RELATIONSHIPS BETWEEN CODE SMELLS AND ARCHITECTURE PATTERNS IN ANDROID APPLICATIONS

Pitiporn PRASITPORN[1*] and Twittie SENIVONGSE[1]

1 Faculty of Engineering, Chulalongkorn University, Thailand; 6270163521@student.chula.ac.th (P. P.) (Corresponding Author)

ARTICLE HISTORY

Received: 21 April 2023 **Revised:** 10 May 2023 **Published:** 22 May 2023

ABSTRACT

Currently, many Android application architecture patterns, i.e. MVC, MVP, MVVM, and MVI, are commonly used. This research has two objectives: (1) to compare code smell density when using these patterns and (2) to analyze whether code smell density increases or decreases when the applications that use these patterns evolve, i.e. the number of commits in the project grows. The study analyzes data from 40 open-source projects, 10 projects for each pattern. Based on the ANOVA results for the first objective, the projects that use MVI have an average code smell density less than the projects that use other architecture patterns. Based on Pearson correlation analysis for the second objective, the projects that use MVP, MVVM, and MVI have very small negative correlation between number of commits and code smell density in each commit, whereas the projects that use MVC have very small positive correlation. Hence, the factor of number of commits in the project is negligible when considering code smell density in the project.

Keywords: Android Application, Architecture Pattern, Code Smell

CITATION INFORMATION: Prasitporn, P., & Senivongse, T. (2023). Relationships Between Code Smells and Architecture Patterns in Android Applications. *Procedia of Multidisciplinary Research, 1*(5), 44.

ความสัมพันธ์ระหว่างร่องรอยที่ไม่ดีในโค้ดและแบบรูปสถาปัตยกรรมของแอนดรอยด์แอปพลิเคชัน

ปิติพร ประสิทธิ์พร[1*] และ ทวิตีย์ เสนีวงศ์ ณ อยุธยา[1]

1 คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย; 6270163521@student.chula.ac.th (ปิติพร) (ผู้ประพันธ์บรรณกิจ)

บทคัดย่อ

ในปัจจุบันมีแบบรูปสถาปัตยกรรมของแอนดรอยด์แอปพลิเคชันหลายแบบที่ใช้กันแพร่หลาย ได้แก่ เอ็มวีซี เอ็มวีพี เอ็มวีวีเอ็ม และเอ็มวีไอ งานวิจัยนี้มีวัตถุประสงค์สองข้อคือ (1) เพื่อเปรียบเทียบความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดเมื่อใช้แบบรูปแต่ละแบบข้างต้น และ (2) เพื่อวิเคราะห์ความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดว่า เพิ่มขึ้นหรือลดลงเมื่อแอปพลิเคชันที่ใช้แบบรูปแต่ละแบบข้างต้นมีวิวัฒนาการหรือมีจำนวนคอมมิตในโครงการเพิ่มขึ้น การศึกษานี้วิเคราะห์ข้อมูลโครงการโอเพนซอร์ซ 40 โครงการ โดยแต่ละแบบรูปมี 10 โครงการ ผลการวิเคราะห์ความแปรปรวนตามวัตถุประสงค์ของงานวิจัยข้อแรกพบว่า โครงการที่ใช้เอ็มวีไอมีค่าเฉลี่ยร่องรอยที่ไม่ดีในโค้ดน้อยกว่าโครงการที่ใช้แบบรูปสถาปัตยกรรมอื่น จากการวิเคราะห์สหสัมพันธ์แบบเพียร์สันตามวัตถุประสงค์ของงานวิจัยข้อที่สองพบว่า โครงการที่ใช้เอ็มวีพี เอ็มวีวีเอ็ม และเอ็มวีไอ มีความสัมพันธ์ระหว่างจำนวนคอมมิตกับความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดแต่ละคอมมิตในเชิงลบในระดับที่น้อยมาก ส่วนโครงการที่ใช้เอ็มวีซีจะมีความสัมพันธ์ในเชิงบวกแต่ในระดับที่น้อยมากเช่นกัน จึงทำให้สามารถละเลยปัจจัยด้านจำนวนคอมมิตในโครงการได้ เมื่อพิจารณาถึงความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดของโครงการ

คำสำคัญ: แอนดรอยด์แอปพลิเคชัน, แบบรูปสถาปัตยกรรม, ร่องรอยที่ไม่ดีในโค้ด

ข้อมูลการอ้างอิง: ปิติพร ประสิทธิ์พร และ ทวิตีย์ เสนีวงศ์ ณ อยุธยา. (2566). ความสัมพันธ์ระหว่างร่องรอยที่ไม่ดีในโค้ดและแบบรูปสถาปัตยกรรมของแอนดรอยด์แอปพลิเคชัน. *Procedia of Multidisciplinary Research, 1*(5), 44.

บทนำ

การพัฒนาโปรแกรมประยุกต์สำหรับอุปกรณ์เคลื่อนที่ (Mobile Application) ได้กลายมาเป็นส่วนสำคัญในตลาดการพัฒนาซอฟต์แวร์ การขยายตัวของความต้องการซอฟต์แวร์สำหรับอุปกรณ์เคลื่อนที่นี้ ส่งผลให้ความเกิดความซับซ้อนและเกิดวิวัฒนาการของการพัฒนาซอฟต์แวร์อย่างรวดเร็ว ในมุมมองของนักพัฒนาซอฟต์แวร์เอง เมื่อต้องทำงานที่ซับซ้อนมากขึ้นเรื่อยๆ ตามความต้องการของผู้ใช้งานที่เพิ่มขึ้น ภายใต้ความกดดันด้านเวลา อาจก่อให้เกิดร่องรอยที่ไม่ดีในโค้ด (Code Smell) ได้ ซึ่งร่องรอยที่ไม่ดีในโค้ดนี้หมายถึงโครงสร้างของโค้ดที่เขียนไม่ดี สามารถนำไปสู่ปัญหาด้านประสิทธิภาพของซอฟต์แวร์ เช่น การใช้งานทรัพยากรฮาร์ดแวร์มากเกินไป หรือปัญหาด้านการบำรุงรักษา เช่น การทำความเข้าใจโค้ดเป็นไปได้ยาก

ระบบปฏิบัติการแอนดรอยด์ (Android Operating System) เป็นระบบปฏิบัติการที่ได้รับความนิยมสูงที่สุดในระบบปฏิบัติการสำหรับอุปกรณ์คลื่อนที่ ปัจจุบันอยู่ภายใต้การดูแลของกูเกิล (Google Inc.) ซึ่งทั้งกูเกิลและชุมชนของนักพัฒนาแอปพลิเคชันแอนดรอยด์ได้มีคำแนะนำเกี่ยวกับการออกแบบแอนดรอยด์แอปพลิเคชันให้มีประสิทธิภาพมากขึ้นหลากหลายแบบในหลายปีที่ผ่านมา หนึ่งในนั้นคือแนวทางในการออกแบบโครงสร้างของแอนดรอยด์แอปพลิเคชัน หรือที่เรียกว่า แบบรูปสถาปัตยกรรมของแอนดรอยด์แอปพลิเคชัน (Android Application Architecture Pattern) ซึ่งเป็นสิ่งที่นักพัฒนาแอปพลิเคชันแอนดรอยด์ควรจะให้ความสำคัญเพื่อให้แอปพลิเคชันมีประสิทธิภาพ สามารถทดสอบและสามารถบำรุงรักษาได้ ในปัจจุบันมีแบบรูปสถาปัตยกรรมของแอนดรอยด์แอปพลิเคชันหลายแบบที่ใช้กันแพร่หลาย ได้แก่ เอ็มวีซี (MVC) เอ็มวีพี (MVP) เอ็มวีวีเอ็ม (MVVM) และเอ็มวีไอ (MVI)

งานวิจัยนี้มีวัตถุประสงค์คือ (1) เพื่อเปรียบเทียบการใช้แบบรูปสถาปัตยกรรมของแอนดรอยด์แอปพลิเคชันแบบเอ็มวีซี เอ็มวีพี เอ็มวีวีเอ็ม และเอ็มวีไอ ว่ามีความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดต่างกันหรือไม่ และ (2) เพื่อวิเคราะห์ความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดว่า เพิ่มขึ้นหรือลดลงเมื่อแอปพลิเคชันที่ใช้แบบรูปแบบเอ็มวีซี เอ็มวีพี เอ็มวีวีเอ็ม และเอ็มวีไอ มีวิวัฒนาการหรือมีจำนวนคอมมิตในโครงการเพิ่มขึ้น การวิเคราะห์ทำโดยใช้ชุดข้อมูลซอฟต์แวร์โอเพนซอร์ซ (Open Source) ในระบบควบคุมเวอร์ชัน GitHub และใช้ชุดเครื่องมือ SNIFFER (Habchi, 2023) เพื่อตรวจหาร่องรอยที่ไม่ดีในโค้ดในแต่ละคอมมิตของซอฟต์แวร์ จากนั้นนำข้อมูลร่องรอยที่ไม่ดีของแอปพลิเคชันที่ใช้แบบรูปสถาปัตยกรรมของแอนดรอยด์แอปพลิเคชันต่างๆ มาเปรียบเทียบกันโดยใช้การวิเคราะห์ความแปรปรวนเพื่อหาว่า มีความแตกต่างของค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดหรือไม่ เมื่อมีการใช้แบบรูปสถาปัตยกรรมต่างๆ รวมทั้งทำการหาค่าสหสัมพันธ์ของความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดในแบบรูปสถาปัตยกรรมของแอนดรอยด์แอปพลิเคชันแต่ละแบบกับจำนวนคอมมิตในโครงการ เพื่อนำมาวิเคราะห์ว่า เมื่อโครงการมีวิวัฒนาการหรือมีการคอมมิตมากขึ้น จะมีผลต่อปริมาณความหนาแน่นของร่องรอยที่ไม่ดีที่เกิดขึ้นในโค้ดหรือไม่

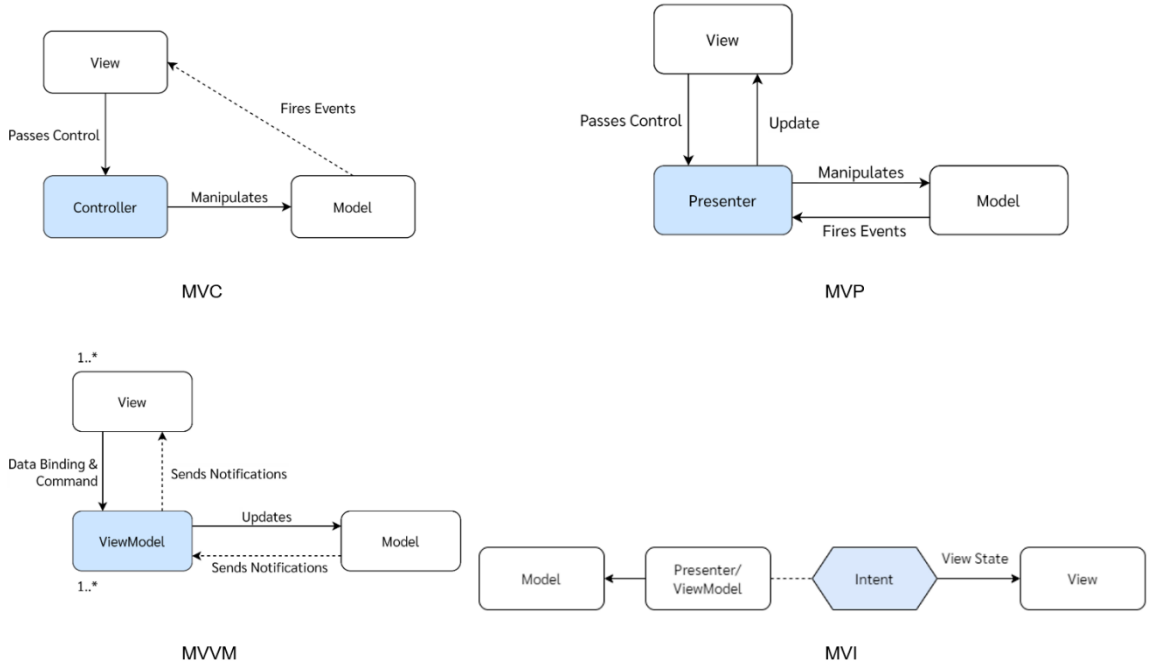
การเปรียบเทียบแบบรูปสถาปัตยกรรมของแอนดรอยด์แอปพลิเคชันโดยอิงจากร่องรอยที่ไม่ดีในโค้ดในงานวิจัยนี้สามารถเป็นหนึ่งในข้อมูลที่ช่วยในการตัดสินใจใช้แบบรูปสถาปัตยกรรมของแอนดรอยด์แอปพลิเคชันในการพัฒนาแอนดรอยด์แอปพลิเคชันในโครงการต่างๆ ที่ได้รับมอบหมายได้ เพราะการเลือกใช้แบบรูปสถาปัตยกรรมของแอนดรอยด์แอปพลิเคชัน จำเป็นต้องทำในตอนเริ่มพัฒนาโครงการ และแบบรูปสถาปัตยกรรมของแอนดรอยด์แอปพลิเคชันก็มีข้อดีและข้อเสียในแต่ละด้านต่างกัน ซึ่งความแตกต่างของการเกิดร่องรอยที่ไม่ดีในโค้ดเองก็เป็นส่วนที่ควรคำนึงถึงเมื่อต้องเลือกใช้แบบรูปสถาปัตยกรรมของแอนดรอยด์แอปพลิเคชัน

การทบทวนวรรณกรรม

แบบรูปสถาปัตยกรรมแอปพลิเคชันที่นิยมใช้ในปัจจุบันมี 4 แบบ (Saelor, 2023; Decanini, 2023) ดังภาพที่ 1 ดังนี้

1) แบบรูปสถาปัตยกรรมแบบเอ็มวีซี (MVC: Model-View-Controller Pattern) เป็นแบบรูปสถาปัตยกรรมเก่าแก่ที่สุดของแอนดรอยด์ ส่วนประกอบของแบบรูปนี้ ได้แก่ (1) โมเดล เป็นส่วนที่เก็บข้อมูลของแอปพลิเคชัน (2) วิว เป็นส่วนที่แสดงผลข้อมูล และ (3) คอนโทรลเลอร์ เป็นส่วนที่ทำหน้าที่เป็นตัวกลาง นำการกระทำ (Action) ในวิวไปจัดการกับ

ข้อมูลในโมเดล รวมถึงจัดการข้อมูลต่างๆ ในโมเดลเพื่อนำไปแสดงผลที่วิวด้วย อย่างไรก็ตาม คอนโทรลเลอร์จะมีการเรียกใช้งานบนแอคทิวิตีที่เป็นส่วนที่จัดการการแสดงผลบนแอนดรอยด์ จึงทำให้แอคทิวิตีมีการละเมิดกฎ Single Responsibility Principle อีกทั้งคอนโทรลเลอร์จะยึดติดกับวิวมากเกินไป ทำให้เมื่อแอปพลิเคชันมีขนาดใหญ่ขึ้น คอนโทรลเลอร์ก็จะมีขนาดใหญ่ขึ้นตาม ทำให้การดูแลรักษาทำได้ยากขึ้น



ภาพที่ 1 แบบรูปสถาปัตยกรรมของแอนดรอยด์แอปพลิเคชัน

2) แบบรูปสถาปัตยกรรมแบบเอ็มวีพี (MVP: Model-View-Presenter Pattern) มีการเพิ่มส่วนประกอบอื่นเพื่อแก้ไขปัญหาที่เกิดขึ้นกับแบบรูปเอ็มวีซี คือ ส่วนพรีเซนเตอร์ ซึ่งเป็นส่วนที่เก็บตรรกะการทำงานของระบบ (Business Logic) และทำหน้าที่เป็นตัวกลางระหว่างโมเดลและวิว คล้ายกับคอนโทรลเลอร์ของเอ็มวีซี แต่พรีเซนเตอร์จะไม่ได้อยู่บนแอคทิวิตีเช่นเดียวกับคอนโทรลเลอร์ แต่จะแยกออกมาและเรียกวิวผ่านอินเทอร์เฟซ (Interface) แทน ทำให้วิวและพรีเซนเตอร์เป็นอิสระจากกัน อย่างไรก็ตาม เมื่อแอปพลิเคชันมีขนาดใหญ่ขึ้น พรีเซนเตอร์ก็ยังคงมีขนาดใหญ่ขึ้นตามไปด้วย อีกทั้งแบบรูปเอ็มวีพียังอาจจะมีการเขียนโค้ดที่มากกว่าแบบรูปอื่น เพราะต้องมีการเขียนโค้ดเพื่อรองรับการกระทำจากวิว และต้องเขียนโค้ดเพื่อแสดงผลบนวิวด้วย

3) แบบรูปสถาปัตยกรรมแบบเอ็มวีวีเอ็ม (MVVM: Model-View-ViewModel) มีการเพิ่มส่วนประกอบอื่น คือ วิวโมเดล (ViewModel) เข้ามา ซึ่งทำหน้าที่เก็บข้อมูลและจัดการข้อมูลที่วิวต้องการใช้ โดยระหว่างวิวและวิวโมเดลจะถูกอิมพลีเมนต์ด้วย DataBinding ซึ่งเมื่อวิวมีการเปลี่ยนแปลงก็จะส่งผลถึงวิวโมเดลด้วย เช่นเดียวกับที่เมื่อวิวโมเดลมีการเปลี่ยนแปลงก็จะส่งผลถึงวิวด้วยเช่นกัน การใช้ DataBinding ทำให้จำนวนโค้ดในแบบรูปเอ็มวีวีเอ็มมีจำนวนน้อยกว่าโค้ดในแบบรูปเอ็มวีพี ในแบบรูปเอ็มวีวีเอ็ม วิวจะถือ Reference ของวิวโมเดลไว้ แต่วิวโมเดลจะไม่รู้จักวิวเลย ดังนั้นความสัมพันธ์ระหว่างวิวโมเดลกับวิวจะเป็นแบบหนึ่งต่อกลุ่ม (One-to-Many) ทำให้แอคทิวิตีหลายๆ ตัวสามารถเรียกใช้วิวโมเดลตัวเดียวได้ แต่ในแบบรูปสถาปัตยกรรมแบบนี้วิวและโมเดลจะไม่สามารถติดต่อกันได้เลย

4) แบบรูปสถาปัตยกรรมแบบเอ็มวีไอ (MVI: Model-View-Intent) เป็นหนึ่งในแบบรูปสถาปัตยกรรมที่ใหม่ที่สุดสำหรับแอนดรอยด์ ซึ่งแตกต่างจากสถาปัตยกรรมแบบอื่นคือ มีการเพิ่มการจัดการสถานะ (State Management) ระหว่างวิวกับพรีเซนเตอร์ (หรือวิวโมเดล) อินเท็นท์จะอ้างถึงขั้นตอนการทำงานทั้งหมด เริ่มจากการที่พรีเซนเตอร์ติดต่อให้โมเดลรับข้อมูลเพื่ออัปเดตสถานะของวิว แล้วส่งสถานะของวิวที่อัปเดตแล้วไปที่วิวเพื่อแสดงผล อินเท็นท์นี้สามารถเริ่มได้

จากพรีเซนเตอร์ หรือเริ่มจากการที่ผู้ใช้มีการกระทำบางอย่าง เช่น กดปุ่มบนหน้าจอ และเมื่อวิวทำการแสดงผลสถานะของวิวเรียบร้อยแล้ว ก็จะถือว่าอินเท็นท์ทำงานได้จบการทำงานแล้ว ประโยชน์ของแบบรูปสถาปัตยกรรมแบบนี้คือ มีเพียง 1 อ็อบเจกต์เท่านั้นที่ต้องจัดการ อีกหนึ่งแนวคิดของแบบรูปแบบเอ็มวีไอ คือ State Reducers ซึ่งจะทำการผสานข้อมูลใหม่กับสถานะของวิวเดิม อย่างไรก็ตามเมื่อแอคทิวิตีมีความซับซ้อนมากๆ State Reducers ก็จะซับซ้อนมากขึ้นเช่นกัน ซึ่งก็จะทำให้การบำรุงรักษาทำได้ยากขึ้นด้วยเช่นกัน

ในแอนดรอยด์แอปพลิเคชันมักพบร่องรอยที่ไม่ดีในโค้ด (Code Smell) (Fowler, 1999) ซึ่งหมายถึงลักษณะของโค้ดที่เขียนไม่ดี โค้ดส่วนนั้นจะเข้าใจได้ยาก ซึ่งอาจนำไปสู่การเขียนโค้ดที่เกิดข้อผิดพลาด (Bug) ในอนาคตได้ ร่องรอยที่ไม่ดีในโค้ดสามารถใช้วัดคุณภาพของโค้ดได้ เพราะการมีร่องรอยที่ไม่ดีในโค้ดหมายถึงการที่โค้ดส่วนนั้นจะยากต่อการแก้ไขและการนำกลับมาใช้ใหม่ คำว่าร่องรอยที่ไม่ดีในโค้ดมีจุดเริ่มต้นมาจากโปรแกรมภาษาเชิงวัตถุ (Object-Oriented Program) สำหรับอุปกรณ์เคลื่อนที่ นอกจากการทำงานของแอปพลิเคชันจะต้องทำงานได้อย่างราบรื่นและมีความน่าเชื่อถือแล้ว ยังต้องไม่ใช้แบตเตอรี่มากเกินไปด้วย Jan Reimann, Brylski, & Assmann (2014) จึงได้จัดทำแค็ตตาล็อกร่องรอยที่ไม่ดีในโค้ดของแอนดรอยด์ (Android Smells Catalogue) ซึ่งอาจจะส่งผลเสียต่อคุณสมบัติเหล่านั้นได้ ตัวอย่างร่องรอยที่ไม่ดีในโค้ดของแอนดรอยด์แอปพลิเคชันแสดงในตารางที่ 1 และตัวอย่างร่องรอยที่ไม่ดีในโค้ดเชิงวัตถุแสดงในตารางที่ 2

ตารางที่ 1 ตัวอย่างร่องรอยที่ไม่ดีในโค้ดของแอนดรอยด์

ร่องรอยที่ไม่ดีในโค้ดของแอนดรอยด์	**คำอธิบาย**
Internal Getter/Setter (IGS)	Internal fields are accessed via getters and setters. But in Android, virtual methods are expensive. (Brylski, 2013)
Member Ignoring Method (MIM)	Non-static methods do not access an object attribute. It is recommended to use a static method. (Brylski, 2013)
Leaking Inner Class (LIC)	Non-static inner classes hold a reference to the outer class. This could lead to a memory leak. (Brylski, 2013)

ตารางที่ 2 ตัวอย่างร่องรอยที่ไม่ดีในโค้ดเชิงวัตถุ

ร่องรอยที่ไม่ดีในโค้ดเชิงวัตถุ	**คำอธิบาย**
Blob Class (BLOB)/ God Class	A class with large number of attributes and/or operations. The Blob class handles a lot of responsibilities compared to other classes. (Brown, Malveau, McCormick, & Mowbray, 1998)
Swiss Army Knife (SAK)	A class with numerous interface signatures, resulting in a very complex class interface designed to handle a wide diversity of abstractions. (Source Making, 2023)
Long Method (LM)	Methods are implemented with many more lines of code than other methods. (Fowler, 1999)
Complex Class (CC)	A class containing complex methods, where complexity refers to cyclomatic complexity. (Sobrinho & Maia, 2021)

งานวิจัยนี้ได้รับแรงบันดาลใจจากงานวิจัยของ Walter & Alkhaeir (2016) เป็นอย่างมาก ซึ่งงานวิจัยดังกล่าวศึกษาความสัมพันธ์ระหว่างแบบรูปการออกแบบ (Design Patterns) กับร่องรอยที่ไม่ดีในโค้ด โดยศึกษาว่า (1) การ

ใช้แบบรูปการออกแบบสัมพันธ์กับการปรากฏของร่องรอยที่ไม่ดีในโค้ดหรือไม่ อย่างไร (2) ความสัมพันธ์เหล่านั้นมีการเปลี่ยนแปลงหรือไม่ในช่วงที่โค้ดมีวิวัฒนาการ และ (3) ความสัมพันธ์ระหว่างแบบรูปการออกแบบแต่ละแบบกับร่องรอยที่ไม่ดีในโค้ดแต่ละแบบเป็นอย่างไร โดยที่เลือกศึกษาแบบรูปการออกแบบ 9 แบบ และร่องรอยที่ไม่ดีในโค้ด 7 ประเภท กับโครงการโอเพนซอร์ซภาษาจาวาจำนวน 2 โครงการ จากงานวิจัยดังกล่าวทำให้ผู้วิจัยได้แนวคิดที่จะหาความสัมพันธ์ระหว่างแบบรูปสถาปัตยกรรมกับร่องรอยที่ไม่ดีในโค้ดของแอนดรอยด์แอปพลิเคชัน เนื่องจากแอนดรอยด์แอปพลิเคชันมีแบบรูปสถาปัตยกรรมและร่องรอยที่ไม่ดีในโค้ดในแบบเฉพาะของตัวเอง

นอกจากนี้ยังมีอีกหลายงานวิจัยที่ศึกษาเกี่ยวกับร่องรอยที่ไม่ดีในแอนดรอยด์แอปพลิเคชัน ตัวอย่างเช่น งานวิจัยของ Hecht, Rouvoy, Moha, & Duchien (2015) มีการนำเสนอเครื่องมือชื่อ Paprika ซึ่งเป็นเครื่องมือที่ใช้วิเคราะห์แอนดรอยด์แอปพลิเคชันเพื่อตรวจหาร่องรอยที่ไม่ดีในโค้ดจากไบต์โค้ด (Bytecode) ทั้งร่องรอยที่ไม่ดีในโค้ดของแอนดรอยด์และร่องรอยที่ไม่ดีในโค้ดเชิงวัตถุ โดยศึกษาว่า ในแอปพลิเคชัน 15 โครงการที่นำมาวิเคราะห์พบร่องรอยที่ไม่ดีในโค้ดของแอนดรอยด์และร่องรอยที่ไม่ดีในโค้ดเชิงวัตถุประเภทใดบ้าง และพบมากน้อยต่างกันอย่างไร ส่วนงานวิจัยของ Habchi, Rouvoy, & Moha (2019) เป็นงานวิจัยเกี่ยวกับการดำรงอยู่ของร่องรอยที่ไม่ดีในโค้ดของแอนดรอยด์แอปพลิเคชัน โดยศึกษาว่า นานเท่าใดที่ร่องรอยที่ไม่ดีในโค้ดของแอนดรอยด์ดำรงอยู่บนโค้ด และปัจจัยอะไรที่มีผลต่อการดำรงอยู่ของร่องรอยที่ไม่ดีในโค้ด การตรวจหาและติดตามร่องรอยที่ไม่ดีในโค้ดของแอนดรอยด์จากทุกคอมมิตของโครงการทำโดยใช้ชุดเครื่องมือ SNIFFER (Habchi, 2023) และเลือกใช้ข้อมูลจาก FDROID Repository จำนวน 324 โครงการ ชุดเครื่องมือ SNIFFER มีพื้นฐานอยู่บนเครื่องมือ Paprika ที่เดิมวิเคราะห์ร่องรอยที่ไม่ดีจาก APK โดยขยายความสามารถให้สามารถวิเคราะห์จากซอร์ซโค้ดได้และสามารถติดตามร่องรอยที่ไม่ดีในโค้ดของแอนดรอยด์จากทุกคอมมิตของโครงการได้ ผู้วิจัยจึงเลือกใช้ชุดเครื่องมือ SNIFFER ในงานวิจัยนี้

วิธีดำเนินการวิจัย

การศึกษาเชิงประจักษ์เกี่ยวกับความสัมพันธ์ระหว่างร่องรอยที่ไม่ดีในโค้ดและแบบรูปสถาปัตยกรรมของแอนดรอยด์แอปพลิเคชันมีขั้นตอนการทำงานทั้งหมด 4 ขั้นตอน ดังนี้

1) กำหนดข้อมูลโครงการที่ใช้ในการวิจัย

ขั้นตอนนี้ทำการเลือกโครงการแอนดรอยด์บน GitHub ซึ่งจะนำมาวิเคราะห์จำนวน 40 โครงการ แบ่งเป็นโครงการที่ใช้แบบรูปสถาปัตยกรรมแบบเอ็มวีซี เอ็มวีพี เอ็มวีวีเอ็ม และเอ็มวีไอ อย่างละ 10 โครงการ แต่ละโครงการมีอย่างน้อย 100 คอมมิต การเลือกกลุ่มตัวอย่างของโครงการเป็นแบบเจาะจง (Purposive Sampling) โดยผู้วิจัยคัดเลือก (Manual Inspection) จากการตั้งชื่อไฟล์ในโครงการซึ่งสอดคล้องกับส่วนประกอบของแบบรูปสถาปัตยกรรมแต่ละแบบ และมีการตรวจสอบภายในโค้ดเพิ่มเติมจากผู้วิจัยว่ามีการทำงานตามแบบรูปสถาปัตยกรรมตามที่ระบุไว้จริง

ภาพรวมของข้อมูล 40 โครงการที่นำมาใช้แสดงในตารางที่ 3

ตารางที่ 3 สรุปภาพรวมของโครงการ

แบบรูปสถาปัตยกรรม	จำนวนโครงการ	จำนวนคอมมิตในโครงการ (ต่ำสุด - สูงสุด)	จำนวนพันบรรทัดของโค้ด (ต่ำสุด - สูงสุด)
MVC	10	110 - 2324	11.7 - 140.6
MVP	10	101 - 1080	7.3 - 42.7
MVVM	10	171 - 3099	5.8 - 108.3
MVI	10	218 - 2385	7.6 - 93.3

2) กำหนดร่องรอยที่ไม่ดีในโค้ดที่นำมาใช้ในงานวิจัย

ผู้วิจัยเลือกศึกษาร่องรอยที่ไม่ดีในโค้ดของแอนดรอยด์แอปพลิเคชัน 3 ประเภท ตามตารางที่ 1 ข้างต้น ได้แก่ Internal Getter/Setter (IGS), Member Ignoring Method (MIM) และ Leaking Inner Class (LIC) และร่องรอยที่ไม่ดีในโค้ดเชิงวัตถุ 4 ประเภท ตามตารางที่ 2 ข้างต้น ได้แก่ Blob Class (BLOB), Swiss Army Knife (SAK), Long Method (LM) และ Complex Class (CC) ซึ่งเป็นร่องรอยที่ไม่ดีที่มีแนวโน้มเกิดขึ้นบ่อยและน่าจะมีความเกี่ยวข้องกับแบบรูปสถาปัตยกรรมของแอนดรอยด์แอปพลิเคชัน เนื่องจากผู้วิจัยเห็นว่า เมื่อมีการแบ่งแยกส่วนการทำงานที่ชัดเจนขึ้น ร่องรอยที่ไม่ดีในโค้ดหลายประเภทน่าจะลดลงได้ เช่น ร่องรอยที่ไม่ดีในโค้ดประเภท Blob Class (BLOB), Swiss Army Knife (SAK), Long Method (LM) และ Complex Class (CC) หรือการที่สถาปัตยกรรมบางแบบรูปมีการอัปเดตวิวโดยที่วิวไม่จำเป็นต้องรู้จักส่วนที่เก็บตรรกะการทำงานของระบบ จึงน่าจะสามารถลดร่องรอยที่ไม่ดีในโค้ดประเภท Internal Getter/Setter (IGS), Member Ignoring Method (MIM) และ Leaking Inner Class (LIC) ได้

3) ตรวจหาความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดของโครงการทั้งหมด

ผู้วิจัยใช้ชุดเครื่องมือ SNIFFER ซึ่งสามารถตรวจหาร่องรอยที่ไม่ดีในโค้ดของแอนดรอยด์แอปพลิเคชันได้จากทุกคอมมิตในโครงการ อย่างไรก็ตาม ในการใช้ชุดเครื่องมือ SNIFFER กับ 40 โครงการ ไม่พบร่องรอยที่ไม่ดีในโค้ดประเภท Long Method (LM) การวิเคราะห์ในขั้นตอนถัดไปจึงใช้เฉพาะข้อมูลร่องรอยที่ไม่ดีในโค้ดประเภทอื่นที่พบเท่านั้น และเนื่องจากแต่ละโครงการมีขนาด (จำนวนบรรทัดของโค้ด) แตกต่างกัน ดังนั้นหลังจากตรวจหาร่องรอยที่ไม่ดีแล้วจะทำการคำนวณหาความหนาแน่นของร่องรอยที่ไม่ดีในโค้ด เพื่อนำไปใช้วิเคราะห์ต่อไป โดยคำนวณจากการนำจำนวนร่องรอยที่ไม่ดีในโค้ดมาหารด้วยจำนวนหนึ่งพันบรรทัดของโค้ด

4) วิเคราะห์ความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดของโครงการที่ใช้แบบรูปสถาปัตยกรรมแต่ละแบบ

การวิเคราะห์ข้อมูลจะใช้โปรแกรม Jupyter Notebook และภาษา R โดยการทดลองและสถิติที่นำมาวิเคราะห์ข้อมูลเป็นดังนี้

4.1. การเปรียบเทียบการใช้แบบรูปสถาปัตยกรรมของแอนดรอยด์แอปพลิเคชันแบบต่างๆ ในโครงการว่า มีความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดต่างกันหรือไม่ ตามวัตถุประสงค์ข้อที่ 1 ของงานวิจัย เป็นการเปรียบเทียบค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดที่ปรากฏเมื่อมีการใช้แบบรูปสถาปัตยกรรมแบบต่างๆ ในโครงการ โดยใช้การวิเคราะห์ความแปรปรวนทางเดียว (One-Way ANOVA) (กัลยา วานิชย์บัญชา, 2542) ซึ่งความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดในการทดลองนี้จะเป็นความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดของคอมมิตล่าสุดในแต่ละโครงการ และหากพบว่า มีค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดของโครงการที่ใช้แบบรูปสถาปัตยกรรมอย่างน้อย 1 แบบที่มีความแตกต่างจากแบบอื่นๆ ผู้วิจัยจะวิเคราะห์เพิ่มเติมโดยใช้การเปรียบเทียบพหุคูณด้วยวิธี Scheffe's Post Hoc Comparison (Scheffe, 1999) เพื่อเปรียบเทียบความแตกต่างเป็นรายคู่ว่า การใช้แบบรูปสถาปัตยกรรมคู่ใดบ้างที่มีค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดของโครงการที่แตกต่างกัน

4.2. การวิเคราะห์ความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดว่าเพิ่มขึ้นหรือลดลง เมื่อโครงการที่ใช้แบบรูปแบบต่างๆ มีวิวัฒนาการหรือมีจำนวนคอมมิตเพิ่มขึ้น ตามวัตถุประสงค์ข้อที่ 2 ของงานวิจัย เป็นการทดสอบความสัมพันธ์ระหว่างจำนวนคอมมิตของโครงการกับความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดรวมทุกประเภทในโครงการที่ใช้แบบรูปสถาปัตยกรรมแบบต่างๆ โดยใช้สัมประสิทธิ์สหสัมพันธ์ของเพียร์สัน (กัลยา วานิชย์บัญชา, 2542)

ผลการวิจัย

1) การเปรียบเทียบค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดที่ปรากฏเมื่อมีการใช้แบบรูปสถาปัตยกรรมแบบต่าง ๆ ในโครงการ

1.1. การเปรียบเทียบค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ด เมื่อพิจารณาร่องรอยที่ไม่ดี รวมทุกประเภท มีสมมติฐานคือ

H_0: ค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดรวมทุกประเภทในโครงการที่ใช้แบบรูปสถาปัตยกรรมแต่ละแบบไม่แตกต่างกัน

H_1: มีโครงการที่ใช้แบบรูปสถาปัตยกรรมอย่างน้อย 1 แบบ ที่มีค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดรวมทุกประเภทที่แตกต่างจากแบบอื่น

หรือ

H_0: $\mu_{MVC} = \mu_{MVP} = \mu_{MVVM} = \mu_{MVI}$

H_1: $\mu_i \neq \mu_j$ อย่างน้อย 1 คู่ โดยที่ $i \neq j$

เมื่อนำความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดรวมทุกประเภทในโครงการที่มีการใช้แบบรูปสถาปัตยกรรมแต่ละแบบมาทำการวิเคราะห์ความแปรปรวน พบว่า มีค่า P-value เป็น 0.0008 ดังนั้นจึงปฏิเสธสมมติฐาน H_0 ที่ระดับนัยสำคัญ 0.05 เนื่องจาก P-value มีค่าน้อยกว่าระดับนัยสำคัญ นั่นคือ มีโครงการที่ใช้แบบรูปสถาปัตยกรรมอย่างน้อย 1 แบบ ที่มีค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดรวมทุกประเภทแตกต่างจากแบบอื่น

ผู้วิจัยทำการวิเคราะห์เพิ่มเติมเพื่อเปรียบเทียบความแตกต่างเป็นรายคู่ว่า การใช้แบบรูปสถาปัตยกรรมคู่ใดบ้างที่มีค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดรวมทุกประเภทแตกต่างกันด้วยวิธี Scheffe ได้ผลดังภาพที่ 2 พบว่า โครงการที่ใช้แบบรูปสถาปัตยกรรมแบบเอ็มวีไอมีค่า P-value รายคู่กับแบบรูปเอ็มวีซีเป็น 0.0063, กับแบบรูปเอ็มวีพีเป็น 0.0043 และ กับแบบรูปเอ็มวีวีเอ็มเป็น 0.0236 ซึ่งน้อยกว่าระดับนัยสำคัญ 0.05 ทั้งหมดทุกคู่ ประกอบกับเมื่อพิจารณาค่าความแตกต่าง (diff) ของค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดรวมทุกประเภทร่วมด้วยแล้ว จะพบว่า โครงการที่ใช้แบบรูปสถาปัตยกรรมแบบเอ็มวีไอมีค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดรวมทุกประเภทน้อยกว่าโครงการที่ใช้แบบรูปสถาปัตยกรรมอื่นทั้งหมด โดยที่น้อยกว่าแบบรูปเอ็มวีวีเอ็มน้อยที่สุด (ความแตกต่างอยู่ที่ 11.9530) ตามด้วยแบบรูปเอ็มวีซี (ความแตกต่างอยู่ที่ 13.9332) และน้อยกว่าแบบรูปเอ็มวีพีมากที่สุด (ความแตกต่างอยู่ที่ 14.4548) อย่างไรก็ตาม เมื่อพิจารณาเฉพาะแบบรูปเอ็มวีวีเอ็ม เอ็มวีซี และเอ็มวีพี ค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดรวมทุกประเภทในโครงการที่ใช้สามแบบรูปนี้ไม่แตกต่างกันที่ระดับนัยสำคัญ 0.05

```
  Posthoc multiple comparisons of means: Scheffe Test
    95% family-wise confidence level

$architecture
              diff     lwr.ci    upr.ci   pval
mvi-mvc  -13.93324 -24.661813 -3.204667 0.0063 **
mvp-mvc    0.52158 -10.206993 11.250153 0.9992
mvvm-mvc  -1.98023 -12.708803  8.748343 0.9608
mvp-mvi   14.45482   3.726247 25.183393 0.0043 **
mvvm-mvi  11.95301   1.224437 22.681583 0.0236 *
mvvm-mvp  -2.50181 -13.230383  8.226763 0.9252

---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '
' 1
```

ภาพที่ 2 ผลการทดสอบ Sheffe ของค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดรวมทุกประเภท

1.2. การเปรียบเทียบค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ด เมื่อพิจารณาร่องรอยที่ไม่ดี แยกประเภท มีสมมติฐานคือ

H_0: ค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดประเภท X ในโครงการที่ใช้แบบรูปสถาปัตยกรรมแต่ละแบบไม่แตกต่างกัน

H_1: มีโครงการที่ใช้แบบรูปสถาปัตยกรรมอย่างน้อย 1 แบบ ที่มีค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดประเภท X ที่แตกต่างจากแบบอื่น

หรือ

H_0: $\mu_{MVC(X)} = \mu_{MVP(X)} = \mu_{MVVM(X)} = \mu_{MVI(X)}$

H_1: $\mu_i \neq \mu_j$ อย่างน้อย 1 คู่ โดยที่ $i \neq j$

และ X หมายถึง ร่องรอยที่ไม่ดีในโค้ดประเภทใดประเภทหนึ่ง

เมื่อนำความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดแต่ละประเภทในโครงการที่มีการใช้แบบรูปสถาปัตยกรรมแต่ละแบบมาทำการวิเคราะห์ความแปรปรวน พบว่า มีค่า P-value ดังตารางที่ 4 ดังนั้นจึงไม่ปฏิเสธสมมติฐาน H_0 ที่ระดับนัยสำคัญ 0.05 สำหรับกรณีร่องรอยที่ไม่ดีในโค้ดประเภท Internal Getter/Setter (IGS), Member Ignoring Method (MIM), Blob Class (BLOB), Swiss Army Knife (SAK) และ Complex Class (CC) เนื่องจากค่า P-value มีค่ามากกว่าระดับนัยสำคัญ นั่นคือ ค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดแต่ละประเภทดังกล่าวในโครงการที่ใช้แบบรูปสถาปัตยกรรมแต่ละแบบไม่แตกต่างกัน

ตารางที่ 4 ผลการวิเคราะห์ความแปรปรวนของความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดแต่ละประเภทสำหรับแบบรูปสถาปัตยกรรมแต่ละแบบ

ร่องรอยที่ไม่ดีในโค้ด (X)	P-Value
Internal Getter/Setter (IGS)	0.0793
Member Ignoring Method (MIM)	0.0952
Leaking Inner Class (LIC)	0.0149
Blob Class (BLOB)	0.4387
Swiss Army Knife (SAK)	0.0861
Complex Class (CC)	0.1096

สำหรับกรณีร่องรอยที่ไม่ดีในโค้ดประเภท Leaking Inner Class (LIC) มีค่า P-value น้อยกว่าระดับนัยสำคัญ 0.05 ดังนั้นจึงปฏิเสธสมมติฐาน H_0 นั่นคือ มีโครงการที่ใช้แบบรูปสถาปัตยกรรมอย่างน้อย 1 แบบ ที่มีค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดประเภท Leaking Inner Class ที่แตกต่างจากแบบอื่น ผู้วิจัยจึงทำการวิเคราะห์เพิ่มเติมเพื่อเปรียบเทียบความแตกต่างเป็นรายคู่ว่า การใช้แบบรูปสถาปัตยกรรมคู่ใดบ้างที่มีค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดประเภท Leaking Inner Class ที่แตกต่างกันด้วยวิธี Scheffe พบว่า มีความแตกต่างกันเพียงคู่เดียวคือ โครงการที่ใช้แบบรูปสถาปัตยกรรมแบบเอ็มวีไอมีค่าเฉลี่ยร่องรอยที่ไม่ดีในโค้ดประเภท Leaking Inner Class แตกต่างจากโครงการที่ใช้แบบรูปสถาปัตยกรรมแบบเอ็มวีพีที่ระดับนัยสำคัญ 0.05 (ค่า P-value เป็น 0.0177) ประกอบกับเมื่อพิจารณาค่าความแตกต่าง (diff) ของค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดประเภท Leaking Inner Class ร่วมด้วยแล้ว จะพบว่า โครงการที่ใช้แบบรูปสถาปัตยกรรมแบบเอ็มวีไอมีค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดประเภท Leaking Inner Class น้อยกว่าโครงการที่ใช้แบบรูปสถาปัตยกรรมแบบเอ็มวีพี (ความแตกต่างอยู่ที่ 12.72264)

2) การทดสอบความสัมพันธ์ระหว่างจำนวนคอมมิตของโครงการกับความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดรวมทุกประเภทในโครงการที่ใช้แบบรูปสถาปัตยกรรมแบบต่าง ๆ

งานวิจัยนี้วิเคราะห์สหสัมพันธ์แบบเพียร์สันเพื่อทดสอบความสัมพันธ์เชิงเส้นระหว่างสองตัวแปร ได้แก่ (1) ตัวเลขจำนวนนับของคอมมิตที่เกิดขึ้นในโครงการ (ได้แก่ คอมมิต 1, 2, 3, ... ซึ่งตัวเลขที่เพิ่มขึ้นจะสะท้อนปริมาณการมีวิวัฒนาการ) และ (2) ความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดรวมทุกประเภทในคอมมิตนั้นๆ ของโครงการที่ใช้แบบรูปสถาปัตยกรรมแบบต่างๆ

ผู้วิจัยคำนวณค่าสัมประสิทธิ์สหสัมพันธ์ระหว่างสองตัวแปรข้างต้นสำหรับแต่ละโครงการ จากนั้นนำค่าสัมประสิทธิ์สหสัมพันธ์ของโครงการต่างๆ ที่ใช้แบบรูปสถาปัตยกรรมแบบเดียวกันมาหาค่าเฉลี่ย ตารางที่ 5 แสดงค่าเฉลี่ยของสัมประสิทธิ์สหสัมพันธ์ของโครงการต่างๆ ที่ใช้แบบรูปสถาปัตยกรรมแต่ละแบบ

ตารางที่ 5 ค่าเฉลี่ยของสัมประสิทธิ์สหสัมพันธ์ของโครงการต่างๆ ที่ใช้แบบรูปสถาปัตยกรรมแบบเดียวกัน

แบบรูปสถาปัตยกรรมที่โครงการใช้	ค่าเฉลี่ยสัมประสิทธ์สหสัมพันธ์ (r)
MVC	0.2056
MVP	-0.2341
MVVM	-0.2262
MVI	-0.1023

จากค่าเฉลี่ยของสัมประสิทธิ์สหสัมพันธ์ที่ได้ สำหรับโครงการที่ใช้แบบรูปสถาปัตยกรรมแบบเอ็มวีซี จะพบว่า จำนวนคอมมิตของโครงการกับความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดรวมทุกประเภท มีความสัมพันธ์เชิงเส้นในเชิงบวก แต่ระดับความสัมพันธ์น้อยมาก (ค่า r เป็น 0.2056) ในทางตรงกันข้าม หากโครงการใช้แบบรูปแบบเอ็มวีพี เอ็มวีวีเอ็ม และ เอ็มวีไอ จะพบว่า จำนวนคอมมิตของโครงการกับความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดรวมทุกประเภท มีความสัมพันธ์เชิงเส้นในเชิงลบ แต่ระดับความสัมพันธ์น้อยมากเช่นกัน (ค่า r เป็น -0.2341, -0.2262 และ -0.1023 ตามลำดับ)

สรุปและอภิปรายผลการวิจัย

จากการทดลองเพื่อศึกษาความสัมพันธ์ระหว่างร่องรอยที่ไม่ดีในโค้ดประเภท Internal Getter/Setter (IGS), Member Ignoring Method (MIM), Leaking Inner Class (LIC), Blob Class (BLOB), Swiss Army Knife (SAK) และ Complex Class (CC) ที่ปรากฏในแอนดรอยด์แอปพลิเคชันในโครงการที่ใช้แบบรูปสถาปัตยกรรมแบบเอ็มวีซี เอ็มวีพี เอ็มวีวีเอ็ม และเอ็มวีไอ โดยใช้ข้อมูลจาก 40 โครงการ สามารถอภิปรายผลการทดลองได้ดังนี้

โครงการที่ใช้แบบรูปสถาปัตยกรรมแบบเอ็มวีไอมีค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดรวมทุกประเภทน้อยกว่าโครงการที่ใช้แบบรูปสถาปัตยกรรมอื่นทั้งหมด โดยที่น้อยกว่าแบบรูปเอ็มวีวีเอ็มน้อยที่สุด ตามด้วยแบบรูปเอ็มวีซีและแบบรูปเอ็มวีพี ถึงแม้ว่าในระหว่างแบบรูปเอ็มวีวีเอ็ม เอ็มวีซี และเอ็มวีพี ด้วยกันเองจะยังไม่สามารถสรุปได้อย่างมีนัยสำคัญว่า มีค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดรวมทุกประเภทที่แตกต่างกันหรือไม่จากข้อมูลที่ใช้ในการทดลอง แต่พอที่จะเห็นแนวโน้มของค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดรวมทุกประเภท ที่ปรากฏเมื่อใช้แบบรูปสถาปัตยกรรมต่างๆ เรียงค่าเฉลี่ยจากน้อยไปมาก ได้แก่ แบบรูปเอ็มวีไอ เอ็มวีวีเอ็ม เอ็มวีซี และเอ็มวีพี ตามลำดับ

เมื่อพิจารณาร่องรอยที่ไม่ดีในโค้ดแยกประเภท ค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดแต่ละประเภทในโครงการที่ใช้แบบรูปสถาปัตยกรรมแต่ละแบบไม่แตกต่างกัน มีเพียงกรณีของร่องรอยที่ไม่ดีประเภท Leaking Inner Class ที่พบว่า โครงการที่ใช้แบบรูปสถาปัตยกรรมแบบเอ็มวีไอมีค่าเฉลี่ยความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดประเภทนี้น้อยกว่าโครงการที่ใช้แบบรูปสถาปัตยกรรมแบบเอ็มวีพี ดังนั้นถึงแม้ว่าในภาพรวมการใช้แบบรูปสถาปัตยกรรมแต่ละแบบจะไม่แตกต่างกันนักในแง่การปรากฏของร่องรอยที่ไม่ดีในโค้ดแต่ละประเภท แต่มีสัญญาณที่บ่งบอกว่า การใช้แบบรูปสถาปัตยกรรมแบบเอ็มวีไอ อาจเป็นประโยชน์กับนักพัฒนาซอฟต์แวร์จากการที่สามารถลดความหนาแน่นของร่องรอยที่ไม่ดีได้บ้าง หากต้องมีการเลือกใช้แบบรูปสถาปัตยกรรม

อย่างไรก็ตาม ในการศึกษาวิวัฒนาการของโครงการ หากโครงการมีจำนวนคอมมิตเพิ่มขึ้นหรือมีวิวัฒนาการมากขึ้น ปริมาณความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดรวมทุกประเภทในแต่ละคอมมิตจะมีทั้งที่เพิ่มขึ้น ตามคอมมิตที่เพิ่มขึ้น

(หากโครงการใช้แบบรูปสถาปัตยกรรมแบบเอ็มวิซี) และลดลง ผกผันกับคอมมิตที่เพิ่มขึ้น (หากโครงการใช้แบบรูปสถาปัตยกรรมแบบเอ็มวีพี เอ็มวีวีเอ็ม และเอ็มวีไอ) แต่สหสัมพันธ์นี้อยู่ในระดับที่น้อยมากจนละเลยได้ (น้อยกว่า 0.3) (Hinkle, Wiersma, & Jurs 1988) ดังนั้นจากข้อมูลที่ใช้ทดลอง จึงสรุปได้ว่าปริมาณความหนาแน่นของร่องรอยที่ไม่ดีรวมทุกประเภทนั้นแทบจะไม่เปลี่ยนแปลงไปตามคอมมิตที่เกิดขึ้นในโครงการ ซึ่งอาจเป็นได้ว่า ในคอมมิตใหม่ที่เพิ่มขึ้น นักพัฒนาซอฟต์แวร์มีการแก้ไขเพิ่มเติมโค้ดหรือลดขนาดโค้ดไปพร้อมๆ กับการเพิ่มร่องรอยที่ไม่ดีเข้าไปในระหว่างการวิวัฒนาการ หรือมีการนำร่องรอยที่ไม่ดีที่พบในขณะพัฒนาออกไปจากโค้ดด้วย จึงทำให้สัดส่วนความหนาแน่นของร่องรอยที่ไม่ดีที่ปรากฏในโค้ดไม่เกิดการเปลี่ยนแปลงมากนัก สรุปผลที่ได้จากการทดลองข้างต้นสามารถเป็นประโยชน์ต่อนักพัฒนาซอฟต์แวร์ ในการใช้ประกอบการพิจารณาเลือกใช้แบบรูปสถาปัตยกรรมของแอนดรอยด์แอปพลิเคชัน

งานวิจัยนี้สามารถพัฒนาต่อยอดโดยทำการวิเคราะห์เพิ่มเติมได้ จากการที่ชุดเครื่องมือ SNIFFER สามารถทำการตรวจสอบร่องรอยที่ไม่ดีในโค้ดที่เกิดขึ้นใหม่ (Smell Introduction) ร่องรอยที่ไม่ดีในโค้ดที่ปรากฏอยู่ (Smell Presence) และร่องรอยที่ไม่ดีในโค้ดที่ถูกแก้ไขไป (Smell Refactoring) ของแต่ละคอมมิตได้ จึงสามารถนำข้อมูลเหล่านี้มาวิเคราะห์ความสัมพันธ์ของร่องรอยที่ไม่ดีในโค้ดและแบบรูปสถาปัตยกรรมของแอนดรอยด์แอปพลิเคชันเพิ่มเติมได้ รวมไปถึงการวิเคราะห์ลักษณะและพฤติกรรมอื่นของร่องรอยที่ไม่ดีในโค้ดของแอนดรอยด์แอปพลิเคชันได้

นอกจากนี้ ปัจจัยการเกิดร่องรอยที่ไม่ดีในโค้ดของแต่ละโครงการแอนดรอยด์อาจจะไม่ได้เกิดจากการใช้แบบรูปสถาปัตยกรรมแต่เพียงอย่างเดียว อีกทั้งการวิเคราะห์และเลือกใช้แบบรูปสถาปัตยกรรมก็ไม่ควรพิจารณาจากการปรากฏของร่องรอยที่ไม่ดีในโค้ดแต่เพียงอย่างเดียวเช่นกัน ควรต้องนำปัจจัยอื่นมาพิจารณาร่วมด้วย ตัวอย่างเช่น ความสามารถของนักพัฒนาซอฟต์แวร์ อาจเป็นปัจจัยสำคัญในการศึกษาความสัมพันธ์ที่มีต่อการเกิดร่องรอยที่ไม่ดีในโค้ดและการเลือกใช้แบบรูปต่างๆ ที่เหมาะสมกับการพัฒนา

เอกสารอ้างอิง

กัลยา วานิชย์บัญชา. (2542). *การวิเคราะห์สถิติ: สถิติเพื่อการตัดสินใจ*. กรุงเทพฯ : โรงพิมพ์แห่งจุฬาลงกรณ์มหาวิทยาลัย.

Brown, W. H., Malveau, R. C., McCormick, H. W. S., & Mowbray, T. J. (1998). *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*, John Wiley & Sons, Inc.

Brylski, M. (2013). *Android Smell Catalogue*. Retrieved 21 April 2023 from https://martinbrylski.github.io/android_smells/index.html

Decanini, E. *Battle of the Android Architectures: MVP vs MVVM vs MVI*. Retrieved 21 April 2023 from https://www.ericthecoder.com/2020/07/20/battle-of-the-android-architectures-mvp-vs-mvvm-vs-mvi/

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Programs*. Addison-Wesley Professional. Habchi, S. *Sniffer source code.* Retrieved 21 April 2023 from https://github.com/HabchiSarra/Sniffer/

Habchi, S., Rouvoy, R., & Moha, N. (2019). *On the Survival of Android Code Smells in the Wild*. 2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft) 25-26 May 2019.

Hecht, G., Rouvoy, R., Moha, N., & Duchien, L. (2015). *Detecting Antipatterns in Android Apps*. 2015 2nd ACM International Conference on Mobile Software Engineering and Systems, 16-17 May 2015.

Hinkle, D. E., Wiersma, W., & Jurs, S. G. (1988). *Applied statistics for the behavioral sciences* (2nd ed.). Houghton Mifflin.

Reimann, J., Brylski, M., & Assmann, U. (2014). A Tool-Supported Quality Smell Catalogue For Android Developers. *Softwaretechnik-Trends*, *34*.

Saelor, N. *MVC MVP MVVM คืออะไร และต่างกันอย่างไร*. Retrieved 21 April 2023 from https://medium.com/@leelorz6/mvc-mvp-mvvm-%E0%B8%84%E0%B8%B7%E0%B8%AD%E0%B8%AD%E0%B8%B0%E0%B9%84%E0%B8%A3-%E0%B9%81%E0%B8%A5%E0%B8%B0%E0%B8%95%E0%B9%88%E0%B8%B2%E0%B8%87%E0%B8%81%E0%B8%B1%E0%B8%99%E0%B8%AD%E0%B8%A2%E0%B9%88%E0%B8%B2%E0%B8%87%E0%B9%84%E0%B8%A3-ca16a19631dc

Scheffe, H. (1999). *The Analysis of Variance*. John Wiley & Sons.

Sobrinho, E.V.D.P., & Maia, M. D. A. (2021). *On the Interplay of Smells Large Class, Complex Class and Duplicate Code*. Brazilian Symposium on Software Engineering (SBES '21), September 27-October 1, 2021, 64-73. Source Making. *Swiss Army Knife*. Retrieved 21 April 2023 from https://sourcemaking.com/antipatterns/swiss-army-knife

Walter, B., & Alkhaeir, T. (2016). The Relationship between Design Patterns and Code Smells: An exploratory study. *Information and Software Technology*, *74*, 127-142. https://doi.org/https://doi.org/10.1016/j.infsof.2016.02.003.

Data Availability Statement: The raw data supporting the conclusions of this article will be made available by the authors, without undue reservation.

Conflicts of Interest: The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's Note: All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.