337	262

ตารางที่ 4.10 จำนวนความถี่ของคำหลักที่ปรากฏทั้งหมดในข้อเสนอแนะของโครงการ WikiMedia

ลำดับที่	คำหลัก	ปี				
		2011	2012	2013	2014	2015
1	Chain	0	0	0	1	33
2	Clump	0	3	3	8	69
3	Complex	0	2	7	31	128
4	Dead	0	0	1	11	18
5	Divergent	0	9	12	8	97
6	Duplicate	0	1	15	17	56
7	Envy	0	3	3	14	34
8	Inappropriate	0	1	12	6	14
9	Incomplete	0	1	0	6	30
10	Inheritance	0	0	2	4	15
11	Large	0	0	4	21	91
12	Long	0	0	0	6	19
13	Many Parameters	0	1	3	18	59
14	Meaning	0	4	22	38	131

ตารางที่ 4.10 จำนวนความถี่ของคำหลักที่ปรากฏทั้งหมดในข้อเสนอแนะของโครงการ WikiMedia (ต่อ)

ลำดับที่	คำหลัก	ปี				
		2011	2012	2013	2014	2015
15	Middle	0	3	17	8	56
16	Primitive	0	0	1	3	21
17	Refused	0	0	0	3	7
18	Speculative	0	2	9	14	96
19	Surgery	0	0	2	5	13
20	Switch	0	0	0	15	21
21	Temporary	0	1	0	0	16
22	Understand	0	0	3	13	87
23	Break	0	2	5	13	44
24	Confuse	0	0	0	0	4
25	Loop	0	1	0	10	35

ผลที่ได้จากการรวบรวมข้อเสนอแนะที่เกิดขึ้นในแต่ละปียังแสดงให้เห็นว่าผู้ตรวจทานโค้ดของทั้ง 2 โครงการ ส่วนใหญ่ไม่ได้ใช้คำหลักโดยตรง ผู้วิจัยได้ทำการสรุปคำที่ผู้ตรวจทานโค้ดทั้ง 2 โครงการ นิยมใช้เพื่อเป็นการบ่งบอกให้นักพัฒนาทำการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริงผ่านประเภทของร่องรอยโค้ดที่ไม่ดีในแต่ละประเภท ดังตารางที่ 4.11 และตารางที่ 4.12 โดยค่าของความนิยม (ร้อยละ) คำนวณมาจาก

$$\frac{k}{\sum k \times 100}\%$$

k : คำที่มีความหมายสื่อถึงคำหลักที่ผู้ตรวจทานใช้แทนคำหลัก

$\sum k$: คำที่มีความหมายสื่อถึงคำหลักทั้งหมดที่ผู้ตรวจทานใช้แทนคำหลัก

ตารางที่ 4.11 คำที่มีความหมายสื่อถึงคำหลักที่ผู้ตรวจทานโค้ดในโครงการ OpenStack นิยมใช้ในการให้คำแนะนำ

ลำดับที่	คำหลัก	คำที่มีความหมายสื่อถึงคำหลัก	ความนิยมในการใช้ (%)
1	Chain	chain	38.71
		connect	25.08
		continue	17.55
2	Clump	lot	52.57
		group	17.31
		cluster	13.53
3	Complex	multiple	46.80
		complex	43.80
4	Dead	dead	67.74
		lost	29.03
5	Divergent	different	90.35
6	Duplicate	duplicate	46.15
		copy	16.08
7	Envy	use	58.42
		apply	35.15
8	Inappropriate	incorrect	69.90
		irrelevant	12.86
		inappropriate	11.17
9	Incomplete	deficient	21.05
		flawed	17.54
		imperfect	10.53
10	Inheritance	hierarchy	26.89
		inherit	26.05
		inheritance	21.01
11	Large	much	45.38
		big , great , large	15.84
12	Long	long	90.87

ตารางที่ 4.11 คำที่มีความหมายสื่อถึงคำหลักที่ผู้ตรวจทานโค้ดในโครงการ OpenStack นิยมใช้ในการให้คำแนะนำ (ต่อ)

ลำดับที่	คำหลัก	คำที่มีความหมายสื่อถึงคำหลัก	ความนิยมในการใช้ (%)
13	Many Parameters	much	61.65
		great	21.02
		many	14.74
14	Meaning	suggestion	36.73
		definition	31.79
15	Middle	between	72.18
		core	15.15
		middle	9.64
16	Primitive	original	83.33
		primitive	16.67
17	Refused	legacy	100.00
18	Speculative	guess	60.43
		assumed	32.47
19	Surgery	convert	48.39
		adjust	16.94
		transform	14.52
20	Switch	switch	50.00
		conversion	25.79
		swap	23.16
21	Temporary	temporary	55.97
		ephemeral	21.50
22	Understand	understand	68.00
		explain	15.65
		accept	14.26
23	Break	break	90.74
24	Confuse	confuse	98.00
25	Loop	loop	97.79

ตารางที่ 4.12 คำที่มีความหมายสื่อถึงคำหลักที่ผู้ตรวจทานโค้ดในโครงการ WikiMedia นิยมใช้ในการให้คำแนะนำ

ลำดับที่	คำหลัก	คำที่มีความหมายสื่อถึงคำหลัก	ความนิยมในการใช้ (%)
1	Chain	chain	52.94
		continue	35.29
		sequence	5.88
2	Clump	lot	68.67
		cluster	16.87
		chunk	8.43
3	Complex	multiple	65.48
		complex	13.69
		mixed	12.50
4	Dead	lost	53.33
		dead	46.67
5	Divergent	different	78.38
6	Duplicate	duplicate	40.45
		copy	23.60
7	Envy	apply	50.00
		use	48.15
8	Inappropriate	incorrect	48.48
		irrelevant	36.36
		inappropriate	9.09
9	Incomplete	partially	27.03
		incomplete	24.32
		partial	21.62
10	Inheritance	legacy	38.10
		inheritance	28.57
		hierarchy	14.29
11	Large	much	37.07
		big	17.24
		great, huge , large	13.79

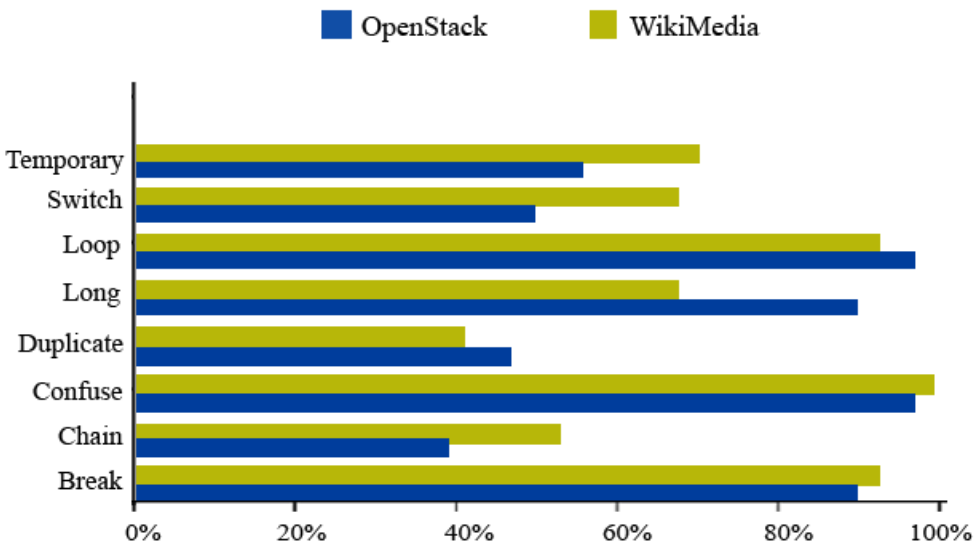
ตารางที่ 4.12 คำที่มีความหมายสื่อถึงคำหลักที่ผู้ตรวจทานโค้ดในโครงการ WikiMedia นิยมใช้ในการให้คำแนะนำ (ต่อ)

ลำดับที่	คำหลัก	คำที่มีความหมายสื่อถึงคำหลัก	ความนิยมในการใช้ (%)
12	Long	long	68.00
		expanded	20.00
		continued	12.00
13	Many Parameters	much	53.09
		many	25.93
		great	19.75
14	Meaning	suggest	34.55
		definition	17.80
		hint and explanation	17.28
15	Middle	between	55.95
		core	28.57
		intermediate	11.90
16	Primitive	original	100.00
17	Refused	legacy	80.00
		heritage	20.00
18	Speculative	guess	89.26
		logical	4.96
		predict	2.48
19	Switch	switch	69.44
		conversion	19.44
		swap	8.33
20	Temporary	temporary	70.59
		ephemeral	17.65
		unstable	11.76
21	Understand	explain	61.17
		accept	30.10
		grasp	4.85
22	Surgery	transform	40.00
		adjust	20.00
23	Break	break	93.75

ตารางที่ 4.12 คำที่มีความหมายสื่อถึงคำหลักที่ผู้ตรวจทานโค้ดในโครงการ WikiMedia นิยมใช้ในการให้คำแนะนำ (ต่อ)

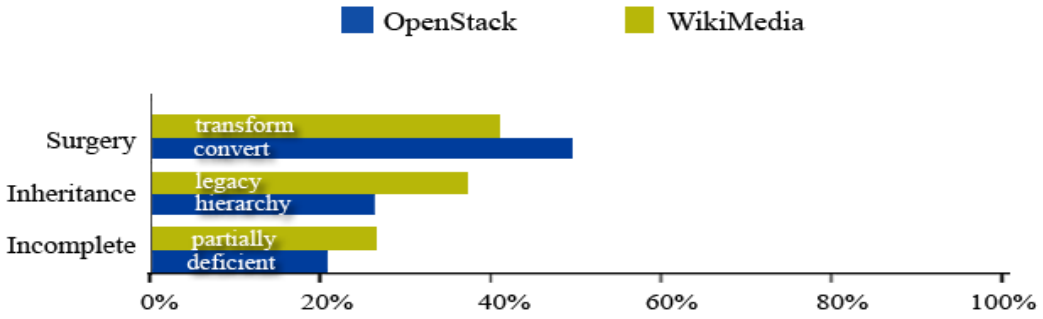
ลำดับที่	คำหลัก	คำที่มีความหมายสื่อถึงคำหลัก	ความนิยมในการใช้ (%)
24	Confuse	confuse	100
25	Loop	loop	93.48

จากตารางที่ 4.11 และตารางที่ 4.12 จะมีคำหลักเพียงไม่กี่คำที่ผู้ตรวจทานโค้ดนิยมใช้ในการให้คำแนะนำที่ตรงตามประเภทของร่องรอยโค้ดที่ไม่ดี ได้แก่ Break, Chain, Confuse, Duplicate, Long, Loop Switch และ Temporary ในรูปที่ 4.6 จะแสดงคำหลักที่ผู้ตรวจทานโค้ดทั้ง 2 โครงการใช้ในการให้คำแนะนำที่ตรงตามประเภทของร่องรอยโค้ดที่ไม่ดี

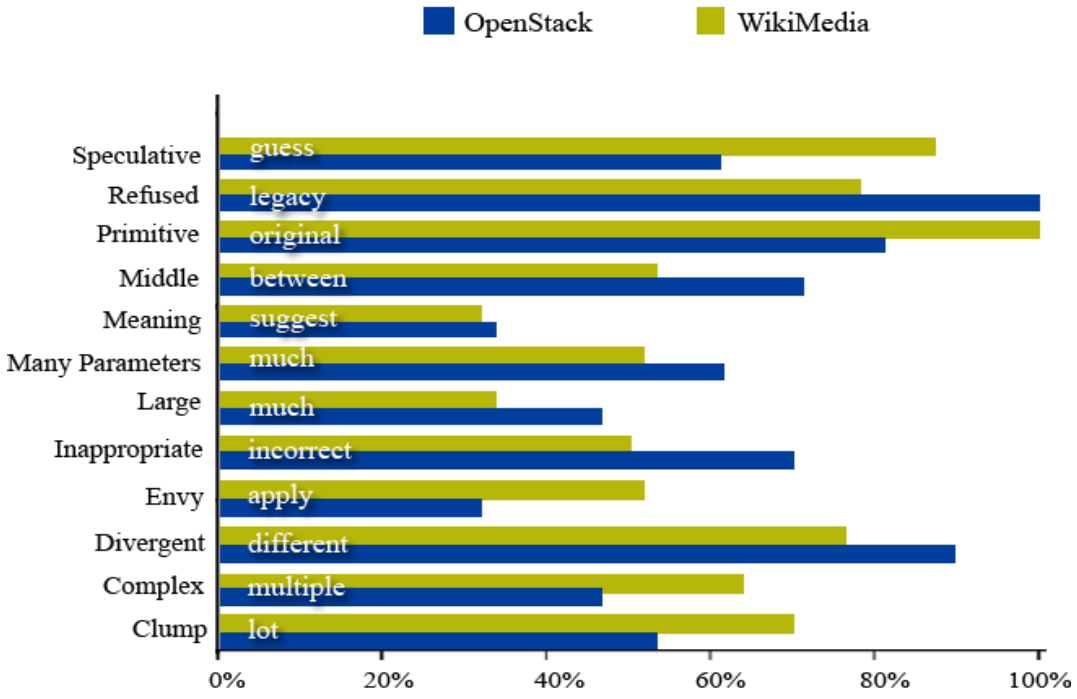


รูปที่ 4.6 คำหลักที่ผู้ตรวจทานโค้ดทั้ง 2 โครงการใช้ในการให้คำแนะนำที่ตรงตามประเภทของร่องรอยโค้ดที่ไม่ดี

จากคำหลัก 25 คำที่ใช้ในการบ่งบอกถึงร่องรอยโค้ดที่ไม่ดีในประเภทต่าง ๆ มีเพียง 8 คำหลักเท่านั้นที่ผู้ตรวจทานโค้ดนิยมใช้ในการให้คำแนะนำ ในรูปที่ 4.7 จะสรุปคำที่มีความหมายสื่อถึงคำหลักที่ผู้ตรวจทานโค้ดใช้ไม่เหมือนกันในการให้คำแนะนำ และรูปที่ 4.8 จะสรุปคำที่มีความหมายสื่อถึงคำหลักที่ผู้ตรวจทานโค้ดทั้ง 2 โครงการใช้ในการให้คำแนะนำเหมือนกัน



รูปที่ 4.7 คำที่มีความหมายสื่อถึงคำหลักที่ผู้ตรวจทานโค้ดทั้ง 2 โครงการใช้ไม่เหมือนกันในการให้คำแนะนำ

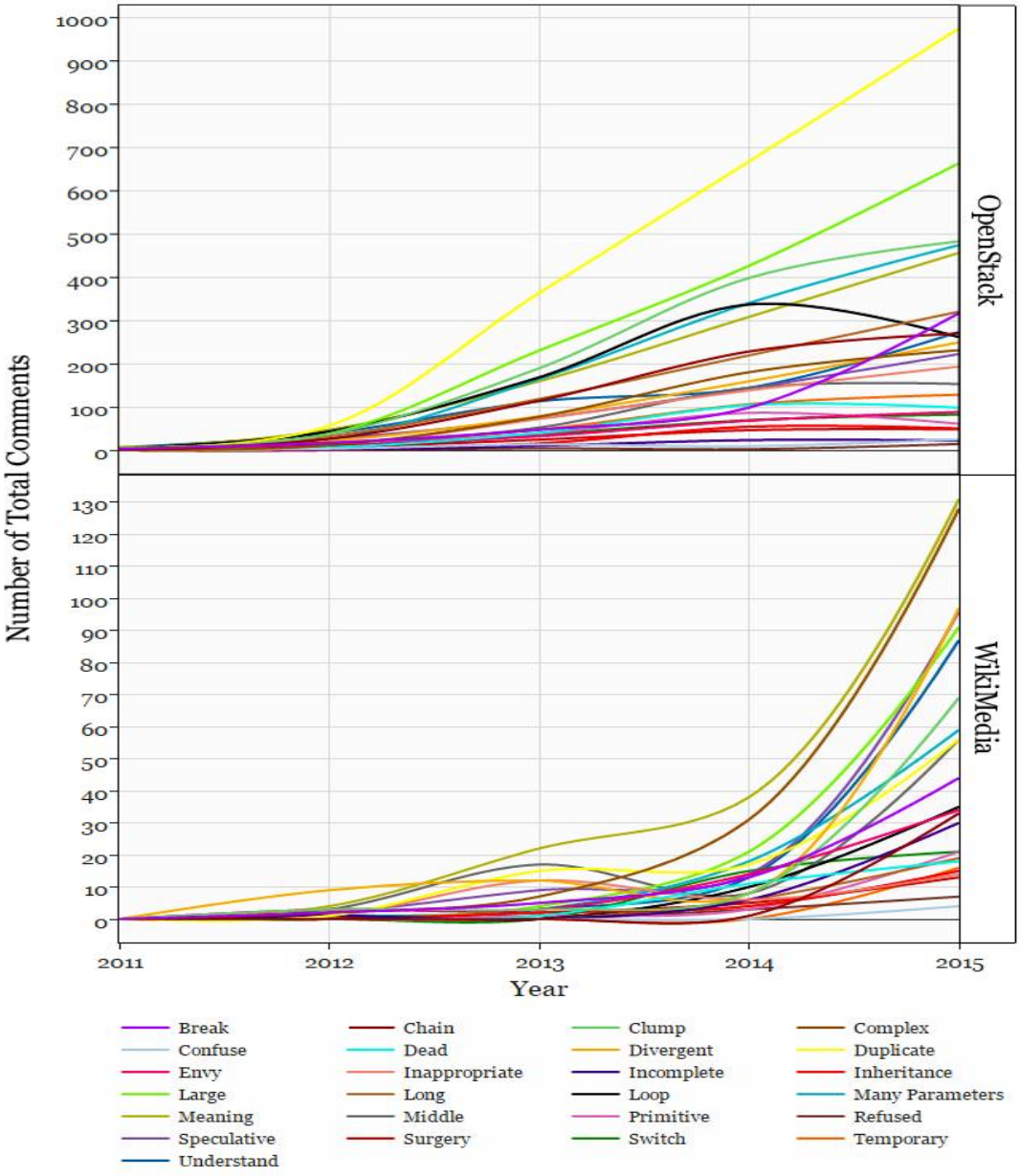


รูปที่ 4.8 คำที่มีความหมายสื่อถึงคำหลักที่ผู้ตรวจทานโค้ดทั้ง 2 โครงการนิยมใช้ในการให้คำแนะนำ

4.4 ผลของการวิเคราะห์ข้อเสนอแนะของผู้ตรวจทานโค้ดทั้ง 2 โครงการ

4.4.1 แนวโน้มของการเกิดร่องรอยโค้ดที่ไม่ดี

งานวิจัยนี้ได้ทำการรวบรวมข้อเสนอแนะของผู้ตรวจทานโค้ดของโครงการซอฟต์แวร์โอเพนซอร์ส 2 โครงการ ได้แก่ OpenStack มีจำนวนข้อเสนอแนะทั้งหมด 822,845 ข้อเสนอแนะ และ WikiMedia มีจำนวนข้อเสนอแนะทั้งหมด 26,947 ข้อเสนอแนะ ผู้วิจัยได้นำข้อเสนอแนะทั้งหมดมาผ่านกระบวนการต่าง ๆ เพื่อทำการคัดกรองหาข้อเสนอแนะที่ผู้ตรวจทานโค้ดได้ให้คำแนะนำเกี่ยวกับการปรับปรุงซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง โดยการนำร่องรอยโค้ดที่ไม่ดี 25 ประเภท มาเป็นช่วยในการค้นหาและทำการรวบรวมข้อเสนอแนะที่เกิดขึ้นในช่วงปี ค.ศ. 2011 – 2015 ซึ่งคำแนะนำที่มีการแนะนำให้นักพัฒนาทำการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริงที่เกิดขึ้นในโครงการ OpenStack มี 13,461ข้อเสนอแนะ และในโครงการ WikiMedia มี 1,622 ข้อเสนอแนะ ในรูปที่ 4.9 จะแสดงแนวโน้มของการเกิดร่องรอยโค้ดที่ไม่ดีในแต่ละปี โดยเส้นกราฟแต่ละสีจะแสดงถึงคำหลักที่ใช้



รูปที่ 4.9 แนวโน้มของการเกิดร่องรอยโค้ดที่ไม่ดี

จากรูปที่ 4.9 แสดงให้เห็นจำนวนข้อเสนอแนะทั้งหมดของร่องรอยโค้ดที่ไม่ดีในแต่ละโครงการโอเพนซอร์สซึ่งมีจำนวนเพิ่มขึ้นเรื่อย ๆ ตั้งแต่ปี ค.ศ. 2011 และมีแนวโน้มที่จะเพิ่มขึ้นอีกในอนาคต ผู้วิจัยได้คาดการณ์จำนวนข้อเสนอแนะที่มีการกล่าวถึงร่องรอยโค้ดที่ไม่ดี ที่เกิดขึ้นในโครงการ OpenStack และโครงการ WikiMedia ด้วยวิธีการทางสถิติที่เรียกว่า การวิเคราะห์การถดถอย (Regression analysis) ซึ่งเป็นวิธีทางสถิติที่ใช้หาความสัมพันธ์ระหว่างตัวแปร 2 ตัวแปร

คือ ตัวแปรอิสระ (Independent variable) และตัวแปรตาม (Dependent variable) เพื่อใช้ในการพยากรณ์แนวโน้มของการเกิดร่องรอยโค้ดที่ไม่ดีในอนาคต

การหาความสัมพันธ์นี้ผู้วิจัยได้กำหนดให้ ปีที่ผู้ตรวจทานโค้ดได้ให้คำแนะนำเกี่ยวกับการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง เป็นตัวแปรอิสระ และจำนวนของข้อเสนอแนะที่ผู้ตรวจทานได้ให้คำแนะนำเกี่ยวกับร่องรอยโค้ดที่ไม่ดี เป็นตัวแปรตาม ในการสำรวจความสัมพันธ์ของปีที่มีการแนะนำเกี่ยวกับการปรับปรุงแก้ไขซอร์สโค้ด โดยการเก็บข้อมูลจากข้อเสนอแนะที่เกิดขึ้นในโครงการ OpenStack จำนวน 5 ปี ปรากฏผลดังตารางที่ 4.13 และในตารางที่ 4.14 จะเป็นการเก็บข้อมูลของโครงการ WikiMedia

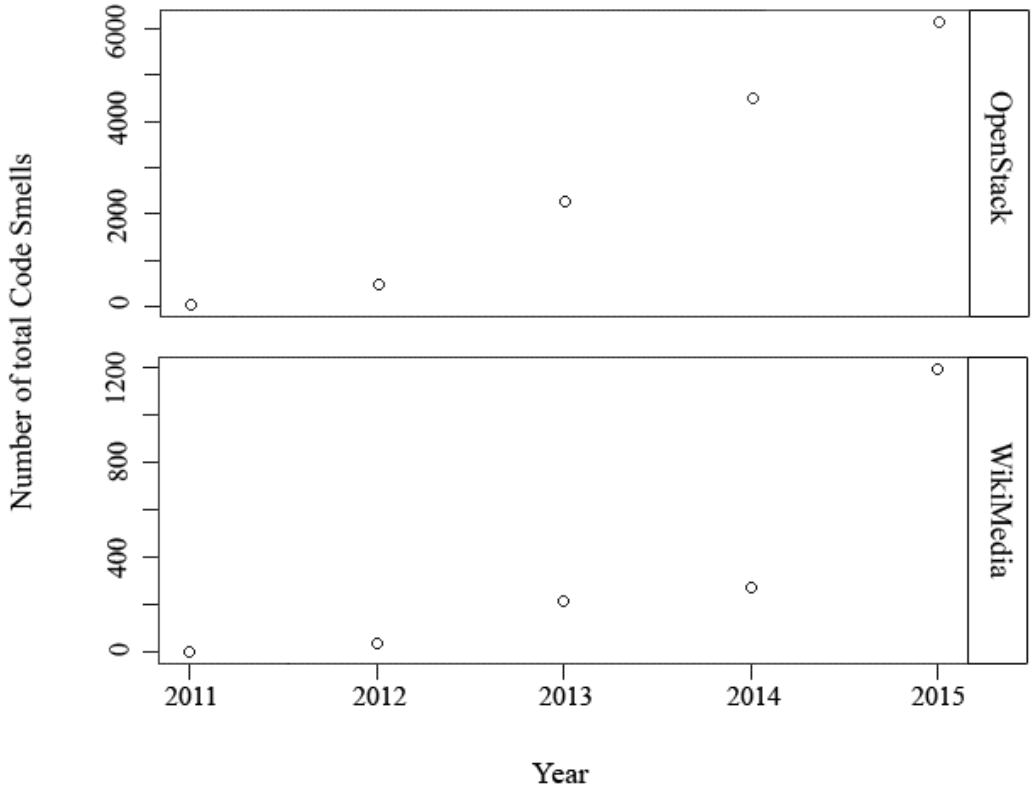
ตารางที่ 4.13 การเก็บข้อมูลจากข้อเสนอแนะที่เกิดขึ้นในโครงการ OpenStack จำนวน 5 ปี

ปี	ปีที่มีการให้คำแนะนำ (x)	จำนวนของข้อเสนอแนะที่ผู้ตรวจทานได้ให้คำแนะนำเกี่ยวกับร่องรอยโค้ดที่ไม่ดี (y)
1	2011	43
2	2012	490
3	2013	2276
4	2014	4496
5	2015	6156

ตารางที่ 4.14 การเก็บข้อมูลจากข้อเสนอแนะที่เกิดขึ้นในโครงการ WikiMedia จำนวน 5 ปี

ปี	ปีที่มีการให้คำแนะนำ (x)	จำนวนของข้อเสนอแนะที่ผู้ตรวจทานได้ให้คำแนะนำเกี่ยวกับร่องรอยโค้ดที่ไม่ดี (y)
1	2011	0
2	2012	34
3	2013	121
4	2014	273
5	2015	1194

จากข้อมูลของแต่ละโครงการจะมีอยู่ 5 ชุดข้อมูลด้วยกันเมื่อนำมาเขียนแผนภาพการกระจายแสดงความสัมพันธ์ โดยให้แกนตั้งแสดงจำนวนของข้อเสนอแนะที่ผู้ตรวจทานได้ให้คำแนะนำเกี่ยวกับร่องรอยโค้ดที่ไม่ดี (y) และแกนนอนแสดงปีที่ผู้ตรวจทานโค้ดได้ให้คำแนะนำ (x) จะได้รูปความสัมพันธ์ของทั้ง 2 ตัวแปร รูปที่ 4.10 แสดงความสัมพันธ์ระหว่างปีที่มีการแนะนำให้นักพัฒนาปรับปรุงแก้ไขซอร์สโค้ดกับจำนวนของข้อเสนอแนะที่ผู้ตรวจทานได้ให้คำแนะนำเกี่ยวกับร่องรอยโค้ดที่ไม่ดี พบว่าปีที่มีการกล่าวถึงร่องรอยโค้ดที่ไม่ดีกับจำนวนของข้อเสนอแนะที่ผู้ตรวจทานได้ให้คำแนะนำเกี่ยวกับร่องรอยโค้ดที่ไม่ดี มีความสัมพันธ์ในทิศทางเดียวกัน คือ เมื่อจำนวนปีที่เพิ่มขึ้น อัตราของการให้คำแนะนำเกี่ยวกับการเกิดร่องรอยโค้ดที่ไม่ดีก็จะสูงและถ้าย้อนกลับไปดูในอดีตที่ผ่านมา อัตราของการให้คำแนะนำเกี่ยวกับการเกิดร่องรอยโค้ดที่ไม่ดีก็จะต่ำลง



รูปที่ 4.10 กราฟเปรียบเทียบความสัมพันธ์ระหว่างปีที่มีการแนะนำให้นักพัฒนาปรับปรุงแก้ไขซอร์สโค้ดด้วยการทำรีแฟคทอริง

จากรูปที่ 4.10 เป็นการกระจายของข้อมูลของตัวแปรทั้ง 2 ตัวแปร สามารถมองเห็นแนวโน้มของความสัมพันธ์ของข้อมูลว่าจะเป็นไปในแนวทางใด ในการพยากรณ์เพื่อหาแนวโน้มที่เกิดขึ้นนั้นสามารถทำได้โดยการลากเส้นผ่านจุดต่างๆ โดยให้ระยะห่างระหว่างจุดกับเส้นที่ลากนั้นทุกจุดมีค่าน้อยที่สุด แล้วเส้นนั้นจะเป็นเส้นที่เหมาะสมที่สุด (line of the best fit) ซึ่งเรียกได้ว่า เป็นเส้นพยากรณ์ (prediction line) หรือเส้นถดถอย (regression line) ที่ดีที่สุด เส้นดังกล่าวนี้จะใช้ในการคาดคะเนค่าต่าง ๆ ของตัวแปรตาม เมื่อรู้ค่าของตัวแปรอิสระในสภาวะต่าง ๆ ดังนั้นเส้นถดถอยที่ใช้คาดคะเนตัวแปรหนึ่งเมื่อรู้ค่าตัวแปรตัวอื่นที่มีความสัมพันธ์ในเชิงเส้นตรง เรียกสมการความสัมพันธ์นั้นว่า สมการถดถอยเชิงเส้นตรง (linear regression equation) หรือที่เรียกกันว่า สมการถดถอย (simple regression equation) ดังนั้นในการพยากรณ์ตัวแปรตาม (y) เมื่อกำหนดตัวแปรอิสระ (x) สามารถหาได้จากสมการถดถอย

$$\hat{Y} = a+bX$$

โดยที่ $\hat{Y}$: ตัวแปรตาม (เนื่องจากค่าของ Y ขึ้นอยู่กับค่าของ X)
X : ตัวแปรอิสระ หรือตัวแปรต้น
a : ค่าคงที่ (Constant)เป็นค่าที่ตัดกันบน Y
b : ความชัน (Slope) ของเส้นกราฟ

ผู้วิจัยได้นำโปรแกรม R มาเป็นเครื่องมือที่ช่วยในขั้นตอนของการคำนวณค่าต่าง ๆ และนำมาใช้ในวิเคราะห์การถดถอย ในการแปลผลมีการกำหนดให้ระดับความเชื่อมั่นอยู่ที่ 95% โดยจะมีการกำหนดสมมติฐานทางสถิติดังนี้

H0 : จำนวนของข้อเสนอแนะที่เกี่ยวกับร่องรอยโค้ดที่ไม่ดีไม่มีความสัมพันธ์กับปีที่มีการแนะนำให้นักพัฒนาปรับปรุงแก้ไขซอร์สโค้ด

H1 : จำนวนของข้อเสนอแนะที่เกี่ยวกับร่องรอยโค้ดที่ไม่ดีมีความสัมพันธ์กับปีที่มีการแนะนำให้นักพัฒนาปรับปรุงแก้ไขซอร์สโค้ด

ในรูปที่ 4.11 จะแสดงผลลัพธ์ที่ทำการวิเคราะห์ข้อมูลของปีที่มีการแนะนำเกี่ยวกับการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริงกับจำนวนของข้อเสนอแนะที่เกี่ยวกับร่องรอยโค้ดที่ไม่ดีที่เกิดขึ้นในโครงการ OpenStack

```
Residuals:
       1      2      3      4      5
   597.2 -579.0 -416.2  180.6  217.4

   Coefficients:
                 Estimate Std. Error t value Pr(>|t|)
   (Intercept) -3264809.4   357270.3  -9.138  0.00277 **
   X               1623.2      177.5   9.146  0.00276 **

Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

   Residual standard error: 561.2 on 3 degrees of freedom
   Multiple R-squared:  0.9654,    Adjusted R-squared:  0.9538
   F-statistic: 83.64 on 1 and 3 DF,  p-value: 0.002763
```

รูปที่ 4.11 ผลลัพธ์ที่ได้จากการวิเคราะห์การถดถอยของโครงการ OpenStack

ผลที่ได้จากการวิเคราะห์โดยใช้โปรแกรม R จะได้ค่า p-value = 0.002763 ซึ่งค่า p-value ที่ได้น้อยกว่า 0.05 จึงทำการปฏิเสธ H0 สามารถสรุปได้ว่าจำนวนของข้อเสนอแนะที่เกี่ยวกับร่องรอยโค้ดที่ไม่ดีกับปีที่มีการแนะนำให้นักพัฒนาปรับปรุงแก้ไขซอร์สโค้ดมีความสัมพันธ์กันแบบมีนัยสำคัญทางสถิติ

จากสมการถดถอย ระหว่างปีที่มีการแนะนำให้นักพัฒนาปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริงกับจำนวนของข้อเสนอแนะที่เกี่ยวกับร่องรอยโค้ดที่ไม่ดีที่เกิดขึ้นในโครงการ OpenStack ได้เป็น

$$\hat{Y} = -3264809.4 + 1623.2x$$

ถ้าต้องการพยากรณ์อัตราการเกิดร่องรอยโค้ดที่ไม่ดีที่เกิดขึ้นในปี ค.ศ. 2017 สามารถทำได้โดยนำปีที่ต้องการจะทำนายแทนค่า x ในสมการ จะได้ค่า y ดังนี้

-3264809.4 + 1623.2(2017) = 9185 คือ ข้อเสนอแนะที่เกี่ยวข้องกับร่องรอยโค้ดที่ไม่ดีที่เกิดขึ้นในปี ค.ศ. 2017

ในรูปที่ 4.12 จะแสดงผลลัพธ์ที่ทำการวิเคราะห์ข้อมูลของปีที่มีการแนะนำเกี่ยวกับการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริงกับจำนวนของข้อเสนอแนะที่เกี่ยวกับร่องรอยโค้ดที่ไม่ดีที่เกิดขึ้นในโครงการ WikiMedia

```
Residuals:
          1       2       3       4       5
       182.8   -45.9  -130.6  -332.3   326.0

      Coefficients:
                    Estimate Std. Error t value Pr(>|t|)
      (Intercept) -528472.50  190714.60  -2.771   0.0695 .
      X               262.70      94.74   2.773   0.0694 .

Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

      Residual standard error: 299.6 on 3 degrees of freedom
      Multiple R-squared:  0.7193,    Adjusted R-squared:  0.6258
      F-statistic: 7.688 on 1 and 3 DF,  p-value: 0.06941
```

รูปที่ 4.12 ผลลัพธ์ที่ได้จากการวิเคราะห์การถดถอยของโครงการ WikiMedia

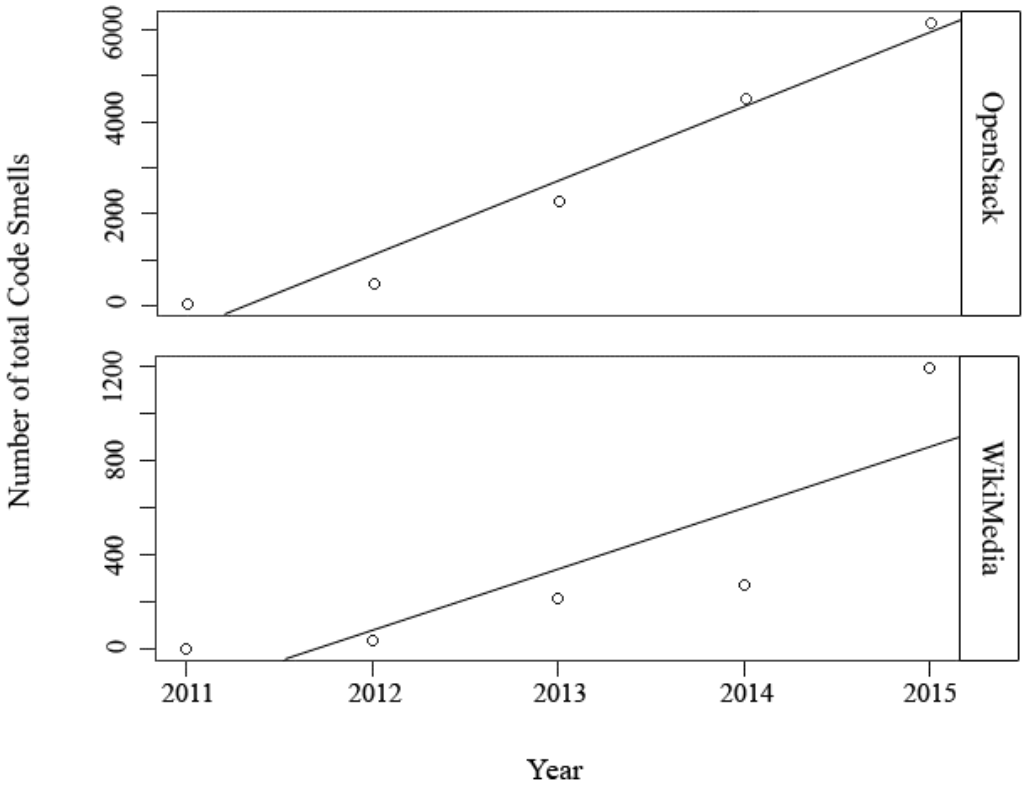
ผลที่ได้จากการวิเคราะห์โดยใช้โปรแกรม R จะได้ค่า p-value = 0.06941 ซึ่งค่า p-value ที่ได้มากกว่า 0.05 จึงทำการยอมรับ H0 สามารถสรุปได้ว่า จำนวนของข้อเสนอแนะที่เกี่ยวกับร่องรอยโค้ดที่ไม่ดีกับปีที่มีการแนะนำให้นักพัฒนาปรับปรุงแก้ไขซอร์สโค้ดไม่มีความสัมพันธ์กันแบบมีนัยสำคัญทางสถิติ

จากสมการถดถอย ระหว่างปีที่มีการแนะนำให้นักพัฒนาปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริงกับจำนวนของข้อเสนอแนะที่เกี่ยวกับร่องรอยโค้ดที่ไม่ดีที่เกิดขึ้นในโครงการ WikiMedia ได้เป็น

$$Y = -528472.50 + 262.70x$$

ถ้าต้องการพยากรณ์อัตราการเกิดร่องรอยโค้ดที่ไม่ดีที่เกิดขึ้นในปี ค.ศ. 2017 สามารถทำได้โดยนำปีที่ต้องการจะทำนายแทนค่า x ในสมการ จะได้ค่า y ดังนี้

-528472.50+ 262.70 (2017) = 1393.4 คือ ข้อเสนอแนะที่เกี่ยวข้องกับร่องรอยโค้ดที่ไม่ดีที่เกิดขึ้นในปี ค.ศ. 2017

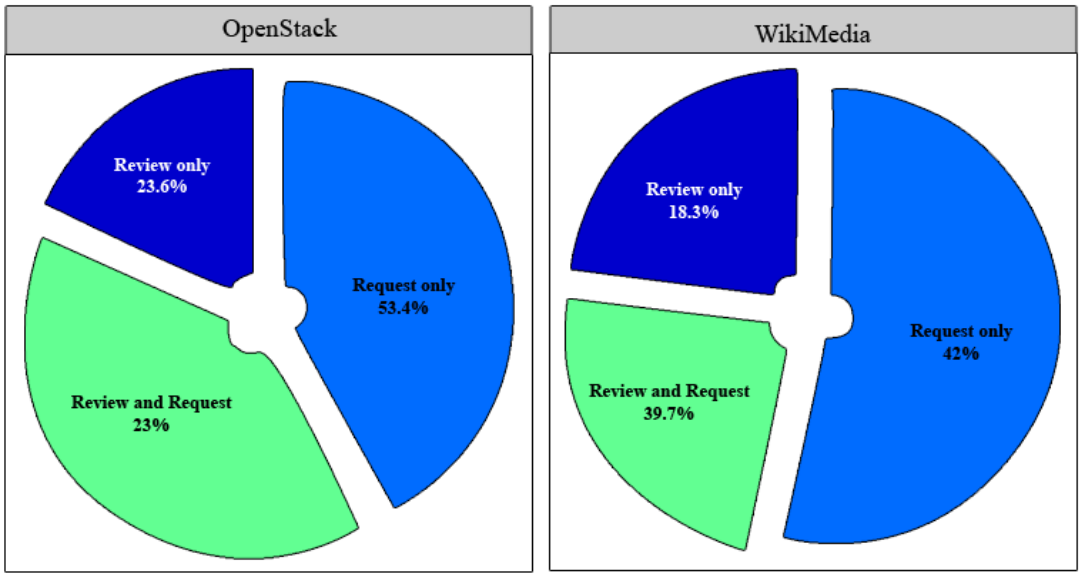


รูปที่ 4.13 กราฟการถดถอยที่ช่วยแสดงแนวโน้มของการเกิดร่องรอยโค้ดที่ไม่ดีในอนาคต

จากรูปที่ 4.13 สามารถทำการพยากรณ์ให้เห็นว่าในอนาคตอัตราการเกิดร่องรอยโค้ดที่ไม่ดีจะมีแนวโน้มเพิ่มขึ้น จากแนวโน้มนี้ผู้วิจัยเชื่อว่าเมื่อเวลาผ่านไปซอฟต์แวร์ก็ยิ่งมีความเสี่ยงต่อการเกิดข้อบกพร่อง เนื่องมาจากความซับซ้อนของระบบที่เพิ่มขึ้นเรื่อย ๆ จำนวนของซอร์สโค้ดที่มีมากขึ้น อาจจะส่งผลให้ข้อผิดพลาดเกิดมากขึ้นด้วย สังเกตได้จากเส้นกราฟการเกิดร่องรอยโค้ดที่ไม่ดีประเภท Duplicate code เป็นเส้นที่ผู้ตรวจทานโค้ดในโครงการ OpenStack ได้ให้คำแนะนำมากกว่าประเภทอื่น ๆ ซึ่งร่องรอยโค้ดที่ไม่ดีประเภท Duplicate code เป็นลักษณะของโค้ดที่มีการทำงานเหมือนกันมากกว่า 1 ที่ และในโครงการ WikiMedia ผู้ตรวจทานโค้ดได้ให้คำแนะนำร่องรอยโค้ดที่ไม่ดีในประเภท Comments meaning มากกว่าประเภทอื่น ๆ ซึ่งเป็นเรื่องของการเขียนคำอธิบายเมื่อครั้งที่ได้ทำการพบข้อบกพร่องของซอร์สโค้ดเมื่อนานมาแล้วไม่ได้ทำการปรับปรุงให้คำแนะนำนั้นเป็นปัจจุบัน เพื่อไม่ให้เกิดความสับสนเมื่อจำเป็นต้องกลับมาอ่านคำอธิบายนั้นอีกในอนาคต

4.4.2 ความสัมพันธ์ระหว่างผู้ตรวจทานโค้ดและข้อเสนอแนะ

ในกระบวนการตรวจทานโค้ดมี 2 บุคคลหลักที่มีความเกี่ยวข้องในกระบวนการตรวจทานโค้ด คือ บุคคลที่ทำหน้าที่ในการตรวจทานโค้ดและบุคคลที่ส่งคำร้องขอเพื่อให้ผู้ตรวจทานโค้ดตรวจทานโค้ดของตนเอง จากข้อมูลของข้อเสนอแนะของทั้ง 2 โครงการ ผู้วิจัยได้ทำการสรุปบทบาทต่าง ๆ ที่มีความเกี่ยวข้องในการทำให้เกิดข้อเสนอแนะ โดยจะแสดงอัตราร้อยละของแต่ละบทบาทที่เกิดขึ้นในแต่ละโครงการ แสดงดังรูปที่ 4.14



รูปที่ 4.14 จำนวนของแต่ละบทบาทที่เกิดขึ้นในแต่ละโครงการ

จากรูปที่ 4.14 แสดงให้เห็นบทบาทที่มีความเกี่ยวข้องกับข้อเสนอแนะของทั้ง 2 โครงการจะมี 3 บทบาทด้วยกันคือ

1) บุคคลที่ทำหน้าที่ในการตรวจทานโค้ดเพียงอย่างเดียว (Review only) เป็นบุคคลที่ทำการตรวจทานโค้ดของผู้อื่นเพียงอย่างเดียว โดยไม่มีส่วนร่วมในการส่งซอร์สโค้ดของตนเองให้ผู้ตรวจทานโค้ดท่านอื่น ๆ ทำการตรวจทานโค้ด

2) บุคคลที่ส่งคำร้องขอเพื่อให้ผู้ตรวจทานโค้ดตรวจทานโค้ดของตนเอง (Request only) เป็นบุคคลที่ส่งคำร้องขอเพื่อให้ผู้ตรวจทานโค้ดตรวจทานโค้ดเพียงอย่างเดียว โดยไม่มีบทบาทในการตรวจทานโค้ดของผู้อื่น

3) บุคคลที่ทำหน้าที่ทั้ง 2 บทบาท (Review and Request) เป็นบุคคลที่เป็นทั้งผู้ตรวจทานโค้ด และเป็นผู้ที่ส่งซอร์สโค้ดมาเพื่อต้องการให้ผู้ตรวจทานโค้ดท่านอื่น ๆ ทำการตรวจทานโค้ดของตนเอง

ผลที่ได้จากรูปที่ 4.14 จะเห็นได้ว่าบุคคลที่มีบทบาทในการตรวจทานโค้ดเพียงอย่างเดียวกับบุคคลที่ทำหน้าที่ในการตรวจทานโค้ดและเป็นบุคคลที่ส่งคำร้องขอในโครงการ OpenStack มีความแตกต่างกันเพียง 0.4% แต่ในโครงการ WikiMedia จำนวนของบุคคลที่มีการทำหน้าที่ทั้ง 2 บทบาท มีอัตราส่วนที่มากกว่าบุคคลที่ทำการตรวจทานโค้ดเพียงอย่างเดียวมีความแตกต่างกันถึง 21.4% อาจจะเป็นไปได้ว่าในโครงการ WikiMedia จะเป็นชุมชนของนักพัฒนาซอฟต์แวร์โอเพนซอร์สกลุ่มเดิมที่ทำหน้าที่ในการพัฒนาและเป็นบุคคลที่คอยทำหน้าที่ในการตรวจทานโค้ดไปด้วย และในทั้ง 2 โครงการมีจำนวนของบุคคลที่ส่งคำร้องขอเพื่อให้ผู้ตรวจทานโค้ดตรวจทานโค้ดของตนเองเพียงอย่างเดียวในอัตราส่วนที่มากกว่าบทบาทอื่น

นอกจากนี้ผู้วิจัยได้ทำการวิเคราะห์เพิ่มเติมในส่วนของผู้ตรวจทานโค้ดที่มี 2 บทบาท จากการรวบรวมข้อมูลของผู้ตรวจทานโค้ดที่มีการแนะนำเกี่ยวกับการปรับปรุงแก้ไขซอร์สในโครงการ OpenStack มีจำนวนผู้ตรวจทานโค้ด 1,233 คน และในโครงการ WikiMedia มีจำนวนผู้ตรวจทานโค้ด 149 คน จากข้อมูลที่ได้ทำการรวบรวมมานั้น มีผู้ตรวจทานโค้ดที่มี 2 บทบาท คือ เป็นผู้ตรวจทานโค้ด และเป็นผู้ที่ส่งซอร์สโค้ดเพื่อต้องการให้ผู้ตรวจทานโค้ดท่านอื่น ๆ ทำการตรวจทานโค้ดของตนเอง ในตารางที่ 4.15 จะแสดงให้เห็นรายละเอียดของผู้ตรวจทานโค้ดที่เป็นทั้งผู้ตรวจทานโค้ดและเป็นผู้ส่งคำร้องขอในโครงการ OpenStack และตารางที่ 4.16 จะเป็นรายละเอียดของโครงการ WikiMedia ผู้วิจัยได้ทำการระบุ ID ขึ้นมาเพื่อใช้ในการแทนชื่อของผู้ตรวจทานโค้ด ซึ่ง ID ที่ปรากฏในตารางจะมี 3 แบบ คือ

Ro_: เป็นชื่อของผู้ตรวจทานโค้ดในโครงการ OpenStack
Rw_: เป็นชื่อของผู้ตรวจทานโค้ดในโครงการ WikiMedia
2R_: เป็นชื่อของผู้ตรวจทานโค้ดที่ทำการตรวจทานโค้ดทั้ง 2 โครงการ

ตารางที่ 4.15 รายละเอียดของผู้ตรวจทานโค้ดที่มี 2 บทบาท ในโครงการ OpenStack

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Ro3	7	19
Ro4	2	1
Ro5	1	6
Ro7	3	3
Ro8	2	2
Ro11	1	13
Ro12	6	6
Ro15	1	1
Ro16	8	5
Ro17	6	1
Ro20	1	1
Ro24	13	3
Ro26	2	8
Ro27	4	28
Ro28	1	1
Ro29	3	1
Ro30	15	9
Ro31	1	4
Ro34	4	2
Ro35	7	19
Ro36	2	1
Ro37	1	2
Ro39	13	14
Ro40	2	2
Ro41	1	1
Ro43	3	1
Ro44	2	4
Ro45	5	6
Ro46	5	11
Ro47	3	2

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Ro49	23	14
Ro50	1	1
Ro51	8	8
Ro52	18	14
Ro53	2	3
Ro54	2	3
Ro55	5	4
Ro56	7	6
Ro57	1	3
Ro58	1	7
Ro60	6	1
Ro61	5	10
Ro63	7	2
Ro64	1	1
Ro66	7	1
Ro67	66	10
Ro69	51	22
Ro70	4	1
Ro71	1	6
Ro72	7	6
Ro74	5	5
Ro77	2	3
Ro78	1	1
Ro79	1	1
Ro80	31	5
Ro81	1	1
Ro84	3	5
Ro85	5	4
2R2	1	5
Ro88	9	4

ตารางที่ 4.15 รายละเอียดของผู้ตรวจทานโค้ดที่มี 2 บทบาท ในโครงการ OpenStack (ต่อ)

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Ro90	3	5
Ro91	4	2
Ro92	1	2
Ro93	22	1
Ro94	13	3
Ro95	1	3
Ro96	8	1
Ro97	1	1
Ro98	1	1
Ro99	3	1
Ro102	114	36
Ro103	1	5
Ro104	35	45
Ro106	1	3
Ro107	19	13
Ro108	4	5
Ro109	4	3
Ro111	18	5
2R3	4	1
Ro125	1	5
Ro126	15	43
Ro129	11	12
Ro130	4	1
Ro133	9	7
Ro135	11	16
Ro136	5	12
Ro137	1	7
Ro139	30	34
Ro142	16	23
Ro143	2	2

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Ro144	1	3
Ro145	1	2
Ro146	1	3
Ro147	22	47
Ro149	4	1
Ro150	150	22
Ro152	1	13
Ro153	2	1
Ro154	1	3
Ro156	64	5
Ro157	8	8
Ro159	2	2
Ro160	8	1
Ro161	2	3
Ro162	2	4
Ro163	7	1
Ro164	25	7
Ro165	1	2
Ro166	57	18
Ro168	9	5
Ro170	22	4
Ro172	11	8
Ro173	28	2
Ro174	8	7
Ro175	3	8
Ro177	1	3
Ro179	10	4
Ro180	1	1
Ro184	1	1
Ro187	112	39

ตารางที่ 4.15 รายละเอียดของผู้ตรวจทานโค้ดที่มี 2 บทบาท ในโครงการ OpenStack (ต่อ)

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Ro189	2	1
Ro191	30	14
Ro192	7	3
Ro193	59	11
Ro194	31	32
Ro200	58	22
Ro203	3	1
Ro204	3	1
Ro208	1	4
Ro209	6	4
Ro210	1	7
Ro212	2	2
Ro215	46	16
Ro216	2	1
Ro218	10	6
Ro219	13	8
Ro220	42	16
Ro223	54	26
Ro224	10	17
Ro226	2	10
Ro227	5	1
Ro229	3	3
Ro230	13	40
Ro231	1	3
Ro232	15	9
Ro234	6	2
Ro235	26	1
Ro236	12	2
Ro237	3	3
Ro238	4	11

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Ro239	14	1
Ro242	1	1
Ro243	22	17
Ro245	1	10
Ro246	6	4
Ro248	3	3
Ro251	140	59
Ro252	9	36
Ro256	5	1
Ro257	13	2
Ro260	2	10
Ro262	3	2
Ro263	1	1
Ro265	10	10
Ro266	1	5
Ro267	40	4
Ro268	7	9
Ro271	9	4
Ro272	6	5
Ro273	2	5
Ro276	2	4
Ro279	5	21
Ro280	41	30
Ro281	8	8
Ro285	1	5
Ro286	24	1
Ro287	4	5
Ro288	1	2
Ro290	1	6
Ro291	12	10

ตารางที่ 4.15 รายละเอียดของผู้ตรวจทานโค้ดที่มี 2 บทบาท ในโครงการ OpenStack (ต่อ)

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Ro297	8	2
Ro299	7	6
Ro300	1	5
Ro301	3	4
Ro303	4	3
Ro304	77	33
Ro305	6	3
Ro308	13	5
Ro309	104	29
Ro310	3	2
Ro311	2	3
Ro312	16	18
Ro316	2	1
Ro318	7	5
Ro319	13	16
Ro320	42	22
Ro321	2	2
Ro272	6	5
Ro322	57	17
Ro323	30	17
Ro324	8	3
Ro325	6	7
Ro328	68	6
Ro330	5	3
Ro331	4	2
2R10	1	1
Ro335	1	1
Ro338	21	15
Ro339	3	2
Ro363	57	11

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Ro364	12	1
Ro365	1	1
Ro367	1	5
Ro371	1	1
Ro372	2	8
Ro373	3	1
Ro374	5	1
Ro375	9	14
Ro376	27	30
Ro379	10	7
Ro341	25	22
Ro342	15	4
Ro343	13	4
Ro344	1	4
Ro345	54	15
Ro346	17	6
Ro347	1	2
Ro349	24	6
Ro351	9	2
Ro352	1	1
Ro345	54	15
Ro346	17	6
Ro347	1	2
Ro349	24	6
Ro351	9	2
Ro352	1	1
Ro353	4	1
Ro354	7	20
Ro355	5	21
Ro356	7	2

ตารางที่ 4.15 รายละเอียดของผู้ตรวจทานโค้ดที่มี 2 บทบาท ในโครงการ OpenStack (ต่อ)

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Ro357	18	20
Ro358	20	30
Ro359	1	1
Ro360	4	1
Ro361	7	5
Ro380	3	5
Ro383	2	1
Ro385	9	3
Ro386	25	12
Ro389	6	1
Ro390	1	1
Ro391	12	17
Ro392	2	6
Ro395	1	2
Ro396	76	75
Ro397	3	7
Ro400	2	2
Ro401	2	1
Ro402	3	13
Ro403	1	8
Ro404	2	5
Ro405	7	3
Ro408	1	5
Ro409	10	4
Ro410	6	2
Ro411	6	7
Ro412	4	9
Ro413	14	7
Ro415	5	5
Ro417	1	1

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Ro419	2	4
Ro420	2	2
Ro421	29	2
Ro422	2	6
Ro423	22	2
Ro424	4	1
Ro425	8	2
Ro428	2	3
Ro429	18	11
Ro431	3	4
Ro432	1	6
Ro434	5	7
Ro435	4	1
Ro437	4	6
Ro438	1	1
Ro440	1	2
Ro442	54	43
Ro443	3	2
Ro446	14	4
2R13	8	2
Ro448	30	4
Ro452	2	5
Ro453	44	43
Ro454	1	2
Ro455	9	14
Ro457	10	5
Ro458	2	2
Ro459	10	9
Ro460	10	10
Ro461	3	3

ตารางที่ 4.15 รายละเอียดของผู้ตรวจทานโค้ดที่มี 2 บทบาท ในโครงการ OpenStack (ต่อ)

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Ro462	1	8
Ro463	1	3
Ro464	4	13
Ro467	1	1
Ro468	9	20
Ro469	1	1
Ro470	3	3
Ro471	2	2
Ro473	1	7
Ro474	1	1
Ro475	5	3
Ro476	1	3
Ro477	11	12
Ro479	2	8
Ro481	4	8
Ro482	2	2
Ro483	3	4
Ro485	94	12
Ro486	5	3
Ro487	64	27
Ro488	3	13
Ro489	3	1
Ro490	22	7
Ro491	4	8
Ro493	6	3
Ro495	4	2
Ro496	5	8
Ro497	6	21
Ro498	6	14
Ro503	3	7

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Ro504	1	4
Ro505	12	8
Ro506	11	8
Ro508	7	15
Ro509	6	9
Ro1177	9	8
Ro1179	15	2
Ro1180	1	2
Ro1181	9	4
Ro1182	3	7
Ro1185	27	24
Ro1187	4	9
Ro1188	4	7
Ro1189	9	19
2R31	5	2
Ro1191	8	1
Ro1192	27	3
Ro1193	4	13
Ro1195	2	1
Ro1196	20	12
Ro1197	2	4
Ro1199	1	2
Ro1200	1	5
Ro1201	1	5
Ro1204	4	4
Ro1205	2	14
Ro1206	24	18
Ro1207	3	3
Ro1208	2	4
Ro1211	1	3

ตารางที่ 4.15 รายละเอียดของผู้ตรวจทานโค้ดที่มี 2 บทบาท ในโครงการ OpenStack (ต่อ)

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Ro1212	1	14
Ro1213	4	20
Ro1215	1	6
Ro1216	8	11
Ro1217	1	4
Ro1218	1	5
Ro1220	13	3
Ro1221	50	9
Ro1222	13	5

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Ro1224	1	1
Ro1225	4	3
Ro1226	1	14
Ro1227	2	4
Ro1228	16	2
Ro1230	2	9
Ro1231	5	10
Ro1232	3	4
Ro1233	1	6

ตารางที่ 4.16 รายละเอียดของผู้ตรวจทานโค้ดที่มี 2 บทบาท ในโครงการ WikiMedia

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Rw2	10	18
Rw3	4	3
Rw4	2	8
2R1	5	1
Rw7	3	8
Rw8	4	11
2R2	73	48
Rw10	1	3
Rw11	3	11
Rw12	1	2
Rw13	10	20
2R3	18	12
Rw15	5	13
Rw16	8	7
Rw17	2	1

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Rw18	4	5
Rw20	5	21
Rw21	11	4
Rw22	1	10
Rw25	4	1
2R6	7	2
Rw28	3	1
2R7	38	26
Rw30	12	5
Rw31	2	12
2R8	6	13
Rw33	2	1
Rw34	2	2
Rw37	3	1
2R10	9	16

ตารางที่ 4.16 รายละเอียดของผู้ตรวจทานโค้ดที่มี 2 บทบาท ในโครงการ WikiMedia (ต่อ)

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Rw40	2	6
2R11	4	5
Rw42	1	19
Rw43	6	2
Rw46	27	11
Rw48	5	2
Rw49	3	4
Rw51	1	1
2R12	10	8
2R13	8	2
Rw57	16	9
2R14	6	12
2R15	124	41
Rw60	22	18
Rw61	1	3
2R16	4	3
Rw64	4	6
2R17	10	2
Rw66	2	7
Rw69	8	5
Rw71	4	3
Rw73	4	11
2R18	101	6
Rw77	18	25
Rw79	1	6
Rw80	2	7
Rw81	1	2
Rw82	1	15
2R19	22	18
Rw84	11	5

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Rw85	1	1
2R20	7	7
2R21	4	3
Rw88	6	19
Rw89	7	2
Rw90	7	1
Rw91	12	25
2R22	5	2
Rw93	2	5
Rw100	57	10
Rw102	3	3
Rw103	1	11
Rw104	4	6
Rw105	1	42
2R24	13	11
2R25	9	5
Rw109	4	4
Rw110	1	3
Rw111	2	5
Rw112	1	3
Rw116	7	22
Rw119	4	26
Rw121	5	1
Rw123	1	1
Rw124	4	11
2R28	1	11
Rw127	4	8
Rw129	1	5
2R29	69	9
2R30	47	13

ตารางที่ 4.16 รายละเอียดของผู้ตรวจทานโค้ดที่มี 2 บทบาท ในโครงการ WikiMedia (ต่อ)

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Rw136	22	7
Rw138	6	2
Rw139	1	2
Rw140	1	3
Rw141	1	7
Rw142	1	3

ID	จำนวนการตรวจทานโค้ด	จำนวนการส่งคำร้องขอ
Rw143	11	1
Rw144	1	22
Rw145	6	12
2R31	133	50
Rw148	1	6
Rw149	1	2

จากตารางที่ 4.15 และตารางที่ 4.16 จะเห็นว่าทั้ง 2 โครงการจะมีบุคคลที่ทำหน้าที่เป็นทั้งผู้ตรวจทานโค้ด และเป็นบุคคลที่ส่งคำร้องขอในการขอให้ผู้ตรวจทานท่านอื่น ๆ ตรวจสอบโค้ดของตนเอง ผู้วิจัยได้นำสถิติสหสัมพันธ์ (Correlation) มาใช้ในการหาความสัมพันธ์ระหว่างบทบาทของการเป็นผู้ตรวจทานโค้ดและบทบาทของการเป็นผู้ส่งคำร้องขอว่ามีความสัมพันธ์กันหรือไม่ และมีผลต่อการเกิดข้อเสนอแนะอย่างไรบ้าง การบอกระดับหรือขนาดของความสัมพันธ์ จะใช้ตัวเลขของค่าสัมประสิทธิ์สหสัมพันธ์ หากค่าสัมประสิทธิ์สหสัมพันธ์มีค่าเข้าใกล้ -1 หรือ 1 แสดงถึงการมีความสัมพันธ์กันในระดับสูง แต่หากมีค่าเข้าใกล้ 0 แสดงถึงการมีความสัมพันธ์กันในระดับน้อยหรือไม่มีเลย สำหรับการพิจารณาค่าความสัมพันธ์จะใช้เกณฑ์ดังนี้

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

เครื่องหมาย +,- หนาตัวเลขสัมประสิทธิ์สหสัมพันธ์จะบอกถึงทิศทางของความสัมพันธ์ในกรณีที่ r มีเครื่องหมาย + หมายถึง การมีความสัมพันธ์กันไปในทิศทางเดียวกัน (ตัวแปรหนึ่งมีค่าสูงอีกตัวหนึ่งจะมีค่าสูงไปด้วย) และหาก r มีเครื่องหมาย - หมายถึงการมีความสัมพันธ์กันไปในทิศทางตรงกันข้าม (ตัวแปรหนึ่งมีค่าสูงอีกตัวหนึ่งจะมีค่าต่ำ)

ในรูปที่ 4.15 จะแสดงผลลัพธ์ที่ได้จากการวิเคราะห์ข้อมูลของความสัมพันธ์ระหว่างบทบาทของการเป็นผู้ตรวจทานโค้ดและบทบาทของการเป็นผู้ส่งคำร้องขอของโครงการทั้ง 2 โครงการ

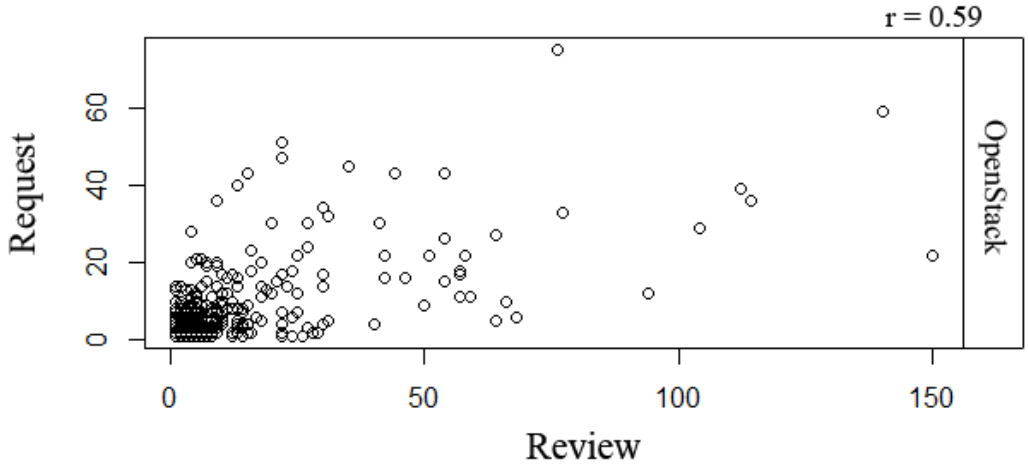
OpenStack	WikiMedia

`#summary of Review and Request`
`Review Request`
`Min. : 1.00 Min. : 1.000`
`1st Qu.: 2.00 1st Qu.: 2.000`
`Median : 4.00 Median : 4.000`
`Mean : 11.32 Mean : 7.849`
`3rd Qu.: 11.00 3rd Qu.: 9.000`
`Max. :150.00 Max. :75.000`
`NA's :177 NA's :177`
 	
`#correlation between Review and Request`
`Review Request`
`Review 1.0000000 0.5979061`
`Request 0.5979061 1.0000000`
	`# summary of Review and Request`
	`Review Request`
	`Min. : 1.00 Min. : 1.000`
	`1st Qu.: 2.00 1st Qu.: 3.000`
	`Median : 4.00 Median : 6.000`
	`Mean : 11.39 Mean : 9.235`
	`3rd Qu.: 9.00 3rd Qu.:11.750`
	`Max. :133.00 Max. :50.000`
	
	`# correlation between Review and Request`
	`Review Request`
	`Review 1.0000000 0.5714167`
	`Request 0.5714167 1.0000000`

รูปที่ 4.15 ผลลัพธ์ที่ได้จากการหาความสัมพันธ์ระหว่างบทบาทของการเป็นผู้ตรวจทานโค้ดและบทบาทของการเป็นผู้ส่งคำร้องขอของโครงการทั้ง 2 โครงการ

ผลที่ได้จากการวิเคราะห์โดยใช้โปรแกรม R จะได้ค่าสัมประสิทธิ์สหสัมพันธ์ของแต่ละโครงการดังนี้

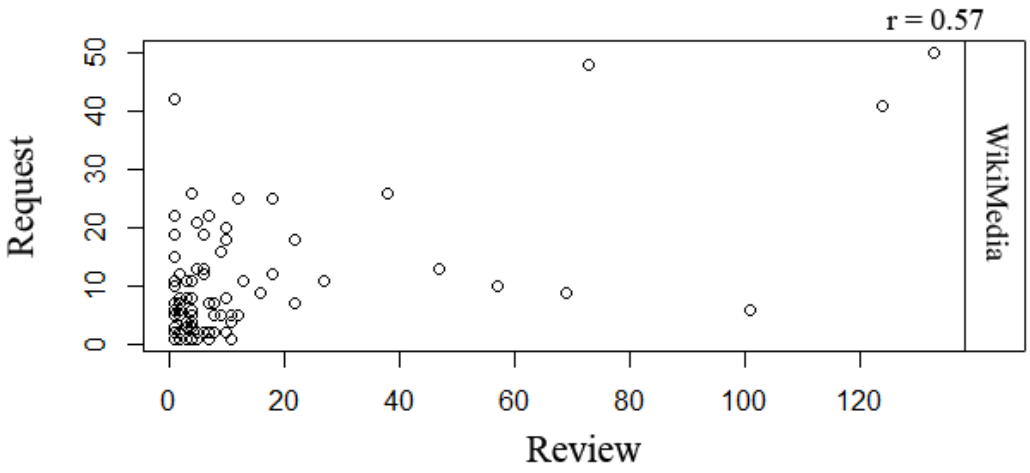
- ในโครงการ OpenStack จะได้ค่าสัมประสิทธิ์สหสัมพันธ์เท่ากับ 0.5979061 ซึ่งค่าสัมประสิทธิ์สหสัมพันธ์ที่ได้ไม่เท่ากับ 0 ดังนั้นจึงสรุปได้ว่าบทบาทของการเป็นผู้ตรวจทานโค้ดและบทบาทของการเป็นผู้ส่งคำร้องขอในโครงการ OpenStack มีความสัมพันธ์กันในระดับปานกลางและเป็นความสัมพันธ์ที่เป็นไปในทิศทางเดียวกัน หมายถึง หากผู้ตรวจทานโค้ดได้ให้คำแนะนำแก่นักพัฒนาในการปรับปรุงแก้ไขซอร์สโค้ดเป็นจำนวนมากก็จะทำการส่งคำร้องขอให้ผู้อื่นทำการตรวจทานโค้ดของตนเองมากเช่นกัน และถ้าผู้ตรวจทานโค้ดมีการให้คำแนะในการปรับปรุงแก้ไขซอร์สโค้ดน้อยจำนวนการของส่งคำร้องขอก็จะน้อยด้วย ในรูปที่ 4.16 จะแสดงกราฟความสัมพันธ์ระหว่างบทบาทของการเป็นผู้ตรวจทานโค้ดและบทบาทของการเป็นผู้ส่งคำร้องขอของโครงการ OpenStack



รูปที่ 4.16 กราฟความสัมพันธ์ระหว่างบทบาทของการเป็นผู้ตรวจทานโค้ดและบทบาทของการเป็นผู้ส่งคำร้องขอของโครงการ OpenStack

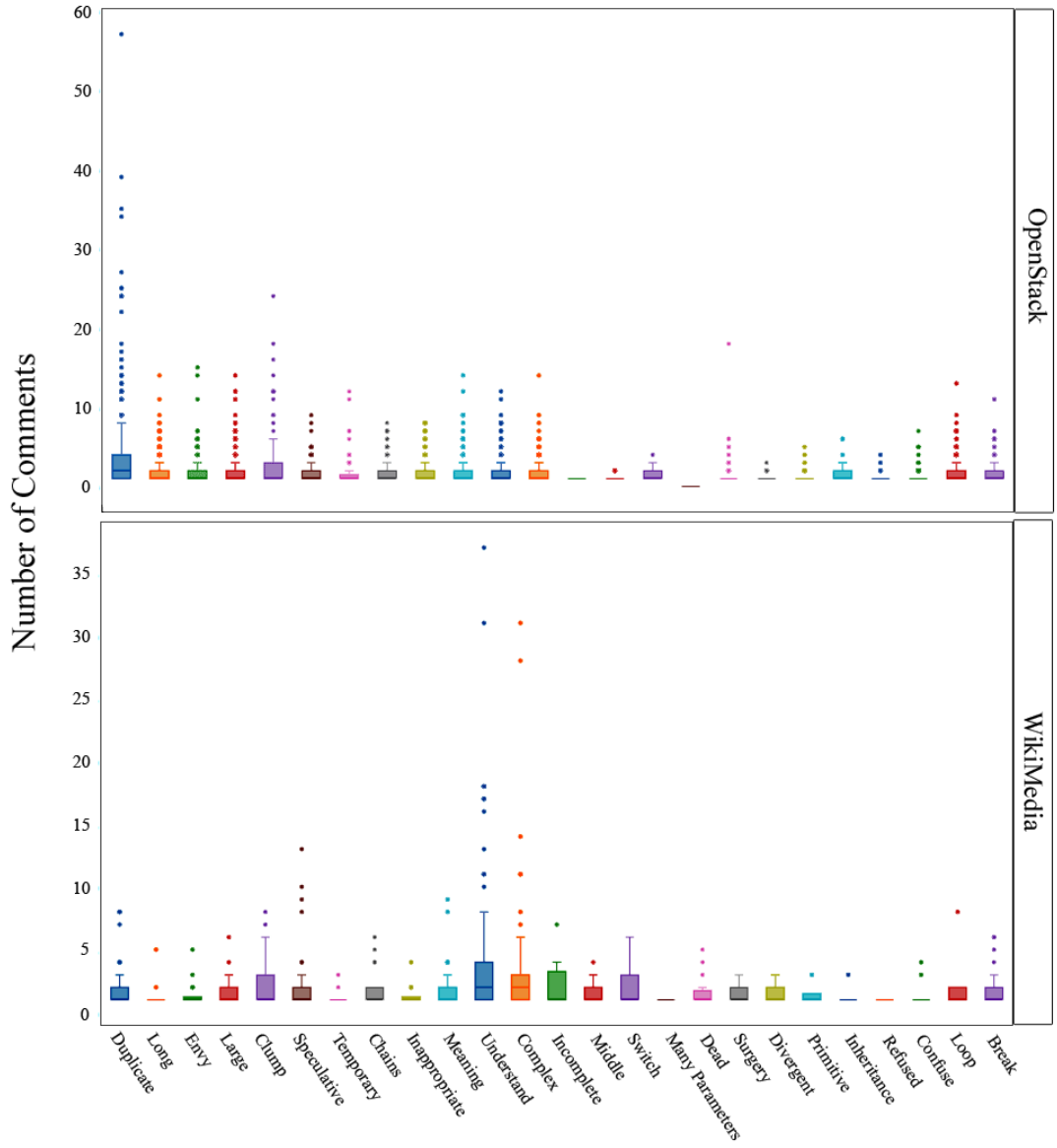
- ในโครงการ WikiMedia จะได้ค่าสัมประสิทธิ์สหสัมพันธ์เท่ากับ 0. 5714167 ซึ่ง

ซึ่งค่าสัมประสิทธิ์สหสัมพันธ์ที่ได้ไม่เท่ากับ 0 ดังนั้นจึงสามารถสรุปได้ว่าบทบาทของการเป็นผู้ตรวจทานโค้ดและบทบาทของการเป็นผู้ส่งคำร้องขอในโครงการ WikiMedia มีความสัมพันธ์กันในระดับปานกลางและเป็นความสัมพันธ์ที่เป็นไปในทิศทางเดียวกัน หมายถึง ถ้าผู้ตรวจทานโค้ดได้ให้คำแนะนำแก่นักพัฒนาในการปรับปรุงแก้ไขซอร์สโค้ดเป็นจำนวนเยอะก็จะทำการส่งคำร้องขอให้ผู้อื่นทำการตรวจทานโค้ดของตนเองเยอะเช่นกัน และถ้าผู้ตรวจทานโค้ดมีการให้คำแนะในการปรับปรุงแก้ไขซอร์สโค้ดน้อยจำนวนการของส่งคำร้องขอก็จะน้อยด้วย ในรูปที่ 4.17 จะแสดงกราฟความสัมพันธ์ระหว่างบทบาทของการเป็นผู้ตรวจทานโค้ดและบทบาทของการเป็นผู้ส่งคำร้องขอของโครงการ WikiMedia



รูปที่ 4.17 กราฟความสัมพันธ์ระหว่างบทบาทของการเป็นผู้ตรวจทานโค้ดและบทบาทของการเป็นผู้ส่งคำร้องขอของโครงการ WikiMedia

นอกจากนี้ ผู้วิจัยได้ทำการหาความสัมพันธ์ระหว่างจำนวนข้อเสนอแนะแต่ละประเภทกับจำนวนผู้ตรวจทานโค้ด รูปที่ 4.18 แสดงกราฟ Boxplot ซึ่งแสดงให้เห็นถึงความหนาแน่นของการให้คำแนะนำของผู้ตรวจทานโค้ดเกี่ยวกับร่องรอยโค้ดที่ไม่ดีในแต่ละประเภท แต่ละจุดในภาพจะแสดงถึงจำนวนข้อแนะนำของผู้ตรวจทานโค้ดแต่ละคน



รูปที่ 4.18 กราฟความหนาแน่นของการให้คำแนะนำของผู้ตรวจทานโค้ดเกี่ยวกับร่องรอยโค้ดที่ไม่ดีในแต่ละประเภท

จากกราฟจะเห็นได้ว่าความหนาแน่นของการให้ข้อเสนอแนะของผู้ตรวจทานโค้ดของทั้ง 2 โครงการจะมีลักษณะที่ใกล้เคียงกัน คือ ผู้ตรวจทานโค้ดส่วนใหญ่จะมีการให้คำแนะนำเกี่ยวกับร่องรอยโค้ดที่ไม่ดีในแต่ละประเภทจะมีจำนวน 1- 2 ข้อเสนอแนะ และมีผู้ตรวจทานโค้ดไม่กี่คนที่ให้ข้อเสนอแนะเกี่ยวกับร่องรอยโค้ดที่ไม่ดีประเภทเดิมซ้ำ ๆ ผู้วิจัยได้คำนวณหาค่าอัตราส่วนของจำนวนข้อเสนอแนะที่ผู้ตรวจทานโค้ดส่วนใหญ่ทำการให้ข้อเสนอแนะโดยการหาค่ามัธยฐาน ผลลัพธ์ของการคำนวณค่ามัธยฐานจะแสดงอยู่ในรูปที่ 4.18 โดยขนาดของกล่องจะสื่อถึงความหนาแน่นของจำนวนผู้ตรวจทานโค้ดที่มีการให้ข้อเสนอแนะเกี่ยวกับร่องรอยโค้ดที่ไม่ดีในแต่ละประเภท โดยจำนวนค่ามัธยฐานของข้อเสนอแนะที่ผู้ตรวจทานโค้ดของทั้ง 2 โครงการจะมีการให้ข้อเสนออยู่ที่ 1 ข้อเสนอแนะ ในโครงการ OpenStack จะมีผู้ตรวจทานโค้ดเพียงคนเดียวที่ให้ข้อเสนอแนะเกี่ยวกับ Duplicate code ถึง 57 คำแนะนำ ในทำนองเดียวกันกับโครงการ WikiMedia มีผู้ตรวจทานโค้ดเพียงคนเดียวที่ให้ข้อเสนอแนะของ Uncommunicative name ถึง 37 คำแนะนำ

จากการค้นหาความสัมพันธ์ระหว่างผู้ตรวจทานโค้ดและข้อเสนอแนะได้พบว่ามีผู้ตรวจทานโค้ดจำนวน 31 คน ที่ทำการตรวจทานโค้ดทั้ง ในโครงการ OpenStack และ โครงการ WikiMedia ตารางที่ 4.17 จะแสดงให้เห็นถึงจำนวนการให้ข้อเสนอแนะในแต่ละโครงการ

ตารางที่ 4.17 ผู้ตรวจทานโค้ดที่ทำหน้าที่ในการตรวจทานโค้ดและให้ข้อเสนอแนะเกี่ยวกับการปรับปรุงแก้ไขซอร์สโค้ดทั้ง 2 โครงการ

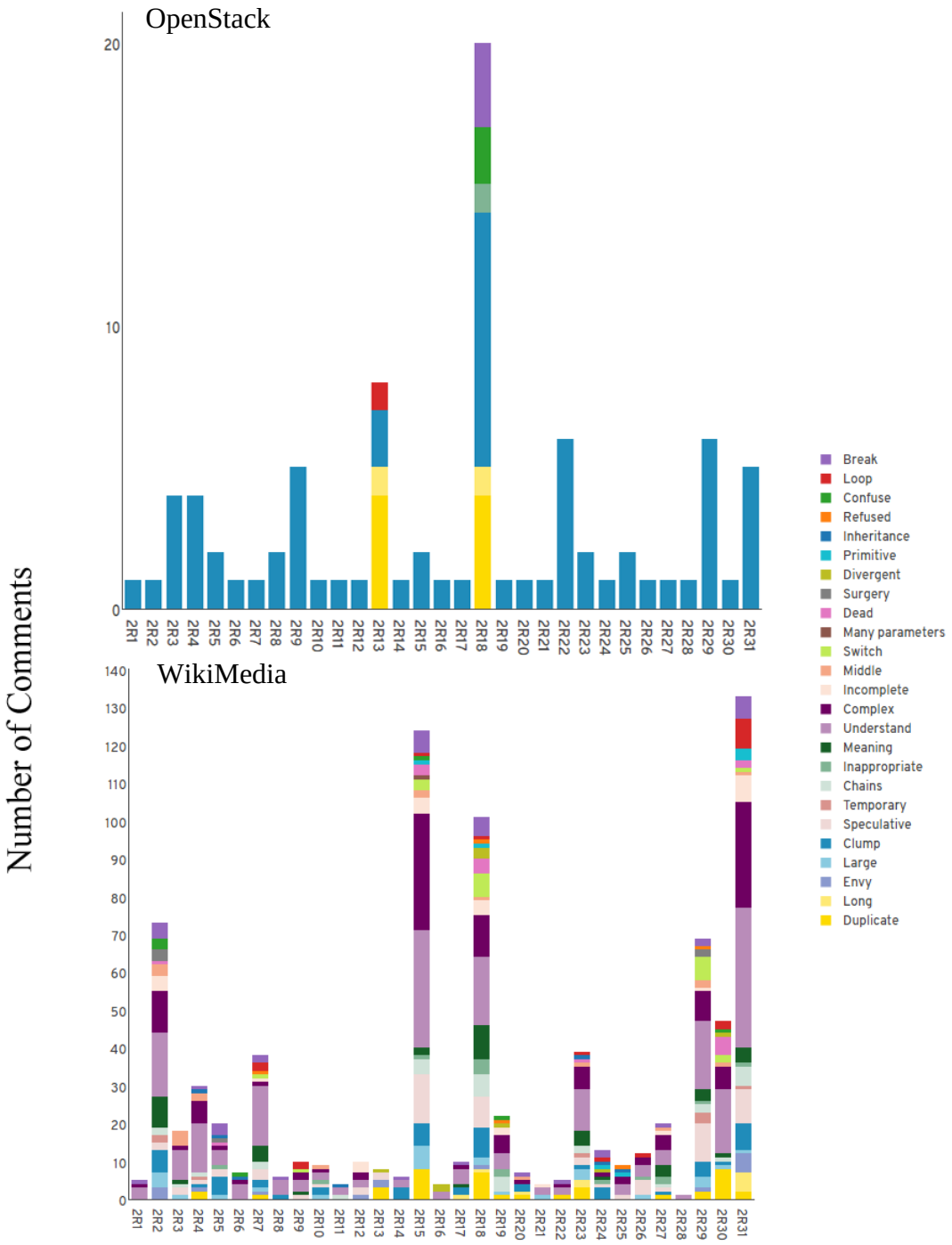
	Comment	
ID	OpenStack	WikiMedia
2R1	1	5
2R2	1	73
2R3	4	18
2R4	4	30
2R5	2	20
2R6	1	7
2R7	1	38
2R8	2	6
2R9	5	10
2R10	1	9

ตารางที่ 4.17 ผู้ตรวจทานโค้ดที่ทำหน้าที่ในการตรวจทานโค้ดและให้ข้อเสนอแนะเกี่ยวกับการปรับปรุงแก้ไขซอร์สโค้ดทั้ง 2 โครงการ (ต่อ)

	Comment	
ID	OpenStack	WikiMedia
2R11	1	4
2R12	1	10
2R13	8	8
2R14	1	6
2R15	2	124
2R16	1	4
2R17	1	10
2R18	20	101
2R19	1	22
2R20	1	7
2R21	1	4
2R22	6	5
2R23	2	39
2R24	1	13
2R25	2	9
2R26	1	12
2R27	1	20
2R28	1	1
2R29	6	69
2R30	1	47
2R31	5	133

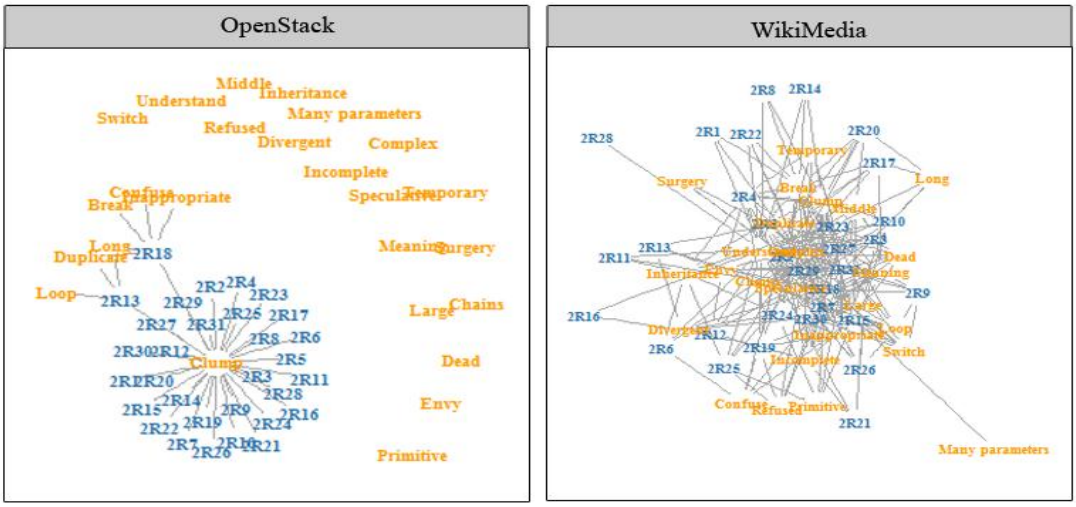
ผู้วิจัยได้ทำการศึกษาลักษณะข้อเสนอแนะของผู้ตรวจทานโค้ดที่มีบทบาทในการตรวจทานโค้ดทั้ง 2 โครงการ เพื่อดูว่าเมื่อเปลี่ยนโครงการแล้ว ผู้ตรวจทานโค้ดยังมุ่งเน้นที่จะให้คำแนะนำในการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริงผ่านร่องรอยโค้ดที่ไม่ดีประเภทเดิมอยู่หรือไม่ รูปที่ 4.19 จะแสดงให้เห็นถึงลักษณะของการให้ข้อเสนอแนะที่เกิดจากผู้ตรวจทานโค้ดคน

เดียวกัน โดยจะทำการวบรวมข้อเสนอแนะจากผู้ตรวจทานโค้ดทั้ง 31 คน ที่ทำการตรวจทานโค้ดทั้งในโครงการ OpenStack และโครงการ WikiMedia



รูปที่ 4.19 กราฟเปรียบเทียบลักษณะของการให้ข้อเสนอแนะที่เกิดจากผู้ตรวจทานโค้ดคนเดียวกัน

จากรูปที่ 4.19 ร่องรอยโค้ดที่ไม่ดีแต่ละประเภทจะถูกแสดงด้วยสีที่แตกต่างกัน และจะใช้ ID ในการระบุตัวตนของผู้ตรวจทานโค้ด ข้อเสนอแนะที่เกิดขึ้นในโครงการ OpenStack ผู้ตรวจทานโค้ดส่วนใหญ่ให้คำแนะนำร่องรอยโค้ดที่ไม่ดีประเภท Data Clump แต่จะมีผู้ตรวจทานโค้ด 2R13 และ 2R18 ที่มีการให้คำแนะนำเกี่ยวกับร่องรอยโค้ดที่ไม่ดีประเภท Duplicate Code, Loop, Confuse และ Break แต่หลัก ๆ ก็จะเป็นการให้คำแนะนำเกี่ยวกับร่องรอยโค้ดที่ไม่ดีประเภท Data Clump เพื่อเป็นการค้นหาว่าเมื่อเปลี่ยนโครงการแล้วผู้ตรวจทานโค้ดมุ่งเน้นที่จะให้คำแนะนำเกี่ยวกับร่องรอยโค้ดที่ไม่ดีประเภทเดิมอยู่หรือไม่ หรือเป็นการให้ความสำคัญกับร่องรอยโค้ดที่ไม่ดีแค่บางประเภท จำเป็นที่จะต้องตรวจสอบลักษณะของคำแนะนำของผู้ตรวจทานโค้ดคนเดิมที่ให้คำแนะนำเกี่ยวกับร่องรอยโค้ดที่ไม่ดี ลักษณะของข้อเสนอแนะที่เกิดขึ้นในโครงการ OpenStack จะเห็นว่าผู้ตรวจทานโค้ดคนเดิมจะให้ข้อเสนอแนะเกี่ยวกับร่องรอยโค้ดที่ไม่ดีอยู่เพียง 7 ประเภท ได้แก่ Break Code, Attribute Confuse, Data Clump, Duplicate Code, Function Loop, Long Parameter List and Method และ Inappropriate intimacy and Generality โดยที่ผู้ตรวจทานโค้ดทุกคนจะมีการให้ข้อเสนอแนะเกี่ยวกับร่องรอยโค้ดที่ไม่ดีประเภท Data Clump จากลักษณะของข้อเสนอแนะที่เกิดขึ้นในโครงการ WikiMedia จะเห็นว่าผู้ตรวจทานโค้ดคนเดิมจะมีการให้ข้อเสนอแนะเกี่ยวกับรองรอยโค้ดที่ไม่ดีหลายประเภท สิ่งนี้อาจจะบ่งบอกได้ว่า ผู้ตรวจทานโค้ดไม่ได้มีการยึดติดว่าตนเองจะต้องมุ่งเน้นที่จะต้องทำการให้คำแนะนำเฉพาะร่องรอยโค้ดที่ไม่ดีประเภทใดประเภทหนึ่งแต่ผู้ตรวจทานโค้ดจะทำหน้าที่ในการให้คำแนะนำตามลักษณะของข้อบกพร่องที่เกิดขึ้นตามความเป็นจริงที่เกิดขึ้นของโครงการ ผู้วิจัยได้ทำการสรุปความสัมพันธ์ของการให้คำแนะนำของผู้ตรวจทานโค้ดดังรูปที่ 4.20



รูปที่ 4.20 กราฟสรุปการให้ข้อเสนอแนะเกี่ยวกับร่องรอยของโค้ดที่ไม่ดี ในกรณีที่ผู้ตรวจทานโค้ดเป็นคนเดียวกัน

จากจำนวนของข้อเสนอแนะทั้งหมดในโครงการ OpenStack มีจำนวนข้อเสนอแนะที่ผู้ตรวจทานโค้ดได้ให้คำแนะนำให้นักพัฒนาทำการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริงมีเพียง 1.63% และในโครงการ WikiMedia มี 6.01% ซึ่งยังนับว่ามีปริมาณที่น้อยมาก แต่ผลจากการวิจัยพบว่า ผู้ตรวจทานโค้ดในโครงการโอเพนซอร์สได้ให้ความสำคัญเกี่ยวกับการปรับปรุงคุณภาพของซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง สังเกตได้จากผลการวิจัยที่แสดงให้เห็นถึงแนวโน้มสำหรับข้อแนะนำเกี่ยวกับร่องรอยโค้ดที่ไม่ดีมีอัตราที่เพิ่มสูงขึ้นเรื่อย ๆ อาจจะเป็นสัญญาณที่ดีว่าผู้ตรวจทานโค้ดเริ่มที่จะหันมาให้ความสนใจเกี่ยวกับการปรับปรุงคุณภาพของโค้ดด้วยวิธีการทำรีแฟคทอริง เพราะในปัจจุบันมีจำนวนโค้ดที่เพิ่มมากขึ้นทำให้ซอฟต์แวร์มีความซับซ้อน เสี่ยงต่อการเกิดข้อบกพร่องหากมีปัญหาเกิดขึ้นก็ย่อมจะส่งผลกระทบในหลาย ๆ ด้าน เช่น เรื่องของเวลา เรื่องของค่าใช้จ่าย ปัญหาเหล่านี้จะเกิดน้อยลงหากมีการจัดการโค้ดที่ดี จึงทำให้ผู้ตรวจทานโค้ดหันมาให้ความสนใจกับการนำวิธีการทำรีแฟคทอริงไปช่วยในการปรับปรุงคุณภาพซอฟต์แวร์ให้ดีขึ้น

บทที่ 5

บทสรุปผลการวิจัยและข้อเสนอแนะ

จากผลของงานวิจัยอิทธิพลของการทำรีแฟคทอริงต่อการตรวจทานโค้ดในโครงการซอฟต์แวร์โอเพนซอร์ส ซึ่งมีวัตถุประสงค์เพื่อทำการศึกษาหาอิทธิพลของการทำรีแฟคทอริงต่อการตรวจทานโค้ดในโครงการซอฟต์แวร์โอเพนซอร์ส พัฒนากระบวนการในการค้นหา และจัดกลุ่มประเภทของร่องรอยโค้ดที่ไม่ดีในโครงการ เพื่อทำการเก็บรวบรวมข้อมูลเชิงประจักษ์ของร่องรอยโค้ดที่ไม่ดีที่เกิดขึ้นในโครงการซอฟต์แวร์โอเพนซอร์ส โดยทำการศึกษาข้อเสนอแนะของผู้ตรวจทานโค้ดที่มีการแนะนำให้นักพัฒนาปรับปรุงหรือแก้ไขซอร์สโค้ดด้วยวิธีการรีแฟคทอริง ในงานวิจัยนี้ได้ทำการรวบรวมข้อเสนอแนะของผู้ตรวจทานโค้ดในโครงการ OpenStack และโครงการ WikiMedia ที่มีการแนะนำให้นักพัฒนาปรับปรุงหรือแก้ไขซอร์สโค้ดด้วยวิธีการรีแฟคทอริง โดยใช้วิธีการทางด้านการทำเหมืองข้อความ และวิเคราะห์ข้อมูลด้วยสถิติ ได้แก่ การวิเคราะห์การถดถอย และการวิเคราะห์สหสัมพันธ์ ผู้วิจัยได้สรุปผลการวิจัย อภิปรายผลการวิจัย สรุปภัยคุกคามต่อความถูกต้องในการทำงานวิจัย รวมทั้งข้อเสนอแนะต่าง ๆ ดังนี้

5.1 สรุปผลการวิจัย

งานวิจัยนี้ได้ทำการศึกษาอิทธิพลของการทำรีแฟคทอริงต่อการตรวจทานโค้ดในโครงการซอฟต์แวร์โอเพนซอร์ส จัดประเภทของร่องรอยโค้ดที่ไม่ดีที่เกิดขึ้น และทำการรวบรวมข้อมูลเชิงประจักษ์ที่ได้จากการศึกษาเพื่อช่วยให้นักวิจัยทางด้านวิศวกรรมซอฟต์แวร์เข้าใจการปรับปรุงคุณภาพซอฟต์แวร์โอเพนซอร์สให้มีคุณภาพ โดยเฉพาะการลดจำนวนร่องรอยโค้ดที่ไม่ดีประเภทต่าง ๆ ซึ่งอาจจะพบได้ในโครงการซอฟต์แวร์โอเพนซอร์ส โดยคำถามวิจัยที่ได้กำหนดไว้มีดังนี้

1) ผู้ตรวจทานโค้ดให้ความสำคัญกับการทำรีแฟคทอริงในโครงการซอฟต์แวร์โอเพนซอร์สหรือไม่
2) ร่องรอยของโค้ดที่ไม่ดีในโครงการซอฟต์แวร์โอเพนซอร์สแบ่งออกได้เป็นกี่ประเภท

จากการรวบรวมข้อเสนอแนะที่เกิดขึ้นในกระบวนการตรวจทานโค้ดตลอดช่วงปี ค.ศ. 2011-2015 ของทั้ง 2 โครงการ ในโครงการ OpenStack มีจำนวนข้อเสนอแนะทั้งหมด 822,845 ข้อเสนอแนะ และในโครงการ WikiMedia มีจำนวนข้อเสนอแนะทั้งหมด 26,947 ข้อเสนอแนะ พบว่าในโครงการ OpenStack มีจำนวนข้อเสนอแนะที่ผู้ตรวจทานโค้ดได้ให้คำแนะนำให้นักพัฒนาทำการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริงมีเพียง 1.63% และในโครงการ WikiMedia มี 6.01% ถือว่ายังเป็นจำนวนที่น้อย จากการพยากรณ์แนวโน้มของการเกิดร่องรอยโค้ดที่ไม่ดีในอนาคตพบว่าในอนาคตอัตราการเกิดร่องรอยโค้ดที่ไม่ดีจะมีแนวโน้มเพิ่มขึ้น จากการรวบรวมข้อมูลเชิงประจักษ์เกี่ยวกับประเภทของร่องรอยโค้ดที่ไม่ดีที่พบในทั้ง 2 โครงการ สามารถแบ่งประเภทของร่องรอยโค้ดที่ไม่ดีออกได้เป็น 25 ประเภท ได้แก่

- Attribute confuse
- Break code
- Comments meaning
- Conditional complexity
- Data clump
- Dead code
- Divergent changes
- Duplicate code
- Feature envy
- Function loop
- Inappropriate intimacy and Generality
- Incomplete library class
- Large class
- Long parameter list and Method
- Message chains
- Middle man
- Parallel Inheritance Hierarchies
- Primitive obsession
- Refused bequest
- Shotgun surgery

- Speculative generality
- Switch statement
- Temporary field
- Too many parameters
- Uncommunicative name

ผลที่ได้จากงานวิจัยนี้จะช่วยเพิ่มองค์ความรู้ใหม่ที่เกี่ยวกับการเพิ่มคุณภาพของซอฟต์แวร์ในโครงการซอฟต์แวร์โอเพนซอร์ส และเพิ่มหลักฐานเชิงประจักษ์ให้กับนักวิจัยด้านวิศวกรรมซอฟต์แวร์ ทำให้นักพัฒนาได้ตระหนักถึงความสำคัญของการนำรีแฟคทอริงไปช่วยในการพัฒนาซอฟต์แวร์ให้มีคุณภาพ

5.2 อภิปรายผลการวิจัย

ในส่วนนี้จะเป็นการอภิปรายผลที่ได้จากงานวิจัยและข้อเสนอแนะสำหรับนักวิจัยที่สนใจ และนักพัฒนาซอฟต์แวร์

จากผลการวิจัยพบว่าในโครงการ OpenStack และโครงการ WikiMedia มีผู้ตรวจทานโค้ดแค่บางกลุ่มที่ให้ความสำคัญเกี่ยวกับการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง สังเกตได้จากจำนวนของข้อเสนอแนะที่มักจะเกิดจากผู้ตรวจทานโค้ดคนเดิม ๆ ที่คอยให้คำแนะนำแก่นักพัฒนาให้ทำการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง ทั้งนี้อาจจะเป็นเพราะว่าผู้ตรวจทานโค้ดไม่มีความเข้าใจที่ดี เกี่ยวกับร่องรอยโค้ดที่ไม่ดีจึงทำให้มีบุคคลแค่บางกลุ่มที่คอยให้คำแนะนำในการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง ซึ่งมีความสอดคล้องกับงานวิจัยของ Yamashita และ Moonen ได้มีการศึกษาเกี่ยวกับความรู้ความเข้าใจของนักพัฒนาในการปรับปรุงซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง ผลที่ได้คือ มีนักพัฒนาจำนวนน้อยที่มีความรู้ความเข้าใจในการปรับปรุงร่องรอยโค้ดที่ไม่ดีในแต่ละประเภทด้วยวิธีการทำรีแฟคทอริง (Yamashita and Moonen, 2013)

จากผลการวิจัยพบว่ามีผู้ตรวจทานโค้ดบางกลุ่มที่ทำหน้าที่ในการให้คำแนะนำเกี่ยวกับการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริงในทั้ง 2 โครงการ ผู้วิจัยจึงทำการศึกษาลักษณะข้อเสนอแนะของผู้ตรวจทานโค้ดที่มีบทบาทในการตรวจทานโค้ดทั้งสองโครงการ ทำให้ทราบ

ว่า ผู้ตรวจทานโค้ดไม่ได้มุ่งเน้นที่จะต้องทำการให้คำแนะนำเฉพาะร่องรอยโค้ดที่ไม่ดีประเภทใดประเภทหนึ่ง แต่ผู้ตรวจทานโค้ดจะทำหน้าที่ในการให้คำแนะนำตามลักษณะของข้อบกพร่องที่เกิดขึ้นตามความเป็นจริงของโครงการ จากผลการวิจัยที่แสดงให้เห็นถึงแนวโน้มสำหรับข้อแนะนำเกี่ยวกับร่องรอยโค้ดที่ไม่ดีมีอัตราที่เพิ่มสูงขึ้นเรื่อย ๆ ซึ่งอาจจะเป็นสัญญาณที่ดีว่าผู้ตรวจทานโค้ดเริ่มที่จะหันมาให้ความสนใจเกี่ยวกับการปรับปรุงคุณภาพของโค้ดด้วยวิธีการทำรีแฟคทอริง

ร่องรอยโค้ดที่ไม่ดีที่ใช้ในการค้นหาข้อเสนอแนะในครั้งนี้มีทั้งหมด 25 ประเภท ซึ่งผู้วิจัยได้นำประเภทของร่องรอยโค้ดที่ไม่ดีทั้ง 22 ประเภท จากหนังสือของ Martin Fowler มาเป็นคำหลักเพื่อใช้ในการค้นหาข้อเสนอแนะของผู้ตรวจทานโค้ด และได้ทำการค้นหาร่องรอยโค้ดที่ไม่ดีประเภทใหม่เพิ่มเติมจากเดิมที่มีอยู่โดยการใช้อัลกอริทึม LDA ทำให้ผู้วิจัยค้นพบร่องรอยโค้ดที่ไม่ดีประเภทใหม่เพิ่มอีก 3 ประเภท ได้แก่ Break code, Attribute confuse และ Function loop

จากการศึกษาเกี่ยวกับร่องรอยโค้ดที่ไม่ดีประเภทต่าง ๆ ที่ผู้ตรวจทานโค้ดได้มีการให้ข้อเสนอแนะ ผลการวิจัยในครั้งนี้พบว่าในโครงการ OpenStack จะมีการให้ข้อเสนอแนะเกี่ยวกับร่องรอยโค้ดที่ไม่ดีประเภท Duplicate code มากกว่าประเภทอื่น ๆ ซึ่งร่องรอยโค้ดที่ไม่ดีประเภท Duplicate code เป็นลักษณะของโค้ดที่มีการทำงานเหมือนกันมากกว่า 1 ที่ และในโครงการ WikiMedia ผู้ตรวจทานโค้ดได้ให้คำแนะนำร่องรอยโค้ดที่ไม่ดีในประเภท Comments meaning มากกว่าประเภทอื่น ๆ

ผู้วิจัยมีความเห็นว่าในปัจจุบันโครงการโอเพนซอร์สมีจำนวนโค้ดที่เพิ่มมากขึ้นทำให้ซอฟต์แวร์มีความซับซ้อน เสี่ยงต่อการเกิดข้อบกพร่องหากมีปัญหาเกิดขึ้นก็ย่อมจะส่งผลกระทบในหลาย ๆ ด้าน เช่น เรื่องของเวลา เรื่องของค่าใช้จ่าย ปัญหาเหล่านี้จะเกิดน้อยลงหากมีการจัดการโค้ดที่ดี จึงทำให้ผู้ตรวจทานโค้ดหันมาให้ความสนใจกับการนำวิธีการทำรีแฟคทอริงไปช่วยในการปรับปรุงคุณภาพซอฟต์แวร์ให้ดีขึ้น จากแนวโน้มในเรื่องคุณภาพของซอฟต์แวร์ในโครงการโอเพนซอร์สที่จะเกิดขึ้นในอนาคตสามารถนำมาเป็นตัวช่วยในการตัดสินใจ ไม่ว่าจะเป็นในมุมมองของนักวิจัย นักพัฒนา และผู้ที่ใช้งานซอฟต์แวร์โอเพนซอร์สเตรียมพร้อมที่จะรับมือกับสิ่งที่จะเกิดขึ้นในอนาคต

เพื่อให้ผู้วิจัย หรือนักพัฒนาซอฟต์แวร์ที่สนใจที่จะดำเนินงานวิจัย และปรับปรุงคุณภาพของโครงการซอฟต์แวร์โอเพนซอร์ส ผู้วิจัยมีข้อเสนอแนะซึ่งได้จากการเรียนรู้ในงานวิจัยนี้ดังนี้

5.2.1 การกำหนดโครงการซอฟต์แวร์โอเพนซอร์สที่จะนำมาใช้ในการทำวิจัยจะต้องทำการศึกษาให้แน่ใจว่าโครงการที่เลือกมานั้นสามารถที่จะเข้าถึงแหล่งข้อมูลได้ เพราะแหล่งที่ใช้เก็บข้อมูลของซอฟต์แวร์โอเพนซอร์สจะมีทั้งแบบที่สามารถให้บุคคลที่สนใจเข้าถึงข้อมูลได้ และจะมีแบบ

ที่ไม่อนุญาตให้บุคคลทั่วไปสามารถเข้าถึงข้อมูลได้นอกจากบุคคลที่มีส่วนเกี่ยวข้องเท่านั้นที่จะสามารถเข้าถึงข้อมูลได้

5.2.2 การกำหนดคำหลักที่จะนำมาเป็นตัวช่วยในการค้นหาเพื่อให้สะดวกและรวดเร็วในการช่วยคัดกรองข้อมูลควรจะกำหนดให้สมบูรณ์ตั้งแต่ต้น เพราะหากมีการเพิ่มเติมเข้ามาทีหลังจะทำให้เกิดความยุ่งยากในการค้นหาข้อมูล ซึ่งจะทำให้เกิดความซ้ำซ้อนกันระหว่างข้อมูลชุดแรกที่ได้รวบรวมไว้กับข้อมูลที่เพิ่มเติมเข้ามาใหม่ และเพื่อให้ได้ข้อมูลที่สมบูรณ์มากยิ่งขึ้นควรจะมีการค้นหาคำที่มีความหมายสื่อถึงคำหลักมาเป็นตัวช่วยในการค้นหา

5.2.3 การทำความสะอาดข้อมูล ควรเลือกเครื่องมือและวิธีการในการทำความสะอาดข้อมูลให้มีความเหมาะสมตามลักษณะของชุดข้อมูลที่ได้ทำการรวบรวมไว้

5.2.4 สำหรับเครื่องมือที่ช่วยในการประมวลผล เช่น คอมพิวเตอร์และซอฟต์แวร์ที่ใช้จะต้องมีประสิทธิภาพมากพอที่จะทำการประมวลผลข้อมูลขนาดใหญ่ได้

5.2.5 ผลที่ได้จากงานวิจัยในครั้งนี้เป็นการศึกษาร่องรอยโค้ดที่ไม่ดีเพียง 25 ประเภท ซึ่งในการพัฒนาซอฟต์แวร์อาจจะเจอร่องรอยโค้ดที่ไม่ดีในลักษณะอื่น ๆ ที่แตกต่างไปจากที่กำหนดไว้ ผลที่ได้ในครั้งนี้เป็นเพียงแนวทางที่แสดงให้เห็นถึงแนวโน้มของการเกิดร่องรอยโค้ดที่ไม่ดีประเภทต่าง ๆ เพื่อให้นักพัฒนาได้ตระหนักและหลีกเลี่ยงการเกิดร่องรอยโค้ดที่ไม่ดีสำหรับในการพัฒนาซอฟต์แวร์ในอนาคต

5.2.6 สำหรับแนวทางการพัฒนางานวิจัยในอนาคตนั้นควรจะศึกษาต่อจากกรณีที่ผู้ตรวจทานโค้ดได้ให้ข้อเสนอแนะเกี่ยวกับการปรับปรุงโค้ดด้วยการทำรีแฟคทอริงแล้วนักพัฒนาได้ปฏิบัติตามคำแนะนำหรือไม่ รวมถึงหลังจากที่มีการทำรีแฟคทอริงแล้วจะสามารถทำให้คุณภาพของซอฟต์แวร์ดีขึ้นมากน้อยเพียงใด

5.3 ภัยคุกคามต่อความถูกต้อง (Threats to validity)

ภัยคุกคามต่อความถูกต้อง เป็นปัจจัยที่ส่งผลต่อความถูกต้อง ความเชื่อถือ และความเที่ยงตรงของการดำเนินงานวิจัย ในส่วนนี้ผู้วิจัยจะระบุเกี่ยวกับภัยคุกคามต่อความถูกต้องที่เกิดขึ้นในการศึกษาอิทธิพลของการทำรีแฟคทอริงต่อการตรวจทานโค้ดในโครงการซอฟต์แวร์โอเพนซอร์ส พร้อมทั้งอธิบายเกี่ยวกับข้อจำกัด ขอบเขต และการนำไปประยุกต์ใช้ โดยจะพิจารณาใน 3 ประเด็นหลัก ๆ คือ ความถูกต้องตามโครงสร้าง (Construct Validity) ความถูกต้องภายใน (Internal Validity) และความถูกต้องภายนอก (External Validity)

5.3.1 ความถูกต้องตามโครงสร้าง (Construct Validity)

ความถูกต้องตามโครงสร้าง เป็นการพิจารณาคุณภาพของเครื่องมือวิจัยเพื่อดูว่าเครื่องมือวิจัยนั้นสามารถใช้วัดหรืออธิบายสิ่งที่ต้องการวัดได้สอดคล้องตามทฤษฎีหรือไม่ ซึ่งครอบคลุมคำถามหรือปัญหาของงานวิจัย วิธีการได้มาของข้อมูล และเครื่องมือวิจัยที่ใช้รวบรวมข้อมูล สำหรับงานวิจัยในครั้งผู้วิจัยได้ระบุถึงภัยคุกคามที่อาจจะส่งผลต่อความถูกต้องตามโครงสร้างไว้ดังนี้

1) ในงานวิจัยนี้ได้ทำการรวบรวมข้อเสนอแนะที่เกิดขึ้นในกระบวนการตรวจทานโค้ดของโครงการซอฟต์แวร์โอเพนซอร์ส ซึ่งข้อเสนอแนะเหล่านี้จะถูกจัดเก็บไว้ในระบบเกอริต ผู้วิจัยได้มีการศึกษาโครงสร้างการจัดเก็บข้อมูลของระบบเกอริต เพื่อนำมาใช้ในการออกแบบและพัฒนาซอฟต์แวร์ที่ช่วยในการสืบค้นและจัดเก็บข้อมูล ซึ่งข้อมูลที่ได้รวบรวมมานั้นอาจจะไม่ครบสมบูรณ์ตามข้อมูลที่แสดงอยู่ในระบบเกอริต เช่น กราฟ รูปภาพ และสัญลักษณ์ต่าง ๆ อย่างไรก็ตามผู้วิจัยได้ทำการรวบรวมข้อมูลที่สำคัญและเป็นข้อมูลที่มีความจำเป็นในการนำมาวิเคราะห์ผลเกี่ยวกับอิทธิพลของการทำรีแฟคทอริงต่อกระบวนการตรวจทานโค้ด เป็นข้อมูลที่เพียงพอในการสรุปผลตามวัตถุประสงค์ที่วางไว้

2) งานวิจัยในครั้งนี้มีการนำประเภทของร่องรอยโค้ดที่ไม่ดี 22 ประเภทมาจากหนังสือของ Martin Fowler และผู้วิจัยได้ทำการค้นพบร่องรอยโค้ดที่ไม่ดีประเภทใหม่เพิ่มอีก 3 ประเภท ทำให้ผลที่ได้จากวิจัยในครั้งนี้จะเป็นผลที่เกิดมาจากการรวบรวมข้อเสนอแนะที่มีความเกี่ยวข้องกับร่องรอยโค้ดที่ไม่ดีทั้งหมด 25 ประเภท มาเป็นคำหลักเพื่อใช้ในการค้นหาข้อเสนอแนะของผู้ตรวจทานโค้ด เหตุผลที่ผู้วิจัยนำประเภทของร่องรอยโค้ดที่ไม่ดีมาจากหนังสือของ Martin Fowler เนื่องจากร่องรอยโค้ดที่ไม่ดีทั้ง 22 ประเภท จากหนังสือเล่มนี้ได้รับการศึกษาและได้รับการยอมรับโดยทั่วไปในงานวิจัยทางด้านวิศวกรรมซอฟต์แวร์ แต่อย่างไรก็ตามจากการทบทวนวรรณกรรมที่เกี่ยวข้องจากหนังสือหรือตำราเล่มอื่น ๆ พบว่าร่องรอยโค้ดที่ไม่ดีประเภทเดียวกันอาจจะมีการใช้ชื่อเรียกที่ไม่เหมือนกัน แต่เมื่อได้อ่านคำอธิบายเกี่ยวกับลักษณะของร่องรอยโค้ดที่ไม่ดี รวมไปถึงวิธีการที่ใช้สำหรับการทำรีแฟคทอริงจะไม่มีความแตกต่างกัน

5.3.2 ความถูกต้องภายใน (Internal Validity)

ความถูกต้องภายใน เป็นการพิจารณาความถูกต้องของผลการวิจัยที่เกิดขึ้น ซึ่งสามารถสรุปได้ว่าผลของการวิจัยเกิดขึ้นมาจากอิทธิพลของสิ่งที่ดำเนินการภายในงานวิจัยเพียงอย่างเดียวเท่านั้น ไม่ได้เกิดมาจากปัจจัยอื่น ๆ ที่นักวิจัยไม่ได้ทำการศึกษา ความถูกต้องภายในของการวิจัยนับเป็นสิ่งที่ถือว่าสำคัญที่สุดของการดำเนินการวิจัย สำหรับงานวิจัยในครั้งผู้วิจัยได้ระบุถึงภัยคุกคามที่อาจจะส่งผลต่อความตรงภายในไว้ดังนี้

1) ในงานวิจัยนี้เป็นการนำข้อเสนอแนะที่เกิดขึ้นในกระบวนการตรวจทานโค้ดของโครงการ OpenStack และ WikiMedia มาทำการค้นหาข้อเสนอแนะที่ผู้ตรวจทานโค้ดได้มีการแนะนำให้นักพัฒนาทำการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง ในการค้นหาข้อเสนอแนะผู้วิจัยได้ทำการกำหนดคำหลักและคำที่มีความหมายสื่อถึงคำหลักที่เกี่ยวข้องกับร่องรอยโค้ดที่ไม่ดีประเภทต่าง ๆ มาเป็นตัวช่วยในการค้นหา ในกรณีที่ผู้ตรวจทานโค้ดมีการใช้คำไม่ตรงตามที่ผู้วิจัยได้กำหนดไว้ ข้อเสนอแนะนั้นก็จะไม่นำมาวิเคราะห์ ผู้วิจัยจะทำการวิเคราะห์ข้อมูลของข้อเสนอแนะที่มีการใช้คำหลัก หรือคำที่มีความหมายสื่อถึงคำหลักตามที่กำหนดไว้เท่านั้น ในขั้นตอนของการทำความสะอาดข้อมูลอาจจะทำให้ข้อมูลที่เป็นประโยชน์บางส่วนสูญหายไปบ้าง อย่างไรก็ตามผู้วิจัยได้ทำการอ่านข้อเสนอแนะทั้งหมดที่ผ่านกระบวนการทำความสะอาดข้อมูลเพื่อทำการตรวจสอบข้อเสนอแนะอีกครั้งก่อนที่จะทำการวิเคราะห์และสรุปผล ซึ่งการอ่านข้อเสนอแนะเพียงคนเดียวอาจจะก่อให้เกิดความเอนเอียง (Bias) ได้ แต่ในการตรวจสอบข้อเสนอแนะผู้วิจัยจะมีการกำหนดเกณฑ์ที่ใช้สำหรับการพิจารณาข้อเสนอแนะ คือ จะตรวจสอบดูว่าในข้อเสนอแนะนั้นมีคำหลักที่สื่อถึงประเภทของร่องรอยโค้ดที่ไม่ดีปรากฏอยู่ในข้อความตามที่กำหนดไว้หรือไม่ หากข้อเสนอแนะนั้นมีประเภทของร่องรอยโค้ดที่ไม่ดีอยู่ก็จะนำข้อเสนอแนะไปทำการวิเคราะห์ แต่ในกรณีที่ไม่มีคำหลักปรากฏในข้อความก็จะไม่นำข้อเสนอแนะมาทำการวิเคราะห์

จากการวิเคราะห์ผลการวิจัยในครั้งนี้เป็นการวิเคราะห์โดยให้ความสำคัญกับลักษณะข้อเสนอแนะของผู้ตรวจทานโค้ดที่มีการแนะนำให้นักพัฒนาทำการปรับปรุงแก้ไขซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริง โดยสื่อผ่านร่องรอยโค้ดที่ไม่ดีประเภทต่าง ๆ ที่เกิดขึ้นในช่วงระยะเวลา 5 ปี จะเห็นว่ามีปัจจัยเรื่องเวลาเข้ามาเกี่ยวข้องอาจจะทำให้ประสบการณ์ของผู้ตรวจทานโค้ดเปลี่ยนไป ผลการวิจัยพบว่าจำนวนข้อเสนอแนะของร่องรอยโค้ดที่ไม่ดีในแต่ละโครงการโอเพนซอร์สมีจำนวนเพิ่มขึ้น และมีแนวโน้มที่จะเพิ่มขึ้นอีกในอนาคตอาจจะเป็นไปได้ว่าข้อเสนอแนะที่เกิดขึ้นในอดีตสู่อนาคตเกิดจากการที่ผู้ตรวจทานโค้ดมีความเข้าใจเกี่ยวกับร่องรอยโค้ดที่ไม่ดีมากขึ้นหรือสนใจเกี่ยวกับการปรับปรุงซอร์สโค้ดด้วยวิธีการทำรีแฟคทอริงมากขึ้น

5.3.3 ความถูกต้องภายนอก (External Validity)

ความถูกต้องภายนอก เป็นการพิจารณาถึงผลวิจัยที่ค้นพบจากการทำวิจัยว่าสามารถนำไปใช้กับกลุ่มตัวอย่างกลุ่มอื่น ๆ หรือสถานการณ์อื่น ๆ ได้มากน้อยเพียงใด หากสามารถนำไปใช้ได้กับประชากรกลุ่มอื่น ๆ นอกเหนือจากกลุ่มตัวอย่างที่เก็บข้อมูลได้ดีก็ย่อมแสดงว่าการวิจัยครั้งนั้นมีความถูกต้องภายนอกสูง สำหรับงานวิจัยในครั้งผู้วิจัยได้ระบุถึงภัยคุกคามที่อาจจะส่งผลต่อความถูกต้องภายนอกไว้ดังนี้

1) ในงานวิจัยนี้ทำการศึกษาข้อเสนอแนะของผู้ตรวจทานโค้ดในช่วงปี ค.ศ. 2011- 2015 อยู่ในช่วงระยะเวลา 5 ปี ไม่ได้นำเสนอข้อมูลของปี ค.ศ. 2016 อาจจะทำให้แนวโน้ม

การให้ความสนใจของผู้ตรวจทานโค้ดในแต่ละประเภทอาจจะมีความคลาดเคลื่อนเล็กน้อย และมุ่งเน้นทำการศึกษาเฉพาะซอฟต์แวร์โอเพนซอร์สโครงการ OpenStack และ โครงการ WikiMedia ดังนั้นอาจจะยังมีความจำเป็นในการศึกษาโครงการโอเพนซอร์สอื่น ๆ ในช่วงระยะเวลาที่นานขึ้น เพื่อดูแนวโน้ม และทิศทางของการให้คำแนะนำของผู้ตรวจทานโค้ดที่แม่นยำขึ้น ซึ่งผลที่ได้จากการวิจัยนี้สามารถนำไปใช้ดูแนวโน้มและทิศทางในการปรับปรุงโค้ดของซอฟต์แวร์เชิงอุตสาหกรรมได้ เพราะผลลัพธ์ที่ได้อาจจะไม่มีความแตกต่างกันภายใต้กระบวนการตรวจทานโค้ดที่มีวิธีการคล้ายคลึงกัน

ผลที่ได้จากงานวิจัยในครั้งนี้เป็นเพียงการสรุปผลขั้นพื้นฐาน อย่างไรก็ตามข้อมูลที่มีอยู่ก็เพียงพอสำหรับนำมาเป็นแนวทางในการพัฒนาหรือการปรับปรุงคุณภาพซอฟต์แวร์โอเพนซอร์สให้ดียิ่งขึ้น ผู้วิจัยเชื่อว่างานวิจัยนี้น่าจะช่วยให้นักพัฒนาซอฟต์แวร์ในโครงการโอเพนซอร์ส หรือนักวิจัยที่ให้ความสนใจเกี่ยวกับการพัฒนาซอฟต์แวร์ได้ตระหนักและเห็นถึงความสำคัญของการนำรีแฟคทอริงไปช่วยในการพัฒนาซอฟต์แวร์ให้มีคุณภาพ เพราะเมื่อเวลาผ่านไปซอฟต์แวร์ก็ยิ่งมีความเสี่ยงต่อการเกิดข้อบกพร่อง เนื่องมาจากความซับซ้อนของระบบที่เพิ่มขึ้นเรื่อย ๆ หากไม่ให้ความสนใจในเรื่องคุณภาพของโค้ดหรือให้ความสนใจเฉพาะผลลัพธ์ ในอนาคตมีแนวโน้มว่า ซอร์สโค้ดนั้นอาจจะก่อให้เกิดปัญหาหรือเกิดข้อบกพร่องในซอฟต์แวร์ ก่อนที่จะกลายเป็นปัญหาใหญ่จนถึงขั้นที่ทำให้ซอฟต์แวร์ทำงานไม่ได้ หากมีปัญหาเกิดขึ้นก็ย่อมจะส่งผลกระทบในหลาย ๆ ด้าน เช่น เรื่องของเวลา เรื่องของค่าใช้จ่าย ซึ่งปัญหาเหล่านี้จะเกิดน้อยลงหากมีการจัดการโค้ดที่ดี

เอกสารอ้างอิง

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

Abebe, S., Haiduc, S., Tonella, P., and Marcus, A. (2011). “The Effect of Lexicon Bad on Concept Location in Source Code.”, *Proceeding of The 11th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM),* Williamsburg, VI, USA: 25-26 September, 2011.

Abu Bakar, N., and Arsat, N. (2014). “Investigating the Factors that Influence the Quality of Open Source Systems.”, *Proceeding of The 5th International Conference on Information and Communication Technology for The Muslim World (ICT4M),* Kuching, Malaysia: 17-18 November, 2014.

Andrzejewski, D., Zhu, X., Craven, M., and Recht, B. (2011). “A framework for Incorporating general domain Kknowledge into Latent Dirichlet Allocation using first-order logic.” *United States. Dept. of Energy, Washington, D.C.,*22(1), 1171-1180.

Badri, M., Drouin, N., and Touré, F. (2012). “On Understanding Software Quality Evolution from a Defect Perspective: A Case Study on an Open Source Software System.”, *Proceeding of International Conference on Computer Systems and Industrial Informatics (ICCSII),* Sharjah, United Arab Emirates: 18-20 December, 2012.

Baum, T., Liskin, O., Niklas, K., and Schneider, K. (2016). “Factors Influencing Code Review Processes in Industry.”, *Proceeding of The 24th ACM SIGSOFT International Conference on Foundations of Software Engineering*, Seattle, WA, USA: 13-18 November, 2016.

Baysal, O., Kononenko, O., Holmes, R., and Godfrey, M. (2013). “The Influence of Non- Technical Factors on Code Review.”, *Proceeding of The 20th Working Conference on Reverse Engineering (WCRE)*, Koblenz, Germany: 14-17 October, 2013.

Baysal, O., Kononenko, O., Holmes, R., and Godfrey, M. (2015). “Investigating technical and non-technical factors influencing modern code review.” *Empirical Software Engineering*, 21(3), 932-959.

Beller, M., Bacchelli, A., Zaidman, A., and Juergens, E. (2014). "Modern Code Reviews In Open-Source Projects: Which Problems do they Fix?", *Proceeding of The 11th Working Conference on mining software repositories*, Hyderabad, India: 31 May - 01 June, 2014.

Blei, David M., Andrew Y. Ng, and Michael I. Jordan. (2003). "Latent dirichlet llocation." *Journal of machine learning research*, 3, 993-1022.

Bosu, A., and Carver. (2013). "Impact of Peer Code Review on Peer Impression Formation: A Survey.", *Proceeding of The International Conference on Empirical Software Engineering and Measurement,* Baltimore, Maryland, USA: 10-11 October, 2013.

Bosu, A., and Carver. (2014). "Impact of Developer Reputation on Code Review Outcomes in OSS projects: An Empirical Investigation.", *Preceding of The 8th ACM/IEEE International Conference on Empirical Software Engineering and Measurement*, Torino, Italy: 18 – 19 September, 2014.

Bosu, A., and Carver, J. (2013). "Peer Code Review to Prevent Security Vulnerabilities: An Empirical Evaluation.", *Proceeding of The 7th International Conference on Software Security and Reliability-Companion (SERE-C),* Gaithersburg, MD, USA: 18-20 June, 2013.

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

Bosu, A., Michaela, G., and Christian, B. (2015). "Characteristics of Useful Code Reviews: An Empirical Study at Microsoft.", *Proceeding of The 12th Working Conference on Mining Software Repositories (MSR),* Florence, Italy: 16-17 May, 2015.

Brose, O. (1965). "The atholic question in english politics, 1820 to 1830. G. I. T. machin." *The Journal of Modern History*, 37(2), 256-266.

Cinnéide, M., Yamashita, A., and Counsell, S. (2016). "Measuring refactoring benefits: a survey of the evidence.", *Proceeding of The 1st International Workshop on Software Refactoring*, Singapore, Singapore: 04 September, 2016.

Fagan, M. (1999). "Design and code inspections to reduce errors in program development." *IBM Systems Journal*, 38(2.3), 258-287.

Fellbaum, C. (1998). “A semantic network of english: the mother of all WordNets” *Computers and the Humanities*, 32(2/3), 209-220.

Fellbaum, C. (1998). *WordNe*t. MIT Press, Cambridge, MA.

Fowler, M., and Beck, K. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional.

Halloran, T., and William, L. (2002). “High quality and open source software Practices.”, *Proceeding of The 2nd Workshop on Open Source Software Engineering*, Orlando, Florida: 19 – 25 May, 2002.

Jeong, G., Kim, S., Zimmermann, T., & Yi, K. (2009). “Improving code review by predicting reviewers and acceptance of patches.”, *Technical Memorandum ROSAEC-2009-006, Research On Software Analysis for Error-free Computing Center,* Seoul National University, 2009.

Joshua Charles, C., Hindle, A., and Stroulia, E. (2014). “Latent Dirichlet allocation: extracting topics from software engineering data.” *The art and science of analyzing software data (1 ed.),* 1-21.

Koponen, T., and Hotti, V. (2005). “Open source software maintenance process Framework.” *ACM SIGSOFT Software Engineering Notes*, 30(4), 1-5.

Kuhn, A., Ducasse, S., and Gîrba, T. (2007). “Semantic clustering: Identifying topics in source code.”, *Information and Software Technology*, 49(3), 230-243.

Luca Milanesio. (2013). *Learning Gerrit Code Review*. Packt Publishing.

Lukins, S., Kraft, N., and Etzkorn, L. (2010). “Bug localization using Latent Dirichlet Allocation.” *Information and Software Technology*, 52(9), 972-990.

Martinfowler.com (n.d.). “Refactoring.” (Online) Available on ttp://martinfowler.com/ books/refactoring.html (25 July 2016).

Maskeri, G., Sarkar, S., and Heafield, K. (2008). “Mining Business Topics in Source Code Using Latent Dirichlet aAllocation.”, *Proceeding of The 1st Conference on India software engineering*, Hyderabad, India: 19-22 February, 2008.

McIntosh, S., Kamei, Y., Adams, B., and Hassan, A. (2014). “The Impact of Code Review and Code Review Participation on Software Quality A Case Study of the Qt, VTK, and ITK Projects.”, *Proceeding of The 11th Working Conference on Mining Software Repositories*, Hyderabad, India: 31 May – 01 June, 2014.

Mens, T., and Tourwe, T. (2004). "A survey of software refactoring." *IEEE Transactions On Software Engineering*, 30(2), 126-139.

Menzies, T., Williams, L., and Zimmermann, T. (2016). *Perspectives on data science for software engineering*. Morgan Kaufmann is an imprint of Elsevier, Cambridge, MA.

Midha, V., and Bhattacherjee, A. (2012). "Governance practices and software Maintenance: A study of open source projects." *Decision Support Systems*, 54(1), 23-32.

Ming-Wei Wu, and Ying-Dar Lin. (2001). "Open source software development: An Overview." *Computer*, 34(6), 33-38.

Moha, N., Guéhéneuc, Y., Meur, A., Duchien, L., and Tiberghien, A. (2009). "From a domain analysis to the specification and detection of code and design smells." *Formal Aspects of Computing*, 22(3), 345-361.

Munro, M. (2005). "Product metrics for automatic identification of bad smell design problems in java source-code.", *Proceeding of The* 11th International *Conference on Software Metrics,* Como, Italy: 19-22 September, 2005.

Navidi, W. (2010). *Principles of statistics for engineers and scientists*. McGraw-Hill, Dubuque, IA.

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

Ouni, A., Kessentini, M., and Sahraoui, H. (2013). "Search-Based Refactoring Using Recorded Code Changes.", *Proceeding of The 17th European Conference on Software Maintenance and Reengineering (CSMR),* Genova, Italy: 5-8 March, 2013.

Ouni, A., Kessentini, M., Sahraoui, H., and Hamdi, M. (2013). "The use of Development History in Software Refactoring Using a Multi-Objective Evolutionary Algorithm.", *Proceeding of The 15th annual Conference on Genetic and evolutionary computation*, Amsterdam, The Netherlands: 06-10 July, 2013.

Ouni, A., Kessentini, M., Sahraoui, H., Inoue, K., and Hamdi, M. (2015). "Improving multi - objective code-smells correction using development history." *Journal of Systems and Software*, 105, 18-39.

Paulson, J., 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.

Raja, U., and Barry, E. (2005). "Investigating quality in large-scale Open Source Software." *ACM SIGSOFT Software Engineering Notes*, 30(4), 1.

Raza, A., Capretz, L., and Ahmed, F. (2013). "Maintenance Support in Open Source Software projects.", *Proceeding of The 8th International Conference on Digital Information Management (ICDIM),* Islamabad, Pakistan: 10-12 September, 2013

Ripley, B. (2001). "The R Project in Statistical Computing." *MSOR Connections*, 1(1), 23-25.

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

Silva, D., Tsantalis, N., and Valente, M. (2016). "Why we Refactor? Confessions of GitHubcontributors.", *Proceeding of The 24th ACM SIGSOFT International Conference on Foundations of Software Engineering*, Seattle, WA, USA: 13 – 18 November, 2016.

Soares, G., Gheyi, R., Murphy-Hill, E., and Johnson, B. (2013). "Comparing approaches to analyze refactoring activity on software repositories. " *Journal of Systems and Software*, 86(4), 1006-1022.

Stamelos, I., Angelis, L., Oikonomou, A., and Bleris, G. (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?", *Proceeding of The 5th International Workshop Conference on Software Quality*, Leipzig, Germany: 20 – 26 May, 2007.

Tao, Y., DongGyun, H., and Sunghun, K. (2014). "Writing Acceptable Patches: An Empirical Study of Open Source Project Patches.", *Proceeding of The International Conference on Software Maintenance and Evolution,* Victoria, BC, Canada: 29 September -3 October, 2014.

Thongtanunam, P., Kula, R., Camargo Cruz, A., and Yoshida, N. (2014). "Improving Code Review Effectiveness through Reviewer Recommendations.", *Proceeding of The 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, Hyderabad, India: 02-03 June, 2014.

Thongtanunam, P., Tantithamthavorn, C., Kula, R., Yoshida, N., Iida, H., and Matsumoto, K. (2015). "Who Should Review my Code? A file Location-Based Code-Reviewer Recommendation Approach for Modern Code Review.", *Proceeding of The 22nd International Conference on Software Analysis, Evolution and Reengineering International (SANER),* Montreal, QC, Canada: 2-6 March, 2015.

VANCE, A. (2009). *Data analysts captivated by R's power*. New York Times 6.5.4, 1-3.

Willett, P. (2006). "The Porter stemming algorithm: then and now." *Program*, 40(3), 219- 223.

Witz, K., Hinkle, D., Wiersma, W., and Jurs, S. (1990). "Applied Statistics for the Behavioral Sciences." *Journal of Educational Statistics*, 15(1), 84.

Xia, X., Lo, D., Wang, X., and Yang, X. (2015). "Who should review this change? : Putting Text and file location analyses together for more accurate ecommendations.", *Proceeding of The International Conference on Software Maintenance and Evolution (ICSME),* Bremen, Germany: 29 September -1 October, 2015.

Xiong, C., Li, Y., Xie, M., Ng, S., and Goh, T. (2009). "A Model of Open Source Software Maintenance Activities.", *Proceeding of The International Conference on Industrial Engineering and Engineering Management (IEEM),* Hong Kong, China: 8-11 Dec ember, 2009.

Yamashita, A., and Moonen, L. (2013). "Do Developers Care About Code Smells? An Exploratory Survey.", *Proceeding of The Working Conference on Reverse Engineering*, Koblenz, Germany: 24 Jun, 2013.

Yoshida, N., Saika, T., Choi, E., Ouni, A., and Inoue, K. (2016). "Revisiting the Relationship Between Code Smells and Refactoring.", *Proceeding of The 24th International Conference on Program Comprehension,* Austin, TX, USA: 16-17 May, 2016.

Yu, L., SCHACH, S., and CHEN,, K. (2005). "Measuring the Maintainability of Open-Source Software.", *Proceeding of The International Conference on Empirical Software Engineering,* Noosa Heads, Qld., Australia: 17-18 November, 2005.

ZHANG, Z., MIAO, D., and GAO, C. (2013). "Short text classification using latent dirichlet allocation." *Journal of Computer Applications*, 33(6), 1587-1590.

Zhou, Y., and Davis, J. (2005). "Open source software reliability model." *ACM SIGSOFT Software Engineering Notes*, 30(4), 1.

ภาคผนวก

ภาคผนวก ก

โครงสร้างการเก็บข้อมูลในเกอริต

ตาราง people : ตารางจัดเก็บข้อมูลของผู้เข้าใช้งาน		
คุณลักษณะ	**คำอธิบาย**	**คีย์**
gerrit _id	ข้อมูลผู้ใช้งาน	PK
full_name	ชื่อผู้ใช้งาน	
preferred_email	อีเมล์ที่ใช้ในการติดต่อ	

ตาราง patches : ตารางจัดเก็บข้อมูลของซอร์สโค้ดที่จะทำการปรับปรุง		
คุณลักษณะ	**คำอธิบาย**	**คีย์**
patch_id	ข้อมูลของซอร์สโค้ดที่จะทำการปรับปรุง	PK
request_id	ข้อมูลการส่งคำร้องขอ	FK1
patchset_id	ข้อมูลของซอร์สโค้ดที่แก้ไข	
revision	การแก้ไข	
committer	ชื่อของผู้ที่ต้องการปรับปรุงซอร์สโค้ด	FK2
creates	วันเวลาของการปรับปรุง	

ตาราง reviews : ตารางเก็บข้อมูลการตรวจทานโค้ด		
คุณลักษณะ	**คำอธิบาย**	**คีย์**
id	หมายเลขลำดับข้อมูล	PK
request_id	ข้อมูลการส่งคำร้องขอ	FK1
people_id	ข้อมูลของผู้เข้าใช้งาน	FK2
verified	ค้นหาความผิดพลาด	
reviewed	ตรวจสอบ	
approved	อนุมัติ	
submitted	ยอมรับ	
verified_on	เวลาที่ค้นหาความผิดพลาด	
reviewed_on	เวลาที่ตรวจสอบ	
approved_on	เวลาที่อนุมัติ	
submitted_on	เวลาที่ยอมรับ	

ตาราง inline_comments : ตารางเก็บข้อมูลของข้อเสนอแนะในแต่ละบรรทัด		
คุณลักษณะ	**คำอธิบาย**	**คีย์**
request_id	ข้อมูลการส่งคำร้องขอ	FK1
patchset_id	ข้อมูลของซอร์สโค้ดที่แก้ไข	
file_name	ชื่อไฟล์	
uuid	ตัวเลขประจำตัวของข้อมูล	
line_number	หมายเลขบรรทัดของโค้ดที่มีการให้ข้อเสนอแนะ	
author_id	ข้อมูลของผู้ตรวจทานโค้ด	FK2
written_on	วันที่ทำการให้ข้อเสนอแนะ	
status	สถานะของข้อเสนอแนะ	
side	ตำแหน่งที่มีข้อเสนอแนะ	
message	ข้อเสนอแนะ	

ตาราง review_comments : ตารางเก็บข้อมูลของข้อเสนอแนะ		
คุณลักษณะ	**คำอธิบาย**	**คีย์**
comments _id	ข้อมูลของข้อเสนอแนะ	PK
request_id	ข้อมูลการส่งคำร้องขอ	FK1
patchset_id	ข้อมูลของซอร์สโค้ดที่แก้ไข	
author	ชื่อผู้ตรวจทานโค้ด	FK2
created	วันเวลาของการให้ข้อเสนอแนะ	
message	ข้อเสนอแนะที่นักพัฒนาให้คำแนะนำ	

ตาราง reguest_ detail : ตารางเก็บข้อมูลรายละเอียดของการส่งคำร้องขอ		
คุณลักษณะ	**คำอธิบาย**	**คีย์**
request_id	ข้อมูลการส่งคำร้องขอ	PK
owner_id	ข้อมูลของผู้ส่งคำร้องขอ	FK1
change_id	ข้อมูลของการเปลี่ยนแปลง	
project	ชื่อของโครงการ	
branch	เส้นทางการจัดเก็บข้อมูล	
uploaded	วันเวลาในการดึงข้อมูล	
updated	วันเวลาที่ได้ทำการอัพเดทข้อมูล	
status	สถานะของคำร้องขอ	
subject	หมวดหมู่หรือประเภทของการส่งคำร้องขอ	
commit_message	ข้อความที่ส่งไป	
topic	เรื่องที่ต้องการให้ทำการตรวจสอบ	
sort_key	ลำดับในการแก้ไข	
mergeable	การรวมโปรแกรมแก้ไข	
starred	เครื่องหมายที่ถูกแสดง	
number_patches	หมายเลขของโปรแกรมที่แก้ไข	
current_patcjest_id	ข้อมูลปัจจุบันของโปรแกรมแก้ไข	
author	ชื่อผู้ส่งคำร้องขอ	FK2

ตาราง patch_details : ตารางเก็บข้อมูลรายละเอียดของซอร์สโค้ดที่จะทำการปรับปรุง		
คุณลักษณะ	**คำอธิบาย**	**คีย์**
request_id	ข้อมูลการส่งคำร้องขอ	FK1
patchset_id	ข้อมูลของซอร์สโค้ดที่แก้ไข	
file_name	ชื่อไฟล์ที่ทำการเปลี่ยนแปลง	
change_type	ประเภทของของการเปลี่ยนแปลง	
patch_type	ประเภทของของซอร์สโค้ดที่จะทำการปรับปรุง	
nbr_comments	จำนวนของข้อเสนอแนะ	
nbr_drafts	จำนวนของการแก้ไข	
insertions	สิ่งที่เพิ่มเข้ามา	
deletions	สิ่งที่ถูกลบออกไป	

ภาคผนวก ข

โครงสร้างฐานข้อมูลที่ใช้ในการจัดเก็บข้อมูลสำหรับงานวิจัย

ตารางโครงสร้างของฐานข้อมูลที่ใช้ในการจัดเก็บข้อมูล

ตาราง temp_comment : ตารางจัดเก็บข้อมูลของข้อเสนอแนะ			
คุณลักษณะ	คำอธิบาย	ชนิด	คีย์
id	ลำดับของข้อมูล	varchar	PK
keywords	คำหลักที่ใช้ในการค้นหา	varchar	
message	ข้อเสนอแนะที่เกิดขึ้นจากการตรวจทานโค้ด	text	
project	ชื่อของโครงการ	varchar	
synonyms	คำที่มีความหมายสื่อถึงคำหลักที่มีการค้นพบ	varchar	
created_on	วันเวลาของการตรวจทาน	datetime	
reviewer	ชื่อของผู้ตรวจทานโค้ด	varchar	
author	ชื่อของเจ้าของซอร์สโค้ด	varchar	

ประวัติผู้เขียน

ชื่อ สกุล นางสาวอปัตตา ชัยสุตานนท์

รหัสประจำตัวนักศึกษา 5830223009

วุฒิการศึกษา

วุฒิ	ชื่อสถาบัน	ปีที่สำเร็จการศึกษา
วิทยาศาสตรบัณฑิต (วิศวกรรมซอฟต์แวร์)	มหาวิทยาลัยสงขลานครินทร์	2558

การตีพิมพ์เผยแพร่ผลงาน

อปัตตา ชัยสุตานนท์, อซีส นันทอมรพงศ์, รัตนา เวทย์ประสิทธิ์. (2559).“อิทธิพลของร่องรอยโค้ดที่ไม่ดีต่อการตรวจทานโค้ดในโครงการซอฟต์แวร์โอเพนซอร์ส”, งานประชุมวิชาการระดับชาติด้านคอมพิวเตอร์และเทคโนโลยีสารสนเทศ ครั้งที่ 12, โรงแรมอวานี ขอนแก่น: 7-8 กรกฎาคม 2559.