การวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์โดยอิงร่องรอยที่ไม่ดีในโค้ดในระบบควบคุมเวอร์ชันแบบกระจายศูนย์

นายณทัศน์ จงประสิทธิ์

สารนิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2561

Software Developer Performance Measurement Based on Code Smells in Distributed Version Control System

Mr. Natach Jongprasit

An Independent Study Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science in Software Engineering

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2018

หัวข้อสารนิพนธ์	การวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์โดยอิงร่องรอยที่ไม่ดีในโค้ดในระบบควบคุมเวอร์ชันแบบกระจายศูนย์
โดย	นายณทัศน์ จงประสิทธิ์
สาขาวิชา	วิศวกรรมซอฟต์แวร์
อาจารย์ที่ปรึกษาหลัก	รองศาสตราจารย์ ดร.ทวิตีย์ เสนีวงศ์ ณ อยุธยา

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับสารนิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

คณะกรรมการสอบสารนิพนธ์

..... ประธานกรรมการ
(รองศาสตราจารย์ ดร.พรศิริ หมื่นไชยศรี)

..... อาจารย์ที่ปรึกษาหลัก
(รองศาสตราจารย์ ดร.ทวิตีย์ เสนีวงศ์ ณ อยุธยา)

..... กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.วีระ เหมืองสิน)

ณทัศน์ จงประสิทธิ์ : การวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์โดยอิงร่องรอยที่ไม่ดีในโค้ดในระบบควบคุมเวอร์ชันแบบกระจายศูนย์. (Software Developer Performance Measurement Based on Code Smells in Distributed Version Control System) อ.ที่ปรึกษาหลัก : รศ. ดร.ทวิตีย์ เสนีวงศ์ ณ อยุธยา

การจัดทีมนักพัฒนาซอฟต์แวร์ให้เหมาะสมกับโครงการซอฟต์แวร์นั้นมีความสำคัญต่อการพัฒนาโปรแกรมภายใต้โครงการ ซึ่งประสิทธิภาพของนักพัฒนาซอฟต์แวร์นั้นสามารถวัดได้จากคุณภาพของซอฟต์แวร์ที่พัฒนาขึ้นซึ่งร่องรอยที่ไม่ดีในโค้ดเป็นปัจจัยหนึ่งที่ส่งผลถึงคุณภาพของซอฟต์แวร์ อย่างไรก็ตามในการพัฒนาซอฟต์แวร์ปัจจุบันมีการใช้ระบบควบคุมเวอร์ชันแบบกระจายศูนย์ ซึ่งนักพัฒนาซอฟต์แวร์ในโครงการเดียวกันจะร่วมกันพัฒนาซอฟต์แวร์ที่อยู่บนระบบ ดังนั้นคุณภาพของซอฟต์แวร์และปริมาณร่องรอยที่ไม่ดีในโค้ดนี้จึงเป็นผลรวมจากประสิทธิภาพในการพัฒนาซอฟต์แวร์ของทั้งทีม ทำให้การพิจารณาประสิทธิภาพของนักพัฒนาซอฟต์แวร์รายคนทำได้ยาก โครงงานมหาบัณฑิตจึงจัดทำขึ้นเพื่อนำเสนอวิธีการและเครื่องมือสนับสนุนการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์รายคนที่ร่วมอยู่ในโครงการที่มีการพัฒนาซอฟต์แวร์บนระบบควบคุมเวอร์ชันแบบกระจายศูนย์โดยอิงร่องรอยที่ไม่ดีในโค้ด โดยจะนำเครื่องมือดีไซน์ไนต์มาตรวจหาจำนวนร่องรอยที่ไม่ดีในโค้ดแต่ละเวอร์ชันที่นักพัฒนาซอฟต์แวร์แต่ละรายในทีมคอมมิตเข้าสู่ระบบสำหรับแต่ละโครงการ หลังจากนั้นจะนำผลลัพธ์จากการตรวจหาจำนวนร่องรอยที่ไม่ดีในโค้ดมาคำนวณค่าประสิทธิภาพของนักพัฒนาซอฟต์แวร์แต่ละราย โดยใช้ค่าเฉลี่ยเบย์เซียนและนำมาประเมินความสอดคล้องในการจัดอันดับประสิทธิภาพของนักพัฒนาซอฟต์แวร์ โดยการเปรียบเทียบกับการจัดอันดับโดยนักพัฒนาซอฟต์แวร์ที่มีประสบการณ์โดยใช้วิธีการวัดค่าสัมประสิทธิ์สหสัมพันธ์อันดับที่ของสเปียร์แมน ผลการทดลองพบว่ามีความสอดคล้องกันในเชิงบวก ดังนั้นวิธีการที่เสนอจึงสามารถช่วยสนับสนุนให้กับผู้จัดการโครงการในการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์แต่ละรายร่วมกับการใช้วิธีอื่น ๆ

สาขาวิชา วิศวกรรมซอฟต์แวร์ ลายมือชื่อนิสิต .....

ปีการศึกษา 2561 ลายมือชื่อ อ.ที่ปรึกษาหลัก .....

5970921721 : MAJOR SOFTWARE ENGINEERING

KEYWORD: Software Developer Performance, Code Smell, Distributed Version Control System, Bayesian Average

Natach Jongprasit : Software Developer Performance Measurement Based on Code Smells in Distributed Version Control System. Advisor: Assoc. Prof. TWITTIE SENIVONGSE

Effectively staffing a software development team for a software project is important to the development of software under the project. Performance of software developers can be measured by the quality of the produced software and the number of code smells is one factor that indicates software quality. However, modern software development uses distributed version control systems in which different software developers collaborate to develop the software on the systems. The quality and number of code smells in the software are hence the result of the aggregate performance of the whole team. This makes it difficult to measure the performance of individual developers. This master project proposes a method and a supporting tool for measuring the performance of individual software developers under a project in a distributed version control system based on code smells. A tool called Designite is used to detect code smells in each version of the code that is committed to the project by each developer. Then the number of code smells is used to calculate the performance of individual developers using Bayesian Average. Spearman's Rank Correlation Coefficients are calculated between ranking of software developers by the proposed tool and ranking by the evaluators who have experiences in software development. The result shows that there is positive correlation. Hence the proposed method can be used with other means to support project managers in measuring the performance of software developers

Field of Study: Software Engineering Student's Signature .....

Academic Year: 2018 Advisor's Signature .....

กิตติกรรมประกาศ

ขอกราบขอบพระคุณท่าน รศ. ดร.ทวิตีย์ เสนีวงศ์ ณ อยุธยา อาจารย์ที่ปรึกษาโครงงานมหาบัณฑิตของข้าพเจ้า เป็นอย่างยิ่งที่ท่านได้เสียสละเวลาอันมีค่าให้คำปรึกษาแนะนำทางด้านการศึกษาหาความรู้ คุณธรรม จริยธรรมและแนวทางสำหรับการทำโครงงานมหาบัณฑิตนี้ ตลอดจนคอยดูแลและประสานงานให้ความช่วยเหลือแก่นิสิตที่ทำโครงงานมหาบัณฑิตทุกคน

ขอขอบพระคุณ เพื่อนๆ พี่ๆ ที่บริษัท แอดวานซ์ อินโฟร์ เซอร์วิส จำกัด (มหาชน) ทุกท่านที่ให้คำปรึกษาแนะนำ รวมทั้งทำแบบสอบถามการประเมินประสิทธิภาพของนักพัฒนาซอฟต์แวร์ทั้ง 2 โครงการ เพื่อนำไปประเมินผลความสอดคล้องในการจัดอันดับประสิทธิภาพของนักพัฒนาซอฟต์แวร์ที่ได้จากเครื่องมือ

ขอขอบคุณ เพื่อน ๆ หลักสูตรวิศวกรรมซอฟต์แวร์ สำหรับกำลังใจและคำปรึกษาแนะนำในการจัดทำโครงงานมหาบัณฑิต

สุดท้ายนี้ ขอกราบขอบพระคุณ บิดา มารดา และสมาชิกในครอบครัวทุกท่านที่คอยให้กำลังใจและให้การช่วยเหลือสนับสนุนมาโดยตลอด

ณทัศน์ จงประสิทธิ์

สารบัญ

หน้า

สารบัญตาราง

หน้า

สารบัญรูปภาพ

หน้า

บทที่ 1
บทนำ

1.1. ความเป็นมาและความสำคัญของปัญหา

การจัดทีมนักพัฒนาซอฟต์แวร์ให้เหมาะสมกับโครงการซอฟต์แวร์นั้นมีความสำคัญต่อการพัฒนาโปรแกรมภายใต้โครงการ และยังถือว่าเป็นการจัดสรรบุคลากรได้อย่างมีประสิทธิภาพ วิธีการจัดทีมนักพัฒนาซอฟต์แวร์ให้เหมาะสมนั้นสามารถพิจารณาจากประสิทธิภาพของนักพัฒนาซอฟต์แวร์ แต่ในปัจจุบันไม่ได้มีการกำหนดวิธีการวัดประสิทธิภาพอย่างชัดเจน และในหลาย ๆ หน่วยงานนั้นมีวิธีการจัดทีมนักพัฒนาซอฟต์แวร์อยู่หลากหลายวิธี เช่น ตัดสินใจจากประสบการณ์การพัฒนา การทำการทดสอบ หรือ การปรึกษากันภายในทีมพัฒนาซอฟต์แวร์ว่านักพัฒนาซอฟต์แวร์รายใดมีความสามารถหรือเหมาะสมกับโครงการมากน้อยกว่ากัน โดยอาจจะไม่ได้มีผลการวัดอย่างแน่ชัด จึงอาจทำให้ทางโครงการไม่สามารถจัดทีมนักพัฒนาซอฟต์แวร์ที่มีประสิทธิภาพตามที่ต้องการมาร่วมในโครงการได้อย่างเหมาะสมนัก และอาจทำให้ประสิทธิภาพของโครงการมีระดับต่ำกว่าที่ต้องการ

ประสิทธิภาพของนักพัฒนาซอฟต์แวร์นั้นสามารถวัดได้จากคุณภาพของซอฟต์แวร์ที่พัฒนาขึ้นซึ่งร่องรอยที่ไม่ดีในโค้ด (Code Smells) [1] [2] เป็นปัจจัยหนึ่งที่ส่งผลถึงคุณภาพของซอฟต์แวร์ โดยร่องรอยที่ไม่ดีในโค้ดคือโค้ดที่มีการพัฒนาขึ้นมาและมีแนวโน้มที่จะก่อปัญหาหรือข้อผิดพลาด ซึ่งซอฟต์แวร์ที่มีร่องรอยที่ไม่ดีปริมาณมากจะส่งผลต่อการบำรุงรักษา และสะท้อนถึงประสิทธิภาพของนักพัฒนาซอฟต์แวร์ในด้านการพัฒนาโค้ดที่มีคุณภาพ อย่างไรก็ตามในการพัฒนาซอฟต์แวร์ปัจจุบันมีการใช้ระบบควบคุมเวอร์ชันแบบกระจายศูนย์ (Distributed Version Control System) [3] ซึ่งนักพัฒนาซอฟต์แวร์ในโครงการเดียวกันจะร่วมกันพัฒนาซอฟต์แวร์ที่อยู่บนระบบ ดังนั้นคุณภาพของซอฟต์แวร์และปริมาณร่องรอยที่ไม่ดีในโค้ดนี้จึงเป็นผลโดยรวมจากประสิทธิภาพในการพัฒนาซอฟต์แวร์ของทั้งทีม ทำให้การพิจารณาประสิทธิภาพของนักพัฒนาซอฟต์แวร์รายคนทำได้ยาก

งานวิจัยนี้จึงจัดทำขึ้นเพื่อนำเสนอวิธีการและเครื่องมือสนับสนุนการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์รายคนที่ร่วมอยู่ในโครงการที่มีการพัฒนาซอฟต์แวร์บนระบบควบคุมเวอร์ชันแบบกระจายศูนย์โดยอิงร่องรอยที่ไม่ดีในโค้ด ผู้วิจัยจะนำเครื่องมือที่ชื่อว่า Designite [4] มาทำการตรวจหาจำนวนร่องรอยที่ไม่ดีในโค้ดในแต่ละเวอร์ชันที่นักพัฒนาซอฟต์แวร์แต่ละรายในทีมคอมมิต (Commit) เข้าสู่ระบบสำหรับแต่ละโครงการ โดยงานวิจัยนี้จะนำร่องรอยที่ไม่ดีในโค้ด ประเภทร่องรอยที่ไม่ดีด้านการออกแบบ (Design Smells) [5] และร่องรอยที่ไม่ดีด้านการพัฒนา (Implementation Smells) [6] มาเป็นตัววัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์ เนื่องจากร่องรอยที่ไม่ดีทั้งสองประเภทนี้มีแนวโน้มที่จะเกิดสูงที่สุด หลังจากนั้นจะนำผลลัพธ์จากการตรวจหา

จำนวนร่องรอยที่ไม่ดีในโค้ดมาคำนวณค่าประสิทธิภาพของนักพัฒนาซอฟต์แวร์แต่ละราย โดยใช้ค่าเฉลี่ยเบย์เซียน (Bayesian Average) [7] [8] ซึ่งวิธีนี้จะทำให้ค่าประสิทธิภาพที่ได้มีความน่าเชื่อถือเพิ่มมากขึ้นหากจำนวนเวอร์ชันของซอฟต์แวร์ที่นักพัฒนาซอฟต์แวร์รายนี้คอมมิตเข้าสู่ระบบมีจำนวนมากขึ้น ซึ่งเป็นการสะท้อนถึงปริมาณการมีส่วนร่วมของนักพัฒนาซอฟต์แวร์รายนี้ต่อการพัฒนาซอฟต์แวร์ของโครงการ การวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์โดยวิธีที่เสนอจะช่วยสนับสนุนการตัดสินใจในการจัดทีมนักพัฒนาซอฟต์แวร์ตามความต้องการที่ได้รับมอบหมายในโครงการ และยังเป็นข้อมูลสนับสนุนสำหรับการพัฒนาศักยภาพของนักพัฒนาซอฟต์แวร์ในโครงการ โดยที่วิธีที่เสนอนี้สามารถนำไปใช้ร่วมกับวิธีวัดประสิทธิภาพนักพัฒนาซอฟต์แวร์วิธีอื่นที่หน่วยงานใช้อยู่ได้

1.2. วัตถุประสงค์ของโครงงาน

1. เพื่อนำเสนอวิธีการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์รายคนที่ร่วมอยู่ในโครงการที่มีการพัฒนาโดยใช้ระบบควบคุมเวอร์ชันแบบกระจายศูนย์
2. สร้างเครื่องมือสนับสนุนวิธีการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์รายคนที่ร่วมอยู่ในโครงการที่มีการพัฒนาโดยใช้ระบบควบคุมเวอร์ชันแบบกระจายศูนย์

1.3. ขอบเขตของโครงงาน

1. พิจารณาร่องรอยที่ไม่ดีในด้านการออกแบบและด้านการพัฒนาที่วัดได้จากเครื่องมือ Designite เท่านั้น
2. วัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์ที่พัฒนาโดยใช้ภาษาซีชาร์ปเท่านั้นโดยใช้ค่าเฉลี่ยเบย์เซียน
3. ใช้ระบบควบคุมเวอร์ชัน GitHub เท่านั้น
4. พัฒนาเครื่องมือด้วยภาษาซีชาร์ป
5. แสดงคะแนนของการวัดประสิทธิภาพรายคนภายในโครงการ
6. ประเมินผลความสอดคล้องในการจัดอันดับประสิทธิภาพของนักพัฒนาซอฟต์แวร์กับผลการประเมินจากนักพัฒนาซอฟต์แวร์ที่มีประสบการณ์ 4 คน โดยจะทำการประเมินทั้งหมด 2 โครงการ

1.4. ขั้นตอนการดำเนินงาน

1. ศึกษาองค์ความรู้และทฤษฎีที่เกี่ยวข้องกับงานวิจัย
2. ศึกษาเครื่องมือที่จะนำมาใช้วิเคราะห์
3. กำหนดสภาพแวดล้อมที่จะใช้ในการวิเคราะห์
4. กำหนดประเภทของร่องรอยที่ไม่ดีในโค้ดที่จะนำมาวิเคราะห์
5. ออกแบบและกำหนดสมการสำหรับการวัดประสิทธิภาพ
6. ออกแบบและพัฒนาเครื่องมือ
7. ทดสอบและประเมินผล
8. สรุปผลการวิจัย
9. จัดทำบทความวิชาการและเล่มโครงงาน

1.5. ประโยชน์ที่คาดว่าจะได้รับ

1. ได้วิธีการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์รายคนที่ร่วมอยู่ในโครงการที่มีการพัฒนาโดยใช้ระบบควบคุมเวอร์ชันแบบกระจายศูนย์เพื่อสนับสนุนการตัดสินใจในการจัดทีมนักพัฒนาซอฟต์แวร์ตามความต้องการที่ได้รับมอบหมายในโครงการ
2. ได้เครื่องมือสำหรับวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์รายคนที่ร่วมอยู่ในโครงการที่มีการพัฒนาโดยใช้ระบบควบคุมเวอร์ชันแบบกระจายศูนย์

1.6. โครงสร้างของเนื้อหาโครงงาน

โครงสร้างของเนื้อหารายงานโครงงานมหาบัณฑิตประกอบด้วยรายละเอียด 6 บท และภาคผนวก ดังต่อไปนี้

บทที่ 1 กล่าวถึงความเป็นมาและความสำคัญของปัญหา วัตถุประสงค์ของโครงงาน ขอบเขตของโครงงาน ขั้นตอนและวิธีดำเนินโครงงาน และประโยชน์ที่คาดว่าจะได้รับจากโครงงานมหาบัณฑิต

บทที่ 2 กล่าวถึงทฤษฎี งานวิจัย และเครื่องมือที่เกี่ยวข้อง

บทที่ 3 กล่าวถึงแนวคิดและวิธีวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์

บทที่ 4 กล่าวถึงการออกแบบและการพัฒนาเครื่องมือ

บทที่ 5 กล่าวถึงการทดสอบและประเมินผล

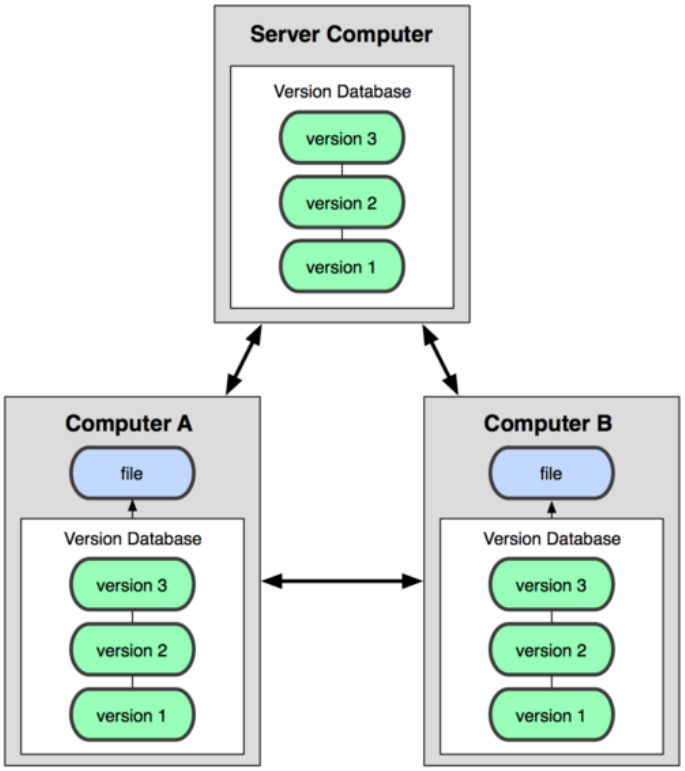
บทที่ 6 กล่าวถึงบทสรุปของโครงงานมหาบัณฑิต และข้อเสนอแนะ

บทที่ 2
เอกสารและงานวิจัยที่เกี่ยวข้อง

2.1 แนวคิดและทฤษฎี

2.1.1. ระบบควบคุมเวอร์ชันแบบกระจายศูนย์

ระบบควบคุมเวอร์ชันแบบกระจายศูนย์ (Distributed Version Control System) [3] เป็นระบบที่จัดเก็บการเปลี่ยนแปลงที่เกิดขึ้นกับไฟล์หนึ่งหรือหลายไฟล์ของซอฟต์แวร์ เพื่อที่จะสามารถเรียกเวอร์ชันใดเวอร์ชันหนึ่งของซอฟต์แวร์กลับมาใช้งานเมื่อใดก็ได้ และสามารถกระจายไฟล์ของซอฟต์แวร์ไปสำเนาไว้บนเครื่องคอมพิวเตอร์ของนักพัฒนาซอฟต์แวร์แต่ละรายได้ นอกจากนั้นยังจะช่วยในการเปรียบเทียบและแก้ไขสิ่งที่เกิดขึ้นในอดีต ดูว่าใครเป็นคนแก้ไขคนสุดท้ายที่อาจทำให้เกิดปัญหา แก้ไขเมื่อใด และยังสามารถกู้คืนไฟล์ที่ถูกลบหรือทำให้เสียหายโดยไม่ได้ตั้งใจได้อย่างง่ายดาย ในการควบคุมเวอร์ชันแบบกระจายศูนย์จะมีการคัดลอกประวัติชิ้นงานทั้งหมดมาไว้ที่เครื่องของผู้ใช้ดังรูปที่ 2.1 เพื่อให้ผู้ใช้สามารถเข้าถึงประวัติงานได้ตลอดเวลา และยังลดจำนวนครั้งที่ต้องการติดต่อกับเซิร์ฟเวอร์ลง เหลือเพียงเมื่อต้องการส่งงานที่ปรับปรุงแก้ไขไปให้เพื่อนร่วมงานเท่านั้นซึ่งการทำงานแบบนี้ทำให้ในกรณีที่เซิร์ฟเวอร์หลักมีปัญหา เกิดความเสียหาย จะไม่ทำให้งานของผู้ใช้เสียหายไปด้วย และยังสามารถคัดลอกงานที่อยู่ในเครื่องของผู้ใช้ขึ้นไปบนเซิร์ฟเวอร์เพื่อกู้ข้อมูลหลังจากที่เซิร์ฟเวอร์ได้รับการซ่อมแซม ในปัจจุบันมีผู้ให้บริการระบบควบคุมเวอร์ชันแบบกระจายศูนย์หลายราย เช่น GitHub, Team Foundation Server, Bitbucket ในงานวิจัยนี้จะพิจารณาเฉพาะ GitHub เนื่องจากเป็นที่นิยมอย่างแพร่หลายในหมู่นักพัฒนาซอฟต์แวร์



รูปที่ 2.1 การควบคุมเวอร์ชันแบบกระจายศูนย์ [3]

2.1.2. ร่องรอยที่ไม่ดีในโค้ด

ร่องรอยที่ไม่ดีในโค้ด (Code Smell) [1] [2] คือ ลักษณะของโค้ดที่ทำการพัฒนาขึ้นมาแล้วมีแนวโน้มก่อปัญหาหรือข้อผิดพลาดที่อาจจะเกิดขึ้นได้ ซึ่งทำให้เวลาการพัฒนาล่าช้าขึ้นไปอีก และสามารถเพิ่มความเสี่ยงที่จะเกิดปัญหาต่าง ๆ ในอนาคตได้ เช่น โค้ดในส่วนนั้นมีแนวโน้มที่จะเอื้อต่อการเกิดจุดบกพร่อง (Bug) ในอนาคตเป็นต้น อย่างไรก็ตามการมีรองร่อยที่ไม่ดีในโค้ดอาจจะไม่ก่อให้เกิดความผิดพลาดเลยก็ได้ แต่สามารถใช้วัดคุณภาพของการออกแบบและคุณภาพของโค้ดว่าอยู่ในระดับที่ไม่ดีนัก ซึ่งหมายถึงโค้ดที่ยากต่อการแก้ไขและนำมาใช้ใหม่

ในปัจจุบันมีเครื่องมือต่าง ๆ ที่ใช้วัดร่องรอยที่ไม่ดีในโค้ด เช่น NDepend, SonarQube และDesignite ซึ่งในงานวิจัยนี้จะใช้เครื่องมือที่มีชื่อว่า Designite [4] ที่สามารถบอกประเภทของร่องรอยที่ไม่ดีในโค้ดและทำการจัดกลุ่มร่องรอยที่ไม่ดีในโค้ดได้อีกด้วย ร่องรอยที่ไม่ดีในโค้ดที่วัดได้โดย Designite สามารถแบ่งได้เป็น 3 กลุ่ม ได้แก่ ร่องรอยที่ไม่ดีในด้านสถาปัตยกรรม (Architecture Smells [9], ร่องรอยที่ไม่ดีในด้านการออกแบบ (Design Smells) [5] และร่องรอยที่ไม่ดีในด้านการพัฒนา (Implementation Smells) [6]

ในงานวิจัยนี้ ผู้วิจัยพิจารณาเฉพาะร่องรอยที่ไม่ดีในด้านการพัฒนาและการออกแบบโดยรายละเอียดของร่องรอยที่ไม่ดีเหล่านี้แสดงดังตารางที่ 2.1 และตารางที่ 2.2

ตารางที่ 2.1 รายละเอียดของร่องรอยที่ไม่ดีในด้านการพัฒนา [10]

Implementation Smell	**Brief Description**
Complex Conditional	a complex conditional statement
Complex Method	a method with high cyclomatic complexity
Duplicate Code	a code clone within a method
Empty Catch Block	a catch block of an exception is empty
Long Identifier	an identifier with excessive length
Long Method	a method is excessively long
Long Parameter List	a method has long parameter list
Long Statement	an excessively long statement
Magic Number	an unexplained number is used in an expression
Missing Default	a switch statement does not contain a default case
Virtual Method Call from Constructor	a constructor calls a virtual method

ตารางที่ 2.2 รายละเอียดของร่องรอยที่ไม่ดีในด้านการออกแบบ [10]

Design Smell	**Brief Description**
Broken Hierarchy	a supertype and its subtype conceptually do not share an “IS-A” relationship
Broken Modularization	data and/ or methods that ideally should have been localized into a single abstraction are separated and spread across multiple abstractions
Cyclically-dependent Modularization	two or more abstractions depend on each other directly or indirectly

ตารางที่ 2.2 รายละเอียดของร่องรอยที่ไม่ดีในด้านการออกแบบ [10] (ต่อ)

Design Smell	Brief Description
Cyclic Hierarchy	a supertype in a hierarchy depends on any of its subtypes
Deep Hierarchy	an inheritance hierarchy is “excessively” deep
Deficient Encapsulation	the declared accessibility of one or more members of an abstraction is more permissive than actually required
Duplicate Abstraction	two or more abstractions have identical names or identical implementation
Hub-like Modularization	an abstraction has high incoming and outgoing dependencies
Imperative Abstraction	an operation is turned into a class
Insufficient Modularization	an abstraction exists that has not been completely decomposed, and a further decomposition could reduce its size, or implementation complexity
Missing Hierarchy	conditional logic to explicitly manage variation in behaviour
Multifaceted Abstraction	an abstraction has more than one responsibility assigned to it
Multipath Hierarchy	a subtype inherits both directly as well as indirectly from a supertype
Rebellious Hierarchy	a subtype rejects the methods provided by its supertype(s)
Unexploited Encapsulation	client code uses explicit type checks
Unfactored Hierarchy	there is unnecessary duplication among types in a hierarchy
Unnecessary Abstraction	an abstraction that is actually not needed
Unutilized Abstraction	an abstraction is left unused
Wide Hierarchy	an inheritance hierarchy is “too” wide

2.1.3. ค่าเฉลี่ยเบย์เซียน

ค่าเฉลี่ยเบย์เซียน (Bayesian Average) [7] [8] เป็นค่าเฉลี่ยแบบหนึ่งที่นิยมใช้ในเว็บไซต์ที่มีการจัดอันดับสิ่งต่าง ๆ จากการให้คะแนนของผู้ใช้เช่น BoardGameGeek และ IMDB เป็นต้น ค่าเฉลี่ยเบย์เซียนสามารถแก้ปัญหาที่เกิดขึ้นจากการคิดคะแนนเฉลี่ยตามปกติหรือค่าเฉลี่ยเลขคณิต (Arithmetic Mean) ของสิ่งที่จะจัดอันดับซึ่งการหาค่าเฉลี่ยเลขคณิตจะมีหลักการง่าย ๆ คือการนำคะแนนทั้งหมดมารวมกันแล้วหารด้วยจำนวนครั้งที่ให้คะแนน จากนั้นจึงทำการจัดอันดับโดยเรียงจากค่าเฉลี่ยที่หาได้ แต่วิธีการดังกล่าวจะมีข้อเสียตามตัวอย่างดังนี้

สมมติว่าสิ่งที่จะจัดอันดับหรือไอเทม P1 มีคะแนนเฉลี่ย 0.8 จากการให้คะแนน 100 ครั้ง (ผู้ใช้ 80 คน ให้คะแนน 1 และผู้ใช้ 20 คน ให้คะแนน 0) แล้วมีไอเทมใหม่ P2 ซึ่งมีคะแนนเฉลี่ย 1 จากการให้คะแนน 1 ครั้ง (มีผู้ใช้เพียง 1 คน และให้คะแนน 1) ในสถานการณ์นี้จะทำให้ไอเทม P2 ขึ้นไปอยู่อันดับที่หนึ่งทันที จึงทำให้การคำนวณค่าเฉลี่ยเลขคณิตในรูปแบบปกติมีความน่าเชื่อถือน้อย ส่วนค่าเฉลี่ยเบย์เซียนนั้นจะเชื่อถือคะแนนของไอเทมที่มีจำนวนครั้งในการให้คะแนนน้อย น้อยกว่าคะแนนของไอเทมที่มีจำนวนครั้งในการให้คะแนนมาก ซึ่งหมายความว่ายิ่งมีจำนวนครั้งในการให้คะแนนมากเท่าใด น้ำหนักของคะแนนเสียงเหล่านี้จะมากยิ่งขึ้นเท่านั้น ซึ่งการคำนวณค่าเฉลี่ยเบย์เซียนนั้นได้แก้ไขข้อเสียของการหาค่าเฉลี่ยเลขคณิตตามปกติดังที่กล่าวมาโดยการใช้จำนวนครั้งในการให้คะแนนมาพิจารณาด้วย ดังนี้

1) ถ้าไอเทมนั้นยิ่งมีจำนวนครั้งในการให้คะแนนมากเท่าใด ค่าเฉลี่ยที่คำนวณได้จะใกล้เคียงกับค่าเฉลี่ยที่ไม่ได้มีการปรับค่าด้วยน้ำหนักคะแนนเสียง (Uncorrected Rating Value)

2) ถ้าไอเทมนั้นมีจำนวนครั้งในการให้คะแนนที่น้อย ค่าเฉลี่ยที่คำนวณได้ควรจะใกล้กับค่าเฉลี่ยของไอเทมทั้งหมด

ดังนั้นเมื่อมีการให้คะแนนใหม่ ค่าเฉลี่ยที่ได้จากการคำนวณจะถูกทำให้ห่างจากค่าเฉลี่ยของไอเทมทั้งหมด และเข้าสู่ค่าเฉลี่ยที่ไม่ได้มีการปรับค่าด้วยน้ำหนักคะแนนเสียง ค่าเฉลี่ยเบย์เซียนมีสมการดังนี้

$$b(r) = [\ W(a) * a + W(r) * r\ ] / (W(a) + W(r)] \qquad (1)$$

โดยที่ b(r) คือ ค่าเฉลี่ยเบย์เซียนของไอเทมนั้นๆ (คะแนนเฉลี่ยของไอเทมนั้น ๆ เพื่อการจัดอันดับ คำนวณเมื่อมีการให้คะแนนใหม่แก่ไอเทมนั้น ๆ)

a คือ ค่าเฉลี่ยเลขคณิตของคะแนนทั้งหมดทุกไอเทม

W(a) คือ ค่าเฉลี่ยเลขคณิตของจำนวนครั้งในการให้คะแนนทั้งหมดทุกไอเทม

r คือ ค่าเฉลี่ยเลขคณิตของคะแนนในไอเทมนั้น ๆ

W(r) คือ จำนวนครั้งในการให้คะแนนในไอเทมนั้น ๆ

2.2 เอกสารและงานวิจัยที่เกี่ยวข้อง

2.2.1. House of Cards: Code Smells in Open-source C# Repositories

งานวิจัยของ Tusher Sharma และคณะ [10] ได้ทำการวิเคราะห์ร่องรอยที่ไม่ดีในแต่ละชนิด เพื่อหาว่าร่องรอยที่ไม่ดีแบบใดมีมากที่สุดในคลังของซอฟต์แวร์โอเพนซอร์ซภาษาซีชาร์ป โดยใช้โปรแกรม Designite [4] ซึ่งโปรแกรมมีการแบ่งร่องรอยที่ไม่ดีในโค้ดเป็น 3 ประเภท ได้แก่ ร่องรอยที่ไม่ดีในด้านสถาปัตยกรรม (Architecture Smells) ร่องรอยที่ไม่ดีในด้านการออกแบบ (Design Smells) และร่องรอยที่ไม่ดีในด้านการพัฒนา (Implementation Smells) โดยโค้ดที่ใช้ในการวัดมีจำนวนมากกว่า 49 ล้านบรรทัด จากผลลัพธ์ที่แสดงออกมาดังรูปที่ 2.4 บ่งบอกว่าร่องรอยที่ไม่ดีแบบ Magic Number ซึ่งอยู่ในประเภทร่องรอยที่ไม่ดีในด้านการพัฒนามีจำนวนมากที่สุด คือจำนวน 2,993,353 จุด หากคิดเป็นความหนาแน่น (Density) ของร่องรอยที่ไม่ดีประเภทนี้จะได้ค่าสูงถึง 55.8 จุดต่อหนึ่งพันบรรทัด ส่วนร่องรอยที่ไม่ดีในด้านการออกแบบ พบว่ามีจำนวนมากเช่นกัน แต่น้อยกว่าร่องรอยที่ไม่ดีในด้านการพัฒนา และพบในรูปแบบที่หลากหลาย

TABLE II
DESCRIPTION OF DETECTED DESIGN SMELLS AND THEIR DISTRIBUTION

Acronym	Design smell	Brief description	#Instances	Percentage
DBH	Broken Hierarchy	a supertype and its subtype conceptually do not share an "IS-A" relationship	20,332	4.8%
DBM	Broken Modularization	data and/or methods that ideally should have been localized into a single abstraction are separated and spread across multiple abstractions	15,624	3.7%
DCM	Cyclically-dependent Modularization	two or more abstractions depend on each other directly or indirectly	52,436	12.5%
DCH	Cyclic Hierarchy	a supertype in a hierarchy depends on any of its subtypes	4,342	1.0%
DDH	Deep Hierarchy	an inheritance hierarchy is "excessively" deep	179	0.04%
DDE	Deficient Encapsulation	the declared accessibility of one or more members of an abstraction is more permissive than actually required	30,214	7.2%
DDA	Duplicate Abstraction	two or more abstractions have identical names or identical implementation	73,992	17.6%
DHM	Hub-like Modularization	an abstraction has high incoming and outgoing dependencies	676	0.2%
DIA	Imperative Abstraction	an operation is turned into a class	11,790	2.8%
DIM	Insufficient Modularization	an abstraction exists that has not been completely decomposed, and a further decomposition could reduce its size, or implementation complexity	26,429	6.3%
DMH	Missing Hierarchy	conditional logic to explicitly manage variation in behaviour	2,598	0.6%
DMA	Multifaceted Abstraction	an abstraction has more than one responsibility assigned to it	1,236	0.3%
DMH	Multipath Hierarchy	a subtype inherits both directly as well as indirectly from a supertype	1,454	0.3%
DRH	Rebellious Hierarchy	a subtype rejects the methods provided by its supertype(s)	11,794	2.8%
DUE	Unexploited Encapsulation	client code uses explicit type checks	6,964	1.6%
DUH	Unfactored Hierarchy	there is unnecessary duplication among types in a hierarchy	20,962	5.0%
DUA	Unnecessary Abstraction	an abstraction that is actually not needed	44,583	10.6%
DTA	Unutilized Abstraction	an abstraction is left unused	90,786	21.6%
DWH	Wide Hierarchy	an inheritance hierarchy is "too" wide	3,140	0.7%

TABLE III
DESCRIPTION OF DETECTED IMPLEMENTATION SMELLS AND THEIR DISTRIBUTION

Acronym	Implementation smell	Brief description	#Instances	Percentage
ICC	Complex Conditional	a complex conditional statement	21,643	0.6%
ICM	Complex Method	a method with high cyclomatic complexity	95,244	2.5%
IDC	Duplicate Code	a code clone within a method	17,921	0.5%
IEC	Empty Catch Block	a catch block of an exception is empty	14,560	0.4%
ILI	Long Identifier	an identifier with excessive length	7,741	0.2%
ILM	Long Method	a method is excessively long	17,521	0.5%
ILP	Long Parameter List	a method has long parameter list	79,899	2.1%
ILS	Long Statement	an excessive long statement	462,491	12.4%
IMN	Magic Number	an unexplained number is used in an expression	2,993,353	80.0%
IMD	Missing Default	a switch statement does not contain a default case	23,497	0.6%
IVC	Virtual Method Call from Constructor	a constructor calls a virtual method	4,545	0.1%

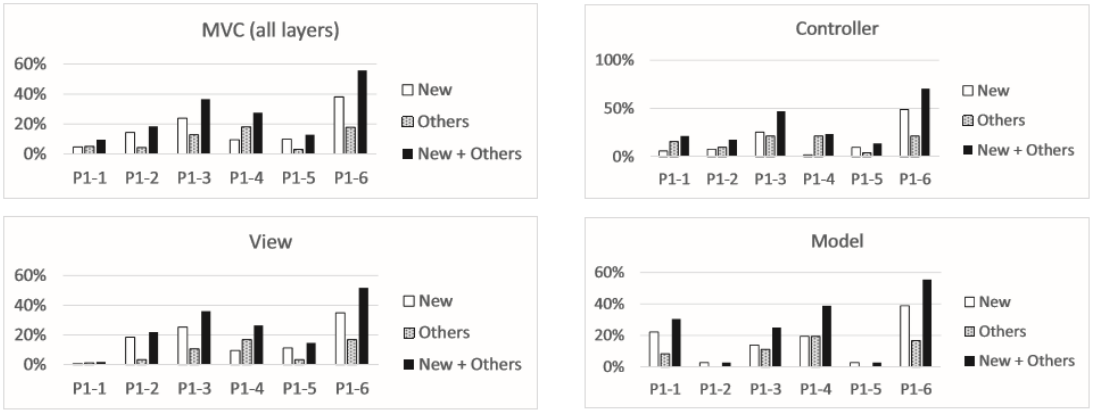
รูปที่ 2.2 ตารางการวัดร่องรอยที่ไม่ดีในโค้ด [10]

ผู้วิจัยได้นำเครื่องมือที่ใช้ในงานวิจัยเรื่องนี้มาใช้ตรวจหาร่องรอยที่ไม่ดีในโค้ด และทำการเลือกร่องรอยที่ไม่ดีในโค้ดด้านการออกแบบและร่องรอยที่ไม่ดีในโค้ดด้านการพัฒนามาใช้เป็น

ตัววัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์ เนื่องจากร่องรอยที่ไม่ดีทั้งสองประเภทนี้มีแนวโน้มที่จะเกิดสูงที่สุด

2.2.2. Analysis of Software Developer Activity on a Distributed Version Control System

งานวิจัยของ Shu Li และคณะ [11] ได้ทำการวิเคราะห์ซอฟต์แวร์ที่มีอยู่ระบบควบคุมเวอร์ชันแบบกระจายศูนย์สำหรับการพัฒนาซอฟต์แวร์ GitHub โดยวิเคราะห์ซอฟต์แวร์ที่มีการพัฒนาในรูปแบบ MVC Framework เพื่อวิเคราะห์ผู้พัฒนาแต่ละคนว่ามีทักษะทางด้าน View, Model หรือ Controller มากที่สุด และสามารถแยกแยะลักษณะของนักพัฒนาซอฟต์แวร์ว่ามีคุณลักษณะในด้านการพัฒนาขั้นสูง การมีส่วนร่วม การมีความคิดริเริ่ม การสนับสนุน และความเป็นผู้นำ ตามตัวอย่างรูปที่ 2.3-2.5



รูปที่ 2.3 ตัวอย่างผลการวิเคราะห์ของโครงการ P1 [11]

Management tool	Filename	Developer ID
bower (js/css manager tools)	bower.json	P1-6
composer (php)	composer.json	P1-6
gulp (nodejs)	gulpfile.js	P1-6
composer (php)	modules/Agent/composer.json	P1-1
composer (php)	modules/Area/composer.json	P1-1
composer (php)	modules/Common/composer.json	P1-3
composer (php)	modules/Master/composer.json	P1-3
composer (php)	modules/Shop/composer.json	P1-1
composer (php)	modules/Trade/composer.json	P1-6
composer (php)	modules/User/composer.json	P1-3
npm (nodejs package manager tools)	package.json	P1-6
bower (js/css manager tools)	public/master-static/bower.json	P1-3

รูปที่ 2.4 ประวัติการสร้างสภาพแวดล้อมของโครงการ P1 [11]

Developer ID	(A) Develop-ment areas	(B) Contrib-ution	(D) Support	(E) Leader-ship
P1-1	CM	1	false	false
P1-2	-	1	false	false
P1-3	CVM	3	true	true
P1-4	CVM	3	true	false
P1-5	CV	2	false	false
P1-6	CVM	3	true	true

รูปที่ 2.5 ตัวอย่างการแยกแยะลักษณะของนักพัฒนาซอฟต์แวร์ [11]

จากผลลัพธ์ของการวิเคราะห์ของโครงการ P1 ดังรูปที่ 2.3 มีนักพัฒนาซอฟต์แวร์จำนวน 6 คนที่ทำการพัฒนาโครงการนี้ โดยจะแบ่งผลลัพธ์ออกเป็น 3 ประเภทคือ ส่วนที่พัฒนาใหม่ (New), ส่วนที่ทำการปรับปรุงแก้ไข (Others) และส่วนที่มีทั้งการพัฒนาใหม่และปรับปรุงแก้ไข (New + Others) ซึ่งจะสามารถสรุปได้ว่านักพัฒนาซอฟต์แวร์ P1-6 มีสัดส่วนการสร้างและแก้ไขโค้ดของโครงการนี้มากกว่า 50 เปอร์เซ็นต์ ทำให้ระดับการมีส่วนร่วมนั้นสูงมาก นอกจากนี้นักพัฒนาซอฟต์แวร์ P1-6 ยังมีคุณลักษณะในด้านการสนับสนุนและความคิดริเริ่ม เนื่องจากได้มีการสร้างสภาพแวดล้อมของโครงการและใช้เครื่องมือในการจัดการหลายครั้งดังรูปที่ 2.4 ดังนั้นนักพัฒนาซอฟต์แวร์ P1-6 จึงมีลักษณะความเป็นผู้นำ ในขณะเดียวกันนักพัฒนาซอฟต์แวร์ P1-4 มีทั้งคุณลักษณะในด้านการมีส่วนร่วมและการสนับสนุน แต่นักพัฒนาซอฟต์แวร์ P1-4 ไม่ได้มีส่วนร่วมในการสร้างสภาพแวดล้อมของโครงการ ดังนั้นจึงไม่มีคุณลักษณะด้านความเป็นผู้นำ นอกจากนี้นักพัฒนาซอฟต์แวร์ P1-2 มีส่วนเกี่ยวข้องในพื้นที่การพัฒนาในชั้นของ MVC อยู่บ้างแต่ถือว่าน้อยมาก และผลงานของนักพัฒนาซอฟต์แวร์อยู่ในระดับต่ำ ซึ่งสะท้อนว่านักพัฒนาซอฟต์แวร์ P1-2 ได้เข้าร่วมโครงการหลังจากที่โครงการได้ดำเนินการไปแล้ว เนื่องจากโครงการ P1 เป็นโครงการของผู้วิจัยในงานวิจัยเรื่องนี้จึงสามารถยืนยันได้ว่าผลลัพธ์ที่แสดงในรูป 2.5 สอดคล้องกับคุณลักษณะที่แท้จริงของนักพัฒนาซอฟต์แวร์ในโครงการ

ผู้วิจัยได้นำแนวคิดของงานวิจัยดังกล่าวในการนำระบบควบคุมเวอร์ชันแบบกระจายศูนย์มาใช้เพื่อทำการแยกแยะคุณลักษณะของนักพัฒนาซอฟต์แวร์ แต่จะพิจารณาในแง่ใช้ในการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์โดยอิงร่องรอยที่ไม่ดีในโค้ดในระบบควบคุมเวอร์ชันแบบกระจายศูนย์

2.2.3. The Application of the Function Point Analysis in Software Developers' Performance Evaluation

งานวิจัยของ Chen Ting และคณะ [12] ได้นำเสนอขั้นตอนการวิเคราะห์ฟังก์ชันพอยต์ (Function Point Analysis) เพื่อคำนวณหาปริมาณงานสำหรับนักพัฒนาซอฟต์แวร์ โดยผลการประเมินสามารถใช้เป็นข้อมูลอ้างอิงในการประเมินภาระงานและประสิทธิภาพในการทำงานของผู้พัฒนาซอฟต์แวร์ การวิเคราะห์ฟังก์ชันพอยต์เป็นวิธีหนึ่งในการวัดปริมาณงานในการพัฒนาซอฟต์แวร์ผ่านรูปแบบของการคาดการณ์ตามฟังก์ชันของซอฟต์แวร์ เมื่อทำการวิเคราะห์ความต้องการของโครงการและมีการคำนวณฟังก์ชันพอยต์ที่สอดคล้องกัน ฟังก์ชันพอยต์จะแสดงเป็นขนาดของซอฟต์แวร์และเปลี่ยนเป็นขนาดของภาระงาน ประเภทของฟังก์ชันแบ่งออกเป็น 5 ประเภท ได้แก่ 1) Internal Logical File (ILF) คือข้อมูลเชิงตรรกะที่ประมวลผลภายในระบบ เช่นการประมวลผลของข้อมูลในฐานข้อมูล 2) External Interface File (EIF) คือข้อมูลที่ต้องการซึ่งถูกส่งมาจากภายนอก เพื่อมาประมวลผลในระบบ 3) External Input (EI) คือข้อมูลที่เข้ามาจากภายนอก เช่น ข้อมูลที่มาจากผู้ใช้ 4) External Output (EO) คือข้อมูลที่ส่งออกสู่ภายนอกและ 5) External Inquery คือข้อมูลที่ถูกประมวลผลจากภายนอกแล้วส่งมาในระบบ ขั้นตอนในการวิเคราะห์มีดังนี้

1) กำหนดขอบเขตของการคำนวณ

ทำการกำหนดบรรทัดฐานการคำนวณของจุดฟังก์ชัน และขอบเขตของระบบ

2) วิเคราะห์ฟังก์ชันพอยต์

ทำการแยกแยะและคาดการณ์องค์ประกอบต่าง ๆ ที่ต้องมีการประเมินของซอฟต์แวร์เช่น จำนวนข้อมูลเข้า จำนวนข้อมูลออก จำนวนตารางข้อมูล จำนวนไฟล์ข้อมูลเชิงตรรกะภายในและภายนอก เพื่อกำหนดจำนวนฟังก์ชันพอยต์

i. คำนวณฟังก์ชันพอยต์

หลังจากได้จำนวนของฟังก์ชันทั้ง 5 ประเภท จึงนำแต่ละประเภทไปคูณกับค่าน้ำหนักของความซับซ้อน (Weighting Factor) เพื่อให้ได้ค่าฟังก์ชันพอยต์ที่ยังไม่ได้ปรับค่า (Unadjusted Function Point: UFP) โดยค่าน้ำหนักของความซับซ้อนจะถูกกำหนดด้วย International Function Point Users Group (IFPUG) โดยในแต่ละประเภทของฟังก์ชันจะมีค่าน้ำหนักของความซับซ้อนอยู่ 3 ระดับตามความซับซ้อนของฟังก์ชันโดยรวมแต่ละประเภท

ii. กำหนดความซับซ้อนของการพัฒนาซอฟต์แวร์

ทำการหาค่าความซับซ้อนของตัวแปรที่มีผลกระทบ (Technical Complexity Factor: TCF) โดยประกอบด้วยตัวแปร 14 ค่า ดังรูปที่ 2.6 โดยค่าความซับซ้อนของตัวแปรที่มีผลกระทบจะมีค่าระหว่าง 0 ถึง 5

TABLE II. TECHNOLOGY COMPLEXITY FACTORS

Sequence number	Adjustment parameter	Description
1	E1	Data communications
2	E2	Performance
3	E3	Heavily used configuration
4	E4	Transaction rate
5	E5	Online data entry
6	E6	End user efficiency
7	E7	Online update
8	E8	Complex processing
9	E9	Reusability case
10	E10	Installation case
11	E11	Operations case
12	E12	Multiple case
13	E13	Facilitate change
14	E14	Distributed functions

รูปที่ 2.6 ตัวแปรของค่าความซับซ้อน [12]

หลังจากให้ค่าตัวแปรทั้งหมดแล้วจะต้องนำมารวมกันทั้งหมดเพื่อที่จะได้ค่าความซับซ้อนของตัวแปรที่มีผลกระทบ และนำค่าดังกล่าวมาคำนวณหาค่าความซับซ้อนของการพัฒนาซอฟต์แวร์ (Value Adjustment Factor: VAF) โดยมีสมการดังนี้

$$VAF = 0.65+0.01\times TCF \quad (2)$$

1. คำนวณหาค่าฟังก์ชันพอยต์ที่ปรับค่า

หลังจากได้ค่าฟังก์ชันพอยต์ที่ยังไม่ได้ปรับค่าและค่าความซับซ้อนของการพัฒนาซอฟต์แวร์มาแล้ว ก็จะสามารถคำนวณหาค่าฟังก์ชันพอยต์ที่ปรับค่า (Adjusted Function Point: AFP) ได้ตามสมการดังนี้

$$AFP = UFP\times VAF \quad (3)$$

งานวิจัยนี้อภิปรายว่าค่าฟังก์ชันพอยต์ที่ปรับค่านั้นนอกจากจะระบุขนาดของซอฟต์แวร์ที่จะพัฒนาได้แล้ว ยังแสดงถึงขนาดของภาระงานในการพัฒนาซอฟต์แวร์และยังบอกถึงประสิทธิภาพในการทำงานของนักพัฒนาซอฟต์แวร์จากขนาดของภาระงานในการพัฒนาซอฟต์แวร์ด้วยโดยนักพัฒนาซอฟต์แวร์ที่สามารถพัฒนาซอฟต์แวร์ขนาดใหญ่ จึงเป็นผู้มีประสิทธิภาพ

ผู้วิจัยได้นำแนวคิดของงานวิจัยดังกล่าวในการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์จากการวิเคราะห์ตัวซอฟต์แวร์โดยตรงพิจารณาจากภาระงานในการพัฒนาซอฟต์แวร์ แต่จะพิจารณาโดยอิงร่องรอยที่ไม่ดีซึ่งอยู่ในระบบควบคุมเวอร์ชันแบบกระจายศูนย์

2.2.4. TopCoder

TopCoder [13] เป็นชุมชนของคนที่มีความสามารถมารวมกันเพื่อทำการแข่งขัน ทำงานร่วมกัน และแลกเปลี่ยนความรู้ด้านการพัฒนาซอฟต์แวร์ Topcoder จะมีสามแทร็กของการแข่งขัน คือ การออกแบบ การพัฒนา และวิทยาศาสตร์ข้อมูล และในแทร็กเหล่านี้ ผู้ที่เข้าร่วมแข่งขันจะทำงานที่ท้าทายตามโจทย์ที่เกิดขึ้นจริงในปัจจุบันเพื่อแก้ปัญหาแก่ลูกค้าที่อยู่ในอันดับ Global 2000 โจทย์ระบบจริงซึ่งมีความซับซ้อนจะถูกแบ่งออกเป็นประเภทการแข่งขันเพื่อให้ผู้เข้าแข่งขันสามารถทำงานตามความชำนาญ เช่น การออกแบบกราฟฟิก การสร้างต้นแบบ การออกแบบโซลูชันอัลกอริทึม หรือการเขียนโค้ด และนอกจากนี้จะมีการแข่งขันกันภายในชุมชน TopCoder เพื่อทดสอบทักษะด้านวิทยาศาสตร์ข้อมูลในประเภทต่าง ๆ และการเขียนโปรแกรม โดยผลงานที่ผู้เข้าร่วมได้ทำมานั้นจะมีการตรวจสอบโดยคณะกรรมการตรวจสอบของ TopCoder ว่าผลงานที่ทำมามีคุณภาพสูง TopCoder ยังได้พัฒนาระบบการให้คะแนนของผู้เข้าร่วมการแข่งขันโดยอิงจากวิธีการหาค่าเฉลี่ย Elo ซึ่งเป็นการคำนวณระดับทักษะของผู้เล่นเมื่อเทียบกับคู่แข่งในเกมคู่แข่ง เช่น หมากรุก เป็นต้น จากนั้นจะมีการให้รางวัลเป็นเหรียญและสิ่งของอื่น ๆ กับผู้เข้าร่วมการแข่งขันในทุกแทร็ก และทำให้คะแนนของผู้เข้าแข่งขันดีขึ้น

นักวิจัยได้นำแนวคิดในการให้คะแนนจากการเข้าแข่งขันของผู้เข้าร่วมแข่งขันในเว็บไซต์ดังกล่าว แต่จะใช้วิธีการให้คะแนนโดยวิธีการหาค่าเฉลี่ยเบย์เซียนแทนโดยอิงจากร่องรอยที่ไม่ดีในโค้ดของนักพัฒนาซอฟต์แวร์แต่ละราย

บทที่ 3
การวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์โดยอิงร่องรอยที่ไม่ดีในโค้ดในระบบควบคุมเวอร์ชันแบบกระจายศูนย์

งานวิจัยนี้ได้นำเสนอวิธีการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์โดยอิงร่อยรอยที่ไม่ดีในโค้ดในระบบควบคุมเวอร์ชันแบบกระจายศูนย์ โดยมีแนวทางการดำเนินงานทั้งหมด 6 ขั้นตอน ดังนี้

3.1 การกำหนดสภาพแวดล้อมที่จะใช้ในการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์

ในขั้นตอนการกำหนดสภาพแวดล้อมที่จะใช้ในการวิเคราะห์นั้น มีจุดประสงค์เพื่อที่จะจำกัดสภาพแวดล้อมสำหรับการวัดประสิทธิภาพของซอฟต์แวร์ โดยจะกำหนดสภาพแวดล้อมที่ใช้ในการวิเคราะห์ดังนี้

3.1.1. การกำหนดภาษาของโครงการที่จะนำมาวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์

ในขั้นตอนกำหนดภาษาของโครงการที่จะนำมาวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์นั้น จะกำหนดให้นำโครงการที่ใช้ภาษาซีชาร์ปในการพัฒนามาทำการวัดประสิทธิภาพเท่านั้น

3.1.2. การกำหนดระบบควบคุมเวอร์ชันที่ใช้ในโครงการที่จะนำมาวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์

ในขั้นตอนกำหนดระบบควบคุมเวอร์ชันที่ใช้ในโครงการที่จะนำมาวัดประสิทธิภาพนั้น จะกำหนดให้ใช้ได้กับระบบควบคุมเวอร์ชัน GitHub เท่านั้น

3.2 การกำหนดประเภทของร่องรอยที่ไม่ดีในโค้ดที่จะนำมาวัดประสิทธิภาพ

ในขั้นตอนกำหนดประเภทของร่องรอยที่ไม่ดีในโค้ดที่จะนำมาวัดประสิทธิภาพนั้น มีจุดประสงค์เพื่อจำกัดประเภทของร่องรอยที่ไม่ดีในโค้ดที่จะนำมาวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์ โดยประเภทของร่องรอยที่ไม่ดีในโค้ดที่ผู้วิจัยจะนำมาวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์ได้แก่ ร่องรอยที่ไม่ดีในโค้ดด้านการออกแบบ และร่องรอยที่ไม่ดีในโค้ดด้านการพัฒนา เนื่องจากร่องรอยที่ไม่ดีในโค้ดของสองประเภทนี้มีแนวโน้มที่จะเกิดสูงสุด ซึ่งรายละเอียดร่องรอยที่ไม่ดีในโค้ดสองประเภทนี้ได้นิยามแสดงไว้ในหัวข้อที่ 2.1.2

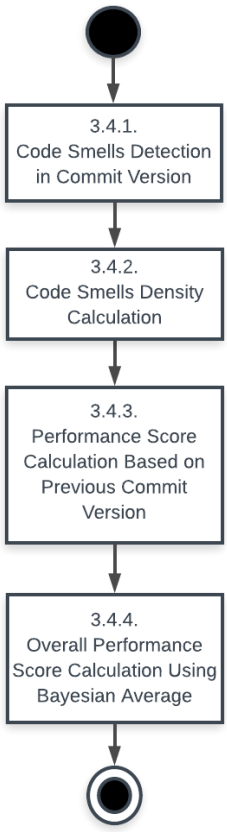
3.3 การกำหนดเครื่องมือที่จะใช้ในการวิเคราะห์เพื่อหาร่องรอยที่ไม่ดีในโค้ด

ในขั้นตอนกำหนดเครื่องมือที่จะใช้ในการวิเคราะห์เพื่อหาร่องรอยที่ไม่ดีในโค้ดนั้น มีจุดประสงค์เพื่อที่จะนำเครื่องมือที่ใช้ในการวิเคราะห์เพื่อหาร่องรอยที่ไม่ดีในโค้ดมาตรวจหาร่องรอยที่ไม่ดีในโค้ดและแยกประเภทของร่องรอยที่ไม่ดีในโค้ด เพื่อจะนำไปวัดประสิทธิภาพของซอฟต์แวร์ โดย

จะกำหนดให้ใช้เครื่องมือที่มีชื่อว่า Designite ในการวิเคราะห์เพื่อหาร่องรอยที่ไม่ดีในโค้ด โดยเครื่องมือนี้สามารถแยกแยะประเภทของร่องรอยที่ไม่ดีในโค้ดตามที่นิยามไว้ในหัวข้อที่ 2.1.2 นอกจากนั้นเครื่องมือนี้ยังสามารถส่งผลลัพธ์ออกมาในรูปแบบไฟล์เอกเซล

3.4 การกำหนดวิธีการวัดประสิทธิภาพ

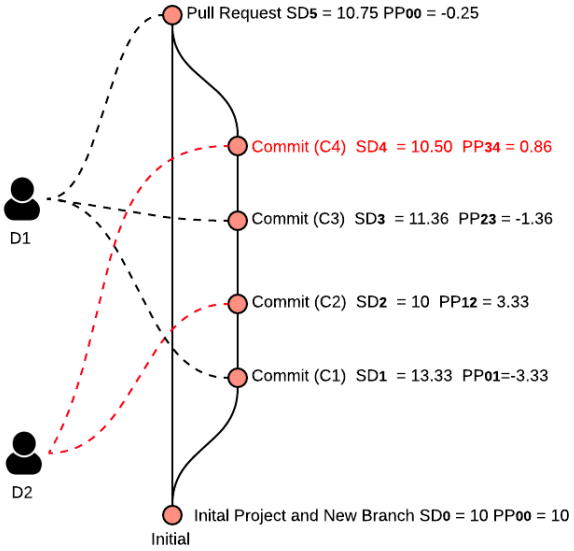
ในขั้นตอนการกำหนดวิธีการวัดประสิทธิภาพนั้น มีจุดประสงค์เพื่อใช้เป็นขั้นตอนในการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์รายคนที่ร่วมอยู่ในโครงการที่มีการพัฒนาโดยใช้ระบบควบคุมเวอร์ชันแบบกระจายศูนย์ ซึ่งจะประกอบด้วยขั้นตอนดังรูปที่ 3.1 โดยมีรายละเอียดดังนี้



รูปที่ 3.1 ขั้นตอนการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์รายคนที่ร่วมอยู่ในโครงการที่มีการพัฒนาโดยใช้ระบบควบคุมเวอร์ชันแบบกระจายศูนย์

3.4.1. การตรวจหาร่องรอยที่ไม่ดีในเวอร์ชันคอมมิต

ในระบบควบคุมเวอร์ชันแบบกระจายศูนย์นอกจากสามารถให้นักพัฒนาซอฟต์แวร์ทำการคอมมิต (Commit) โค้ดที่นักพัฒนาซอฟต์แวร์เพิ่มหรือแก้ไขแล้ว ยังสามารถสร้างสาขาใหม่ (New Branch) หรือรวมสาขาเข้าด้วยกัน (Pull Request) โดยจะแสดงโครงสร้างการคอมมิตบนระบบควบคุมเวอร์ชันแบบกระจายศูนย์ตามตัวอย่างในรูปที่ 3.2



รูปที่ 3.2 ตัวอย่างโครงสร้างการคอมมิตบนระบบควบคุมเวอร์ชันแบบกระจายศูนย์

งานวิจัยนี้จะนำเฉพาะการกระทำของนักพัฒนาซอฟต์แวร์ที่เป็นการคอมมิตโค้ดมายังเซิร์ฟเวอร์ของระบบควบคุมเวอร์ชันแบบกระจายศูนย์มาวัดประสิทธิภาพเท่านั้น เนื่องจากการคอมมิตเป็นการกระทำโดยที่นักพัฒนาซอฟต์แวร์ทำการสร้างหรือแก้ไขโค้ดภายในโครงการ แต่ในการกระทำอื่น ๆ นั้นไม่ใช่การกระทำที่ทำการสร้างหรือแก้ไขโค้ดภายในโครงการเช่น การสร้างสาขาใหม่ เป็นต้น หรืออาจจะมีบางการกระทำที่ทำการสร้างหรือแก้ไขโค้ดภายในโครงการโดยทางอ้อมเช่น การรวมสาขาเข้าด้วยกัน เป็นต้น แต่การรวมสาขาเข้าด้วยกันนั้น ผู้รวมอาจไม่ใช่ผู้พัฒนาก็ได้จึงไม่นำมาพิจารณา

ผู้วิจัยจะยกตัวอย่างการวัดประสิทธิภาพในแต่ละขั้นตอนจากที่แสดงในรูปที่ 3.1 มาเพื่อแสดงการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์ โดยจะยกตัวอย่างจากตัวอย่างโครงสร้างการคอมมิตจากรูปที่ 3.2 ผู้วิจัยจะทำการเลือกเฉพาะการกระทำที่เป็นการคอมมิตมาวัดประสิทธิภาพได้แก่ C1, C2, C3 และ C4 เท่านั้น ซึ่งในตัวอย่างที่จะแสดงในหัวข้อที่ 3.4.2-3.4.4 ดังต่อไปนี้จะทำการวัดประสิทธิภาพนักพัฒนาซอฟต์แวร์ D2 โดยจะคำนวณจากคอมมิต C4 ซึ่งเป็นคอมมิตล่าสุดที่นักพัฒนาซอฟต์แวร์รายนี้คอมมิตเข้ามาในระบบควบคุมเวอร์ชันแบบกระจายศูนย์

3.4.2. การคำนวณหาความหนาแน่นของร่องรอยที่ไม่ดีในโค้ด

การคำนวณหาความหนาแน่นของร่องรอยที่ไม่ดีในโค้ด (Code Smell Density) [10] เป็นขั้นตอนหนึ่งในการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์ โดยจะใช้จำนวนของร่องรอยที่ไม่ดีในโค้ดมาทำการคำนวณกับจำนวนของบรรทัดของโค้ด โดยมีสมการดังนี้

$$SD_j = \frac{\text{Number of Smells in Code}_j}{KLOC_j} \quad (4)$$

โดยให้ j คือ เวอร์ชันปัจจุบัน

SD_j คือ ค่าความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดเวอร์ชันปัจจุบัน

$\text{Number of Smells in Code}_j$ คือ จำนวนร่องรอยที่ไม่ดีในโค้ดเวอร์ชันปัจจุบัน

$KLOC_j$ คือ จำนวนต่อหนึ่งพันบรรทัดของโค้ดเวอร์ชันปัจจุบัน

จากรูปที่ 3.2 กำหนดให้จำนวนของร่องรอยที่ไม่ดีในโค้ดเวอร์ชันปัจจุบันซึ่งก็คือเวอร์ชัน C4 ที่นำมายกตัวอย่างมีค่าเป็น 25 และกำหนดให้ค่าจำนวนต่อหนึ่งพันบรรทัดของโค้ดเวอร์ชันปัจจุบันมีค่าเป็น 2.38 ดังนั้นจากสมการที่ (4) ค่าความหนาแน่นของร่องรอยที่ไม่ดีในโค้ด (Code Smell Density) ในเวอร์ชันปัจจุบันมีค่าเป็น 10.50 จุดต่อหนึ่งพันบรรทัด โดยจะแสดงตัวอย่างการคำนวณในตารางที่ 3.1 และกำหนดให้ C1, C2, C3 มีค่าความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดมีค่าเป็น 13.33, 10, 11.36 จุดต่อหนึ่งพันบรรทัดตามลำดับ

ตารางที่ 3.1 รายละเอียดการคำนวณค่าความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดเวอร์ชันปัจจุบัน

$\text{Number of Smells in Code}_j$	25
$KLOC_j$	2.38
SD_j	$\frac{25}{2.38} = 10.50$

3.4.3.การคำนวณหาค่าประสิทธิภาพของนักพัฒนาซอฟต์แวร์ที่เวอร์ชันคอมมิตปัจจุบันเมื่อเทียบกับเวอร์ชันคอมมิตก่อนหน้า

การคำนวณหาค่าประสิทธิภาพของนักพัฒนาซอฟต์แวร์ที่เวอร์ชันคอมมิตปัจจุบันเมื่อเทียบกับเวอร์ชันคอมมิตก่อนหน้า เป็นขั้นตอนที่คำนวณจากค่าความหนาแน่นของร่องรอยที่ไม่ดีที่เปลี่ยนแปลงไป โดยผู้วิจัยจะเรียกค่านี้ว่า ค่าประสิทธิภาพของนักพัฒนาซอฟต์แวร์ที่เวอร์ชันคอมมิตปัจจุบันเมื่อเทียบกับเวอร์ชันคอมมิตก่อนหน้า โดยมีสมการดังนี้

$$PP_{ij} = SD_i - SD_j \quad (5)$$

โดยให้ i คือ เวอร์ชันคอมมิตก่อนหน้า

j คือ เวอร์ชันคอมมิตปัจจุบัน

PP_{ij} คือ ค่าประสิทธิภาพของนักพัฒนาซอฟต์แวร์ที่เวอร์ชันคอมมิตปัจจุบันเมื่อเทียบกับเวอร์ชันคอมมิตก่อนหน้า (พิจารณาจากค่าความหนาแน่นของร่องรอยที่ไม่ดีที่เปลี่ยนแปลงไป)

SD_i คือ ค่าความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดเวอร์ชันคอมมิตก่อนหน้า

SD_j คือ ค่าความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดเวอร์ชันคอมมิตปัจจุบัน

จากรูปที่ 3.2 SD_i จะแทนด้วยค่าความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดเวอร์ชันซึ่งมีค่าก่อนหน้า 11.36 จุดต่อหนึ่งพันบรรทัด และ SD_j จะแทนด้วยค่าความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดเวอร์ชันปัจจุบันซึ่งมีค่า 10.50 จุด ซึ่งค่าความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดเวอร์ชันปัจจุบันมีค่าลดลงจากเวอร์ชันก่อนหน้า ดังนั้นจากสมการที่ (5) PP_{ij} จะมีค่าเท่ากับ 0.86 โดยจะแสดงตัวอย่างการคำนวณในตารางที่ 3.2 ซึ่งหมายความว่านักพัฒนาซอฟต์แวร์ D2 สามารถทำให้โค้ดเวอร์ชันปัจจุบันนั้นมีคุณภาพที่ดีขึ้นจากเวอร์ชันก่อนหน้า (แต่ถ้าค่า PP_{ij} จากการคำนวณเท่ากับ 0 หมายความว่านักพัฒนาซอฟต์แวร์ไม่ได้ทำให้โค้ดเวอร์ชันปัจจุบันมีคุณภาพดีขึ้นหรือแย่ลง หรือถ้าค่า PP_{ij} จากการคำนวณน้อยกว่า 0 หมายความว่า นักพัฒนาซอฟต์แวร์ทำให้โค้ดเวอร์ชันปัจจุบันมีคุณภาพแย่ลง) ดังนั้นค่า PP_{ij} ของ C1, C2 และ C3 จะมีค่าเท่ากับ -3.33, 3.33 และ -1.36 ตามลำดับ

ตารางที่ 3.2 รายละเอียดการคำนวณค่าประสิทธิภาพของนักพัฒนาซอฟต์แวร์ที่เวอร์ชันคอมมิตปัจจุบันเมื่อเทียบกับเวอร์ชันคอมมิตก่อนหน้า

SD_i	11.36
SD_j	10.50
PP_{ij}	11.36 - 10.50 = 10.50

3.4.4.การคำนวณหาประสิทธิภาพโดยรวมของนักพัฒนาซอฟต์แวร์โดยใช้สมการหาค่าเฉลี่ยเบย์เซียน

การคำนวณหาค่าประสิทธิภาพโดยรวมของนักพัฒนาซอฟต์แวร์โดยใช้สมการหาค่าเฉลี่ยเบย์เซียน เป็นขั้นตอนที่ใช้คำนวณหาค่าประสิทธิภาพโดยรวมของนักพัฒนาซอฟต์แวร์ที่คอมมิตโค้ดในเวอร์ชันปัจจุบัน โดยใช้สมการดังนี้

$$PB_r = \frac{((N_a \times A_a)+(N_r \times A_r))}{(N_a+N_r)} \quad (6)$$

โดยให้ PB_r คือค่าประสิทธิภาพโดยรวมของนักพัฒนาซอฟต์แวร์ที่คอมมิตโค้ดในเวอร์ชันปัจจุบัน

r คือ นักพัฒนาซอฟต์แวร์ที่คอมมิตโค้ดในเวอร์ชันปัจจุบัน

a คือ นักพัฒนาซอฟต์แวร์ทั้งหมดในโครงการ

N_a คือ ค่าเฉลี่ยเลขคณิตของจำนวนครั้งในการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์ทั้งหมดแต่ละรายในโครงการ โดยหาจากสมการดังนี้

$$N_a = \frac{\text{No. of all commits in project}}{\text{No. of all software developers in project}} \quad (7)$$

A_a คือ ค่าเฉลี่ยเลขคณิตของค่าประสิทธิภาพที่คำนวณเทียบกับเวอร์ชันก่อนหน้าของนักพัฒนาซอฟต์แวร์ทั้งหมดในโครงการ โดยหาจากสมการดังนี้

$$A_a = \frac{\sum PP_{ij} \text{ of all commits in project}}{\text{No. of all commits in project}} \quad (8)$$

N_r คือ จำนวนครั้งในการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์ที่คอมมิตโค้ดในเวอร์ชันปัจจุบัน (คือ No. of commits in project by software developer r)

A_r คือ ค่าเฉลี่ยเลขคณิตของค่าประสิทธิภาพของนักพัฒนาซอฟต์แวร์ที่คอมมิตโค้ดในเวอร์ชันปัจจุบันที่คำนวณเทียบกับเวอร์ชันก่อนหน้า โดยหาจากสมการดังนี้

$$A_r = \frac{\sum PP_{ij} \text{ of commits in project by software developer r}}{\text{No. of all commits in project by software developer r}} \quad (9)$$

ดังนั้น N_a จะแทนด้วยค่าเฉลี่ยเลขคณิตของจำนวนครั้งในการวัดประสิทธิภาพทั้งหมดต่อนักพัฒนาซอฟต์แวร์ทั้งหมดแต่ละรายในโครงการ คือ 2 (4/2) และ A_a จะแทนด้วยค่าเฉลี่ยเลขคณิตของค่าประสิทธิภาพที่คำนวณเทียบกับเวอร์ชันก่อนหน้า คือ -0.13 และ N_r จะแทนด้วยค่าของจำนวนครั้งในการวัดประสิทธิภาพทั้งหมดของนักพัฒนาซอฟต์แวร์ที่คอมมิตโค้ดในเวอร์ชันปัจจุบัน คือ 2 และ A_r จะแทนด้วยค่าเฉลี่ยเลขคณิตของค่าประสิทธิภาพของนักพัฒนาซอฟต์แวร์ที่คอมมิตโค้ดในเวอร์ชันปัจจุบันที่คำนวณเทียบกับเวอร์ชันก่อนหน้า คือ 2.1 หลังจากคำนวณตามสมการที่กำหนดไว้ ดังนั้น PB จะมีค่า เท่ากับ 0.99 โดยจะแสดงตัวอย่างการคำนวณในตารางที่ 3.3

ตารางที่ 3.3 รายละเอียดการคำนวณค่าประสิทธิภาพโดยรวมของนักพัฒนาซอฟต์แวร์ที่คอมมิตโค้ดในเวอร์ชันปัจจุบัน

	C1	C2	C3	C4
PP	-3.33	3.33	-1.36	0.86
N_a	$\frac{4}{2} = 2$			
A_a	$\frac{(-3.33)+3.33+(-1.36)+0.86}{4} = -0.13$			
N_r	2			
A_r	$\frac{3.33+0.86}{2} = 2.1$			
PB_r	$\frac{(2\times(-0.13))+(2\times2.1)}{4} = 0.99$			

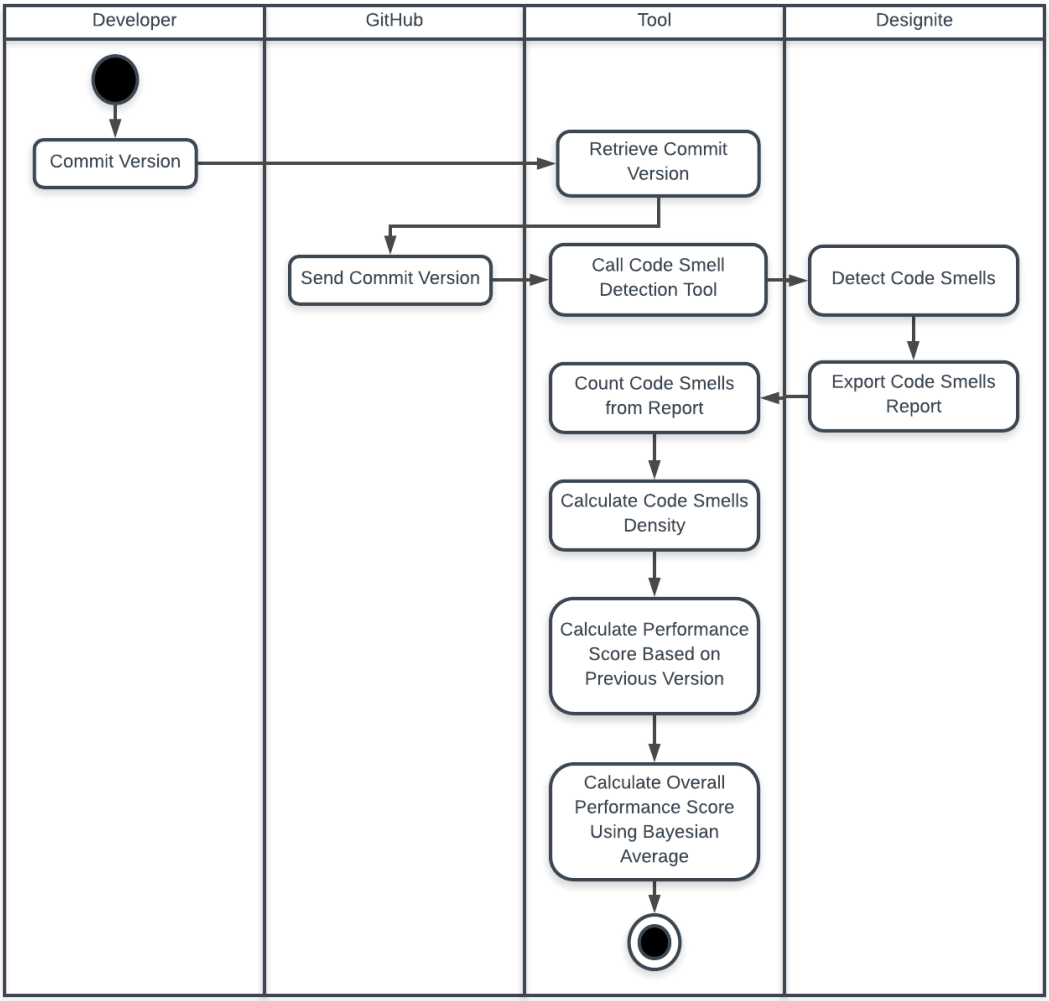
จากตัวอย่างข้างต้นซึ่งคำนวณตามขั้นตอนการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์รายคนที่ร่วมอยู่ในโครงการที่มีการพัฒนาโดยใช้ระบบควบคุมเวอร์ชันแบบกระจายศูนย์ดังรูปที่ 3.1 นั้น จะทำให้นักพัฒนาซอฟต์แวร์ D2 มีคะแนนในการวัดประสิทธิภาพโดยอิงร่องรอยที่ไม่ดีในโค้ดของโครงการนี้เท่ากับ 0.99 คะแนน เมื่อพิจารณาถึงคอมมิต C4

จากที่กล่าวมาแล้วในหัวข้อที่ 3.4.1 ว่าในการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์จะไม่นำการรวมสาขา (Pull Request) มาคำนวณค่าประสิทธิภาพของผู้ร่วมสาขา เช่น จากรูปที่ 3.2 การวัดประสิทธิภาพของนักพัฒนา D1 จะไม่พิจารณาโค้ดที่ได้จากการรวมสาขา เนื่องจาก

D1 ไม่ได้แก้ไขโค้ดแต่ทำการรวมสาขาเท่านั้น อย่างไรก็ตามยังมีการคำนวณค่า SD_{ij} และ PP_{ij} ณ จุดที่เป็นการรวมสาขาอยู่ เพื่อใช้คำนวณค่าประสิทธิภาพของนักพัฒนาซอฟต์แวร์ ณ จุดคอมมิดถัดไป

3.5 การออกแบบและพัฒนาเครื่องมือ

ในขั้นตอนออกแบบและพัฒนาเครื่องมือนั้น มีจุดประสงค์เพื่อใช้เครื่องมือเป็นต้นแบบในการสนับสนุนขั้นตอนของวิธีการที่เสนอ โดยผลลัพธ์ของเครื่องมือนี้จะสามารถแสดงถึงคะแนนของการวัดประสิทธิภาพรายคนทั้งภายในโครงการหนึ่งและทุกโครงการที่นักพัฒนาซอฟต์แวร์รายนั้นเข้าร่วม การทำงานของเครื่องมือที่ผู้วิจัยจะทำการพัฒนามีขั้นตอนการทำงานดังรูปที่ 3.3 รายละเอียดการออกแบบและพัฒนาดูได้ในบทที่ 4



รูปที่ 3.3 ขั้นตอนในการทำงานของเครื่องมือวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์รายคนที่มีการพัฒนาโดยใช้ระบบควบคุมเวอร์ชันแบบกระจายศูนย์

จากรูปที่ 3.3 สามารถอธิบายการทำงานของเครื่องมือวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์รายคนที่มีการพัฒนาโดยใช้ระบบควบคุมเวอร์ชันแบบกระจายศูนย์ โดยเริ่มจากนักพัฒนาซอฟต์แวร์ทำการคอมมิตโค้ดเวอร์ชันปัจจุบันเข้าไปใน Github จากนั้นนักพัฒนาซอฟต์แวร์จะใช้เครื่องมือเพื่อดึงคอมมิตเวอร์ชันล่าสุดจาก Github มาหาค่าร่องรอยที่ไม่ดีในโค้ดโดยใช้เครื่องมือ Designite ซึ่งเครื่องมือ Designite จะทำการส่งค่าร่องรอยที่ไม่ดีในโค้ดออกมาในรูปแบบไฟล์เอกเซล จากนั้นเครื่องมือจะทำการอ่านค่าจากไฟล์เอกเซลและนำค่าไปคำนวณหาค่าความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดเวอร์ชันปัจจุบัน จากนั้นจะนำค่าร่องรอยที่ไม่ดีในโค้ดเวอร์ชันปัจจุบันไปเทียบกับเวอร์ชันก่อนหน้า และนำค่าที่ได้ไปคำนวณหาค่าเฉลี่ยโดยใช้สมการหาค่าเฉลี่ยเบย์เซียนจึงได้ออกมาเป็นค่าประสิทธิภาพของนักพัฒนาซอฟต์แวร์โดยอิงร่องรอยที่ไม่ดีในโค้ด

3.6 การทดสอบและประเมินผล

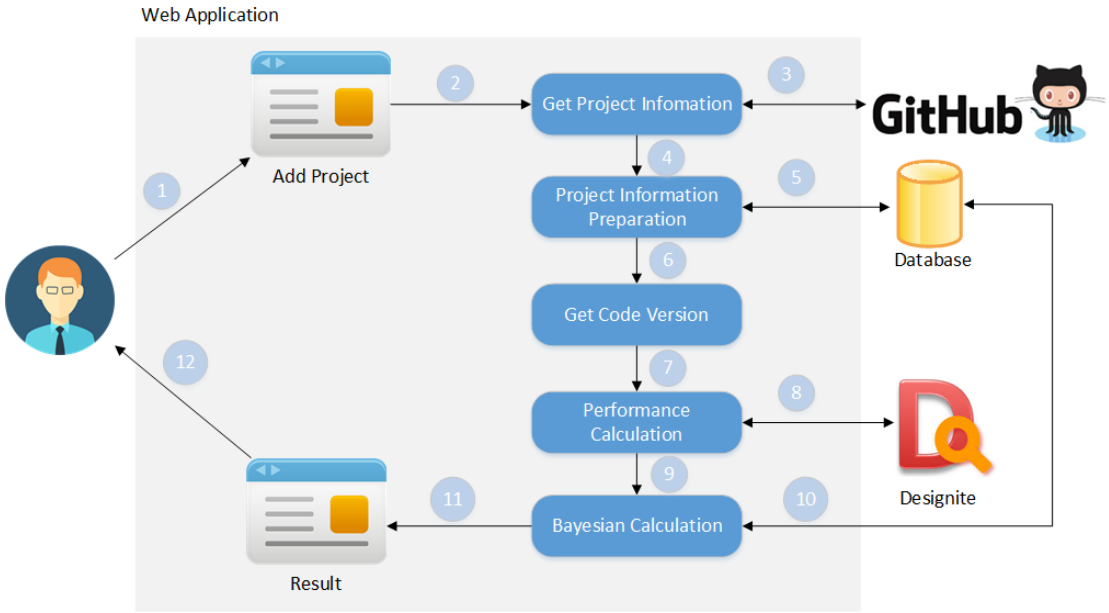
ในขั้นตอนการประเมินผล มีจุดประสงค์เพื่อตรวจสอบว่าการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์โดยอิงร่องรอยที่ไม่ดีในโค้ดบนระบบควบคุมเวอร์ชันแบบกระจายศูนย์นั้นทำให้สามารถจัดอันดับนักพัฒนาซอฟต์แวร์ไปในแนวทางเดียวกันกับการจัดอันดับโดยผู้เชี่ยวชาญหรือไม่

การประเมินผลทำโดยใช้โค้ดอย่างน้อย 2 โครงการและจะใช้นักพัฒนาซอฟต์แวร์ที่มีประสบการณ์และไม่ได้อยู่ในโครงการอย่างน้อยจำนวน 4 คน มาทำการรีวิวว่าโค้ดที่นักพัฒนาซอฟต์แวร์แต่ละรายได้ทำการพัฒนาในโครงการเดียวกันนั้นมีคุณภาพมากน้อยเพียงใด โดยจะทำการจัดอันดับว่านักพัฒนารายใดพัฒนาโค้ดได้มีประสิทธิภาพมากกว่ากัน และนำคะแนนที่ได้จากเครื่องมือที่ผู้วิจัยสร้างมาจัดอันดับ แล้วจึงนำมาทดสอบสมมติฐานทางสถิติว่ามีอันดับที่สอดคล้องกันมากน้อยเพียงใด รายละเอียดการทดสอบและประเมินผลดูได้ในบทที่ 5

บทที่ 4
การออกแบบและพัฒนาเครื่องมือ

4.1 ภาพรวมการออกแบบและพัฒนาเครื่องมือ

รูปที่ 4.1 เป็นภาพรวมแนวคิดการทำงานของเครื่องมือ เริ่มต้นจากผู้ใช้งานเข้าสู่เว็บแอปพลิเคชันที่เป็นระบบในการวัดประสิทธิภาพนักพัฒนาซอฟต์แวร์โดยอิงร่องรอยที่ไม่ดีในโค้ดที่โครงงานมหาบัณฑิตนี้ได้พัฒนาขึ้น หลังจากนั้นผู้ใช้สามารถทำการเพิ่มข้อมูลของโครงการที่ต้องการจะนำมาวัดประสิทธิภาพของนักพัฒนาได้ ระบบจะทำการส่งชื่อโครงการและชื่อเจ้าของโครงการโดยการเรียกใช้งานส่วนต่อประสานโปรแกรมประยุกต์หรือเอพีไอ (Application program interface: API) ของ Github ซึ่งนักพัฒนาสามารถเรียกใช้งานได้ หลังจากได้ข้อมูลของโครงการกลับมาจาก Github ระบบจะทำการจัดเตรียมข้อมูลเพื่อนำไปวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์แต่ละรายในทีมที่คอมมิตเข้าสู่ระบบในโครงการนั้น ๆ ซึ่งผลลัพธ์ที่ได้จากเครื่องมือคือคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์แต่ละรายในโครงการนั้น ๆ แล้วแสดงผลลัพธ์ให้ผู้ใช้งานเห็นทางหน้าจอ โดยภาพรวมแนวคิดการทำงานของเครื่องมือมีรายละเอียดดังนี้



รูปที่ 4.1 ภาพรวมแนวคิดการทำงานของเครื่องมือ

1) ผู้ใช้ทำการเพิ่มโครงการที่จะนำมาวัดประสิทธิภาพ โดยระบุชื่อโครงการและเจ้าของโครงการ

2) ระบบทำการเรียกฟังก์ชันเพื่อดึงข้อมูลโครงการที่จะนำมาวัดประสิทธิภาพ

3) ระบบทำการดึงข้อมูลโครงการจาก Github

4) ระบบนำข้อมูลโครงการมาจัดเก็บไว้ในรูปแบบสำหรับบันทึกลงฐานข้อมูล

5) ระบบนำข้อมูลโครงการที่จัดเก็บไว้มาเพิ่มลงในฐานข้อมูล

6) ระบบนำข้อมูลโครงการไปใช้เป็นพารามิเตอร์เพื่อดึงโค้ดแต่ละเวอร์ชัน

7) ระบบเตรียมโค้ดแต่ละเวอร์ชันเพื่อนำไปคำนวณประสิทธิภาพของนักพัฒนาซอฟต์แวร์

8) ระบบนำโค้ดไปคำนวณหาค่าร่องรอยที่ไม่ดีโดยใช้เครื่องมือ Designite

9) ระบบนำค่าร่องรอยที่ไม่ดีที่ได้มาคำนวณคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์โดยใช้สมการหาค่าเฉลี่ยเบย์เซียน

10) ระบบนำคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์มาเพิ่มลงในฐานข้อมูล

11) ระบบนำคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์มาแสดงบนเครื่องมือ

12) ผู้ใช้นำผลลัพธ์ที่ได้มาสนับสนุนการตัดสินใจในการจัดทีมนักพัฒนาซอฟต์แวร์

4.2 การวิเคราะห์และออกแบบเครื่องมือ

การวิเคราะห์และออกแบบเครื่องมือ ซึ่งแสดงถึงการวิเคราะห์และออกแบบด้วยแผนภาพต่าง ๆ แบ่งเป็น 4 ส่วนหลัก ดังนี้

4.2.1. การออกแบบแผนภาพยูสเคส

จากรูปที่ 4.2 แผนภาพยูสเคสแสดงหน้าที่การทำงานของเครื่องมือมีรายละเอียดดังนี้

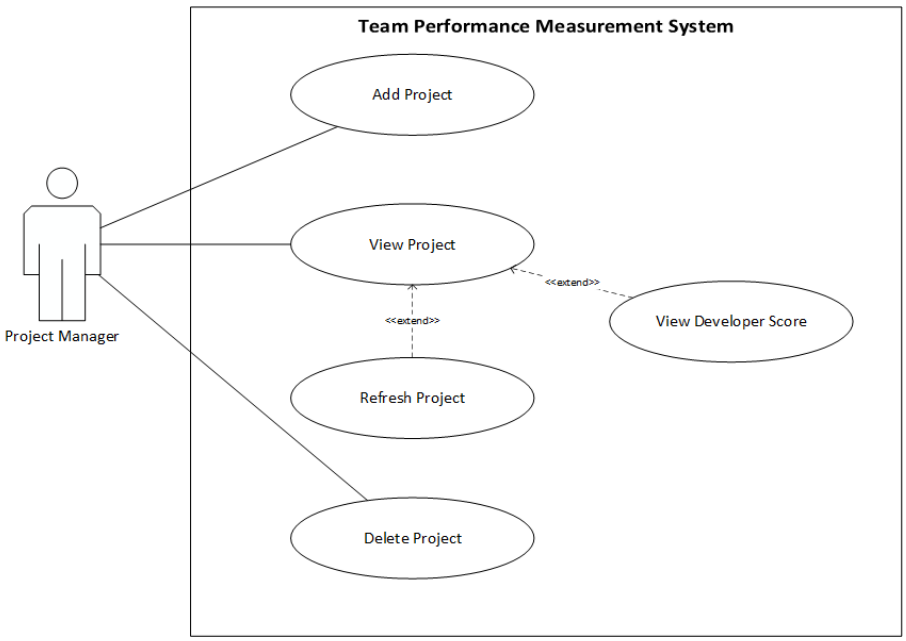
1) เพิ่มโครงการ (Add Project) ทำหน้าที่ในการเพิ่มโครงการที่จะนำมาวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์

2) แสดงโครงการ (View Project) ทำหน้าที่ในการแสดงคะแนนของนักพัฒนาซอฟต์แวร์ในโครงการที่นำมาวัดประสิทธิภาพ

3) แสดงคะแนนของนักพัฒนาซอฟต์แวร์รายหนึ่ง (View Developer Score) ทำหน้าที่ในการแสดงคะแนนของนักพัฒนาซอฟต์แวร์รายหนึ่งในทุกโครงการที่นักพัฒนาซอฟต์แวร์รายนั้นเกี่ยวข้อง

4) ปรับปรุงโครงการ (Refresh Project) ทำหน้าที่ในการปรับปรุงโครงการโดยจะทำการตรวจสอบว่า ถ้ามีคอมมิตเวอร์ชันใหม่ให้นำมาคำนวณหาคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์

5) ลบโครงการ (Delete Project) ทำหน้าที่ในการลบโครงการที่ไม่ต้องการนำมาวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์



รูปที่ 4.2 แผนภาพยูสเคสแสดงหน้าที่การทำงานของเครื่องมือ

4.2.2. การออกแบบแผนภาพคลาส

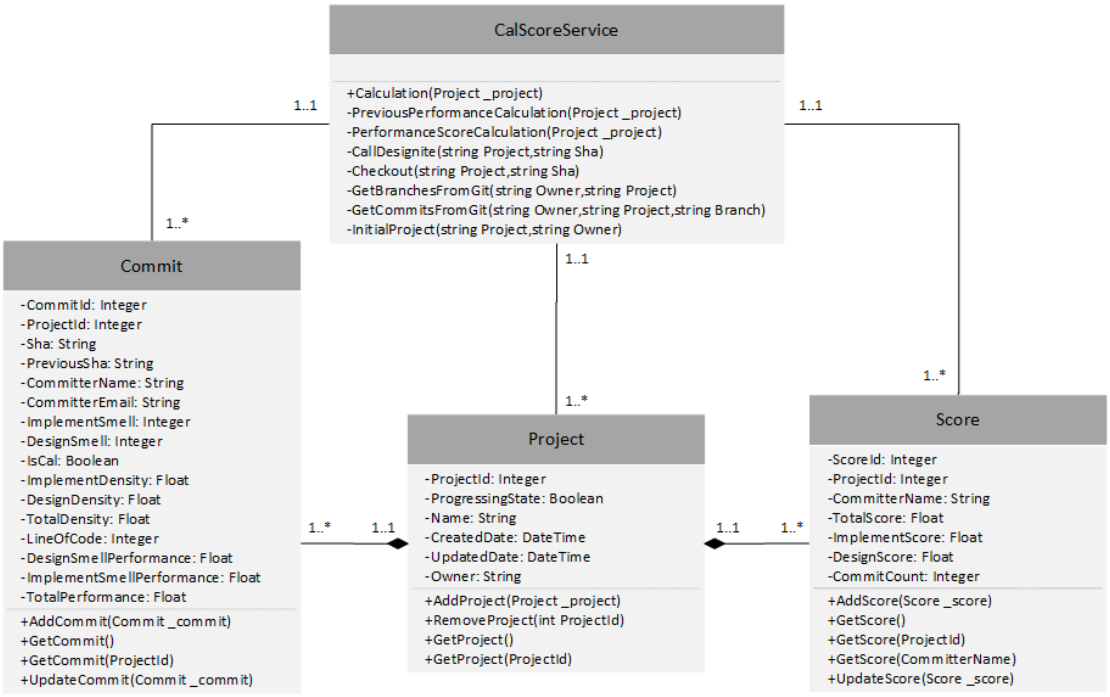
จากรูปที่ 4.3 แผนภาพคลาสแสดงหน้าที่การทำงานของเครื่องมือมีรายละเอียดดังนี้

1) คลาส CalScoreService ทำหน้าที่เป็นคลาสในการคำนวณประสิทธิภาพของนักพัฒนาซอฟต์แวร์ ติดต่อ Github และเรียกใช้งาน Designite

2) คลาส Project ทำหน้าที่จัดการโครงการ ได้แก่ เพิ่มโครงการ ลบโครงการ และดึงข้อมูลโครงการ เป็นต้น

3) คลาส Commit ทำหน้าที่จัดการคอมมิตของโครงการนั้น ๆ ได้แก่ เพิ่มคอมมิต แก้ไขคอมมิต และดึงข้อมูลคอมมิต เป็นต้น

4) คลาส Score ทำหน้าที่จัดการคะแนนของนักพัฒนาซอฟต์แวร์แต่ละรายของแต่ละโครงการ ได้แก่ เพิ่มคะแนน แก้ไขคะแนน และดึงคะแนนของนักพัฒนาซอฟต์แวร์ เป็นต้น

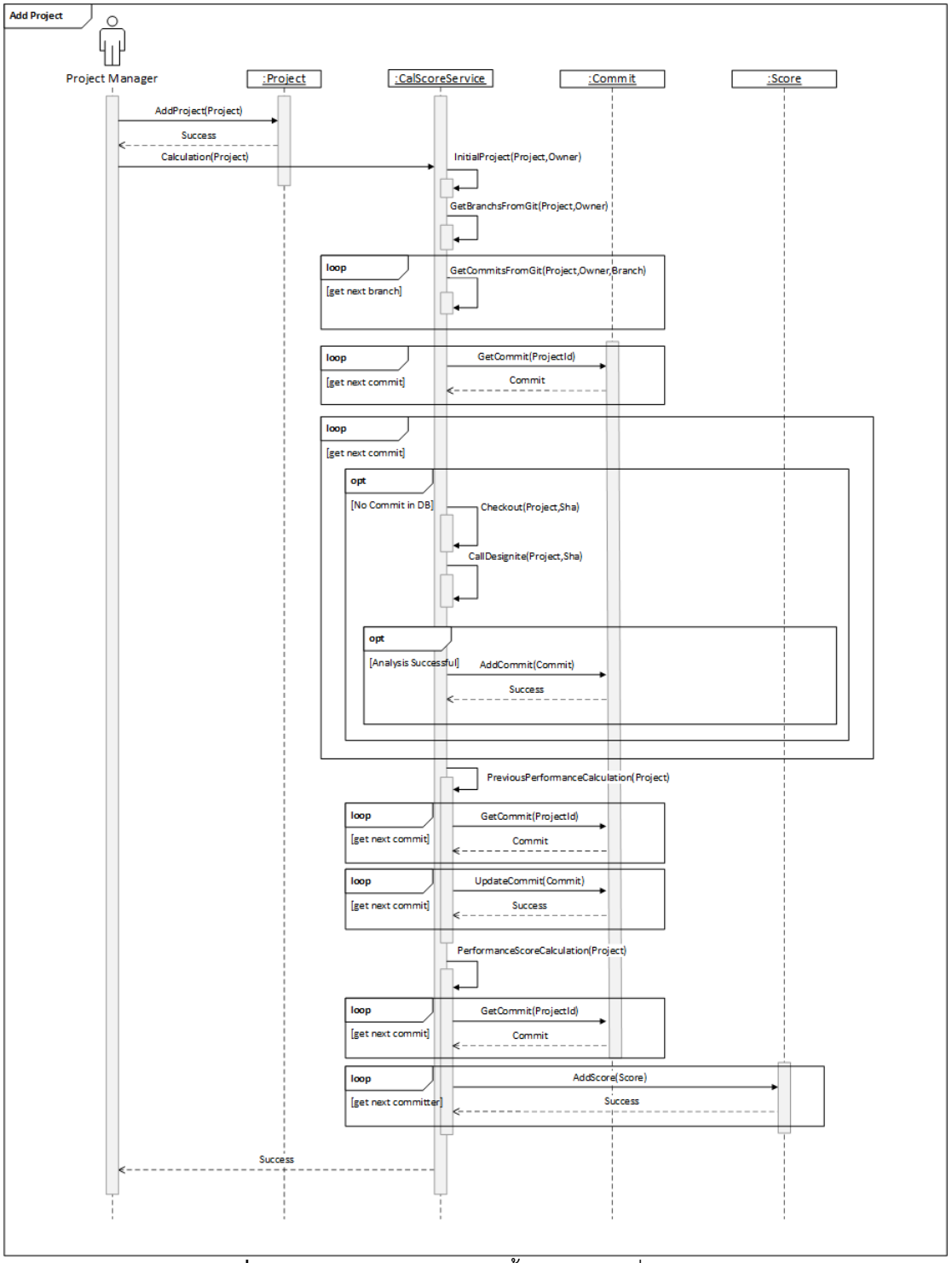


รูปที่ 4.3 แผนภาพคลาสแสดงหน้าที่การทำงานของเครื่องมือ

4.2.3. การออกแบบแผนภาพลำดับ

จากรูปที่ 4.4 แผนภาพลำดับแสดงขั้นตอนการเพิ่มโครงการมีลำดับการทำงานดังนี้

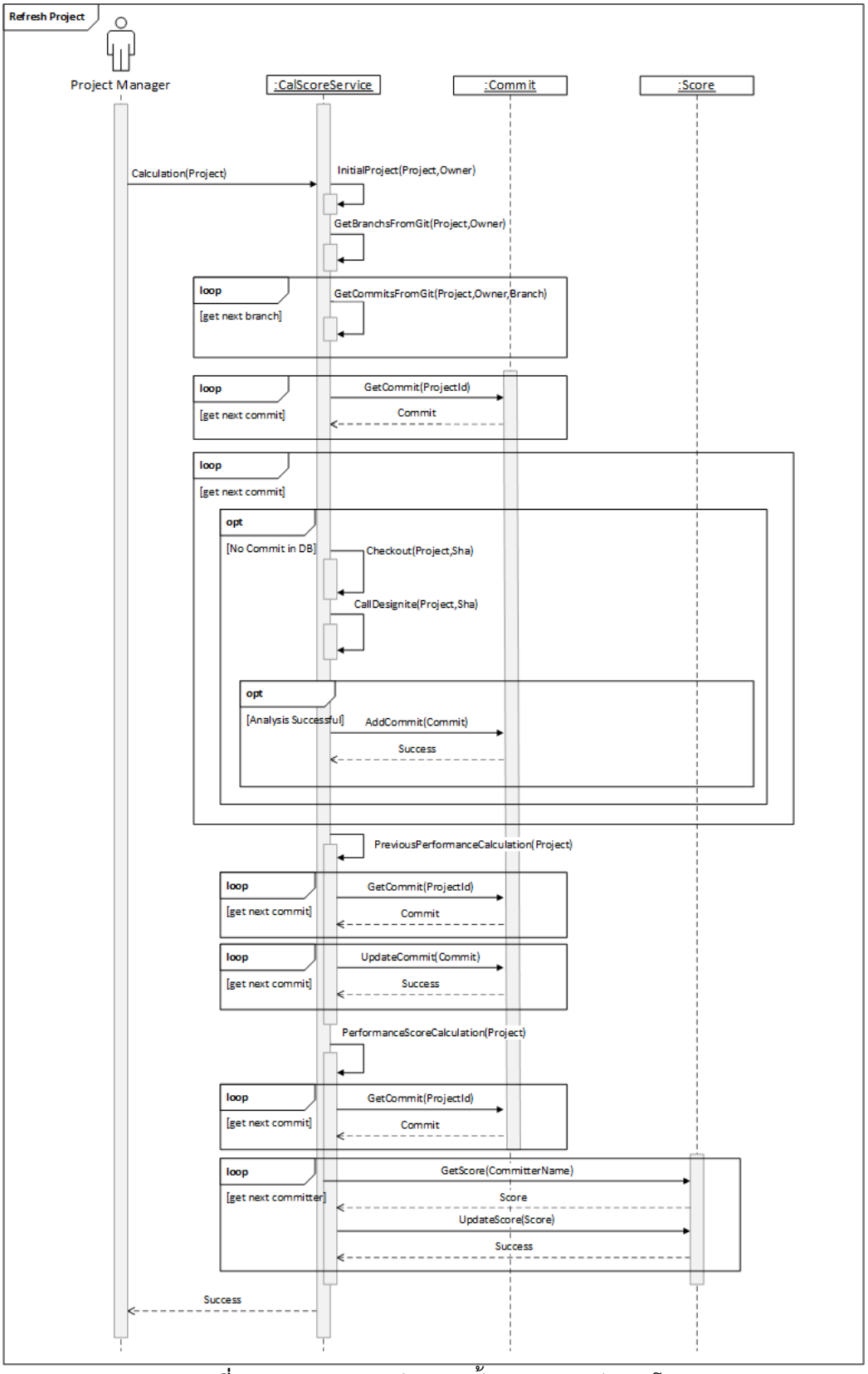
- ผู้ใช้ (ผู้จัดการโครงการ) ทำการกรอกชื่อโครงการและเจ้าของโครงการเข้าสู่ระบบ
- คลาส Project จะรับข้อมูลของโครงการมาบันทึกลงในฐานข้อมูล
- คลาส CalScoreService จะรับข้อมูลของโครงการมาคำนวณหาคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์แต่ละรายในโครงการ โดยจะมีการติดต่อกับคลาส Commit และ Score เพื่อทำการเพิ่ม แก้ไข และดึงข้อมูลมาใช้ในการคำนวณ
- คลาส Commit จะถูกเรียกจาก CalScoreService เพื่อทำการเพิ่ม แก้ไข และดึงข้อมูลมาใช้ในการคำนวณ
- คลาส Score จะถูกเรียกจาก CalScoreService เพื่อทำการเพิ่ม แก้ไข และดึงข้อมูลมาใช้ในการคำนวณ



รูปที่ 4.4 แผนภาพลำดับแสดงขั้นตอนการเพิ่มโครงการ

จากรูปที่ 4.5 แผนภาพลำดับแสดงขั้นตอนการปรับปรุงโครงการมีลำดับการทำงานดังนี้

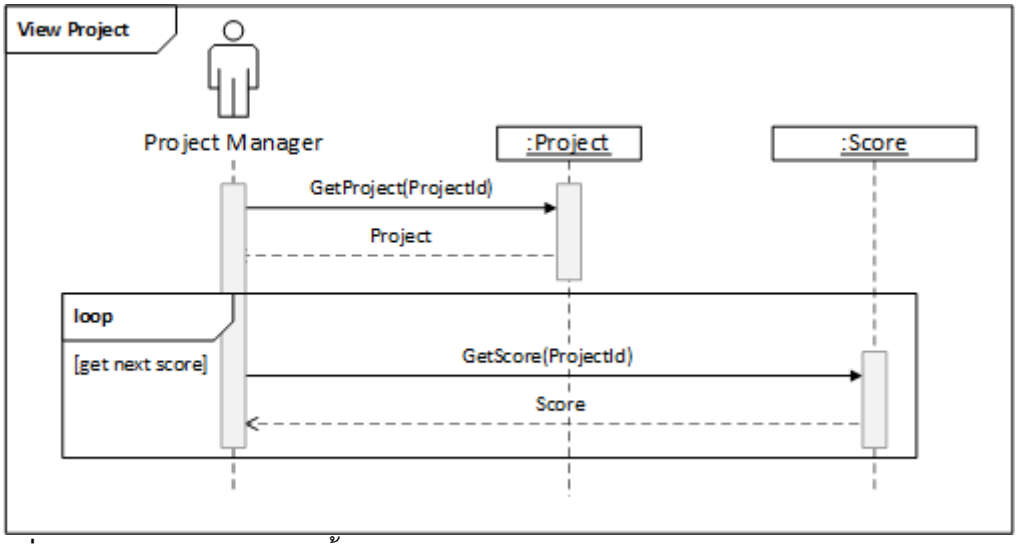
- ผู้ใช้ (ผู้จัดการโครงการ) ทำการเลือกโครงการที่ต้องการปรับปรุงในระบบ
- คลาส CalScoreService จะรับข้อมูลของโครงการมาคำนวณหาคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์แต่ละรายในโครงการ โดยจะมีการติดต่อกับคลาส Commit และ Score เพื่อทำการเพิ่ม แก้ไข และดึงข้อมูลมาใช้ในการคำนวณ
- คลาส Commit จะถูกเรียกจาก CalScoreService เพื่อทำการเพิ่ม แก้ไข และดึงข้อมูลมาใช้ในการคำนวณ
- คลาส Score จะถูกเรียกจาก CalScoreService เพื่อทำการเพิ่ม แก้ไข และดึงข้อมูลมาใช้ในการคำนวณ



รูปที่ 4.5 แผนภาพลำดับแสดงขั้นตอนการปรับปรุงโครงการ

จากรูปที่ 4.6 แผนภาพลำดับแสดงขั้นตอนการแสดงคะแนนของนักพัฒนาซอฟต์แวร์ในโครงการ มีลำดับการทำงานดังนี้

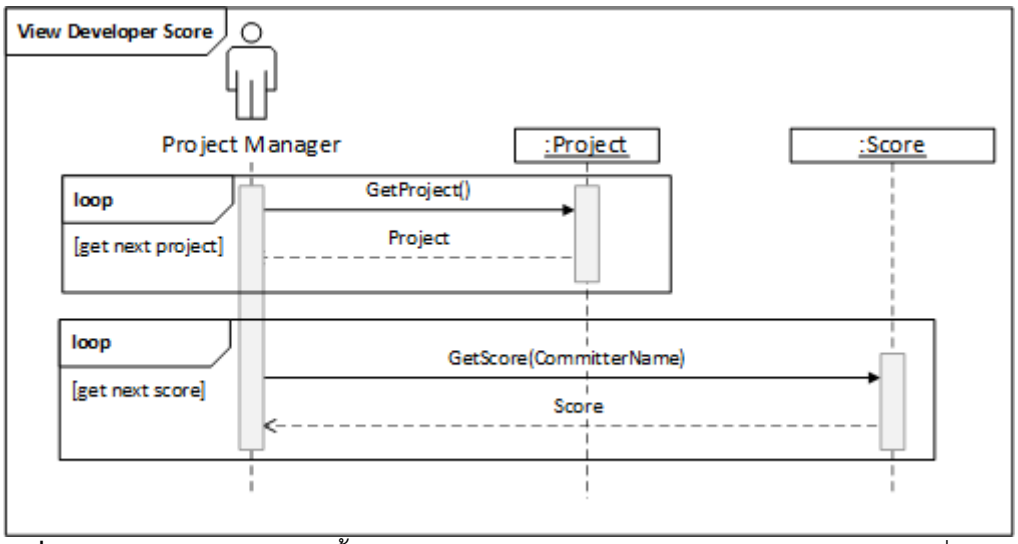
- ผู้ใช้ (ผู้จัดการโครงการ) ทำการเลือกโครงการที่ต้องการดูคะแนนของนักพัฒนาซอฟต์แวร์ในโครงการ
- คลาส Project จะถูกเรียกเพื่อดึงข้อมูลโครงการไปแสดงในระบบ
- คลาส Score จะถูกเรียกเพื่อดึงข้อมูลคะแนนของนักพัฒนาซอฟต์แวร์ในโครงการไปแสดงในระบบ



รูปที่ 4.6 แผนภาพลำดับแสดงขั้นตอนการแสดงคะแนนของนักพัฒนาซอฟต์แวร์ในโครงการ

จากรูปที่ 4.7 แผนภาพลำดับแสดงขั้นตอนการแสดงคะแนนของนักพัฒนาซอฟต์แวร์หนึ่งในทุกโครงการที่นักพัฒนาซอฟต์แวร์เข้าร่วมมีลำดับการทำงานดังนี้

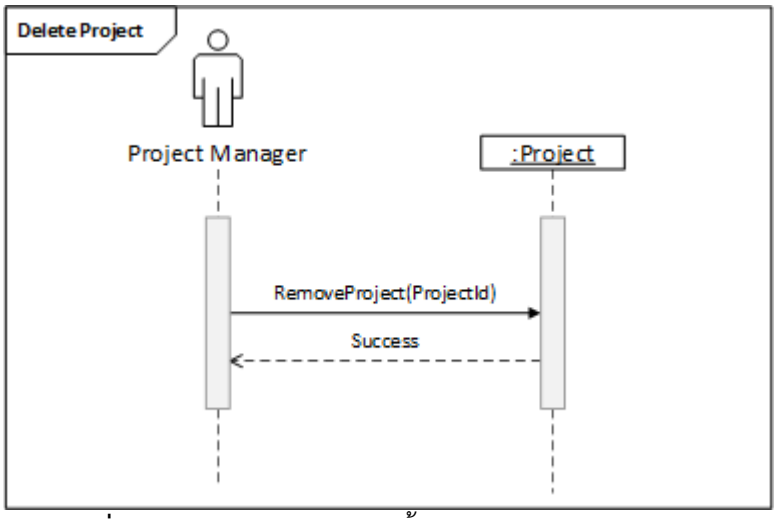
- ผู้ใช้ (ผู้จัดการโครงการ) ทำการเลือกนักพัฒนาซอฟต์แวร์ที่ต้องการเพื่อดูคะแนนทั้งหมดทุกโครงการที่นักพัฒนาคนนี้เกี่ยวข้อง
- คลาส Project จะถูกเรียกเพื่อดึงข้อมูลโครงการทั้งหมดเพื่อนำข้อมูลไปใช้เป็นพารามิเตอร์สำหรับดึงข้อมูลคะแนนทั้งหมดทุกโครงการที่นักพัฒนาคนนี้เกี่ยวข้อง
- คลาส Score จะถูกเรียกเพื่อดึงข้อมูลคะแนนทั้งหมดทุกโครงการที่นักพัฒนาคนนี้เกี่ยวข้องไปแสดงในระบบ



รูปที่ 4.7 แผนภาพลำดับแสดงขั้นตอนการแสดงคะแนนของนักพัฒนาซอฟต์แวร์รายหนึ่งในทุกโครงการที่นักพัฒนาซอฟต์แวร์เข้าร่วม

จากรูปที่ 4.8 แผนภาพลำดับแสดงขั้นตอนการลบโครงการมีลำดับการทำงานดังนี้

- ผู้ใช้ (ผู้จัดการโครงการ) ทำการเลือกโครงการที่ต้องการลบ
- คลาส Project จะถูกเรียกเพื่อลบโครงการที่ถูกเลือกออกจากระบบ



รูปที่ 4.8 แผนภาพลำดับแสดงขั้นตอนการลบโครงการ

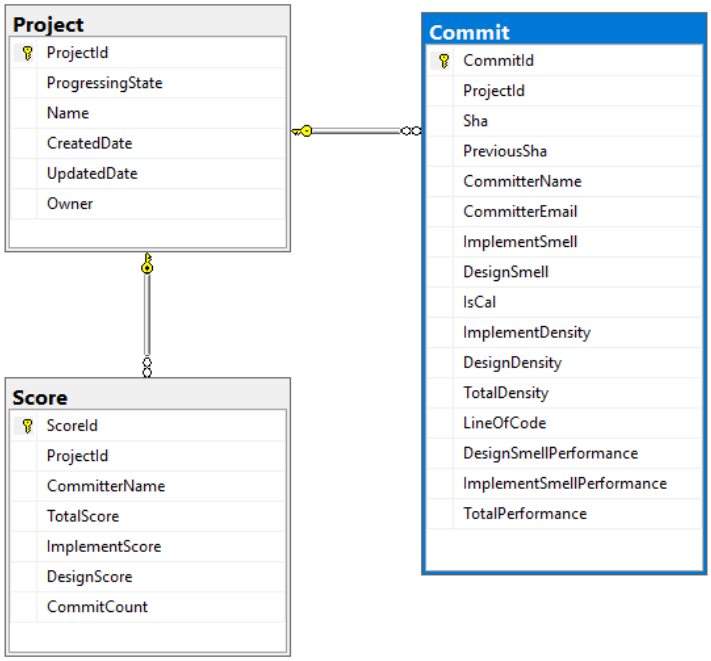
4.2.4. การออกแบบฐานข้อมูล

จากรูปที่ 4.9 แสดงให้เห็นถึงความสัมพันธ์ของแต่ละตารางในฐานข้อมูลโดยใช้แผนภาพฐานข้อมูล และได้แสดงรายละเอียดโครงสร้างของแต่ละตารางรวมถึงประเภทของข้อมูลที่จัดเก็บโดยแต่ละตารางมีรายละเอียดดังนี้

1) ตาราง Project จะทำการเก็บข้อมูลของโครงการที่จะนำมาวัดประสิทธิภาพ

2) ตาราง Commit จะทำการเก็บข้อมูลของคอมมิตทั้งหมดของโครงการที่จะนำมาวัดประสิทธิภาพ รวมถึงคะแนนของร่องรอยที่ไม่ดีในโค้ดที่จะนำไปคำนวณเป็นคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์

3) ตาราง Score จะทำการเก็บข้อมูลของคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์แต่ละรายที่ได้จากการคำนวณโดยแยกแต่ละโครงการ



รูปที่ 4.9 แผนภาพความสัมพันธ์ของแต่ละตารางในฐานข้อมูล

4.3 เครื่องมือที่ใช้ในการพัฒนาระบบ

4.3.1. ฮาร์ดแวร์ที่ใช้ในการพัฒนาระบบ

1) เครื่องคอมพิวเตอร์ที่ใช้ในการพัฒนาระบบ

- หน่วยประมวลผล อินเทล คอร์ไอ 7 ความเร็ว 3.4 กิกะเฮิร์ทซ์
- หน่วยความจำ ดีดีอาร์ 4 ขนาด 8 กิกะไบต์

- ฮาร์ดดิสก์ ความจุ 1 เทราไบต์

4.3.2. ซอฟต์แวร์ที่ใช้ในการพัฒนาระบบ

1) ระบบปฏิบัติการ

- ระบบปฏิบัติการไมโครซอฟท์วินโดวส์ 10 (Microsoft Windows 10)

2) เครื่องมือที่ใช้ในการออกแบบและจัดทำเอกสารของกระบวนการ

- ไมโครซอฟท์ออฟฟิซ รุ่น 2013 (Microsoft Office 2013)
- ไมโครซอฟท์วิสิโอ รุ่น 2013 (Microsoft Visio 2013)

3) เครื่องมือที่ใช้ในการพัฒนาเครื่องมือ

- ไมโครซอฟท์วิชวลสตูดิโอ รุ่น 2015 (Microsoft Visual Studio 2015)

4.4 ขั้นตอนการพัฒนาระบบ

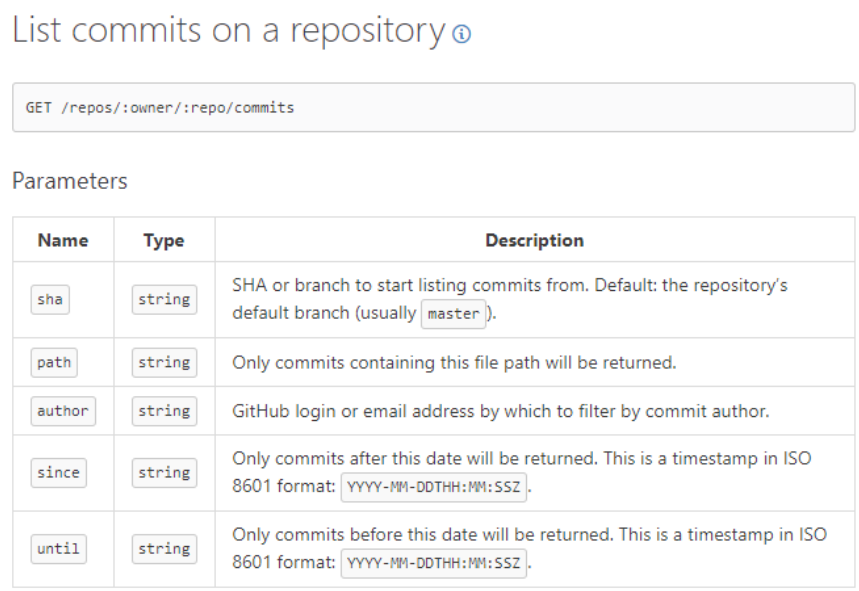
ในส่วนนี้จะกล่าวถึงขั้นตอนการพัฒนาส่วนของเครื่องมือวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์แต่ละรายโดยอิงร่องรอยที่ไม่ดีในโค้ดในระบบควบคุมเวอร์ชันแบบกระจายศูนย์ซึ่งมีรายละเอียดดังนี้

4.4.1. การจัดเตรียมฐานข้อมูลเพื่อใช้ในการจัดเก็บข้อมูลของโครงการที่ทำการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์

ในส่วนจัดเตรียมฐานข้อมูลเพื่อใช้ในการจัดเก็บข้อมูลของโครงการที่ทำการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์นั้น จะอยู่ในรูปแบบของฐานข้อมูล เอสคิวแอล เซิร์ฟเวอร์ 2017 (SQL Server 2017)

4.4.2. การเรียกใช้งานเอพีไอและรูปแบบของข้อมูลจากเว็บไซต์ Github

ในส่วนของการเรียกใช้งานเอพีไอและรูปแบบของข้อมูลจากเว็บไซต์ Github นั้น ทางเว็บไซต์ Github ได้จัดเตรียมเอพีไอสำหรับให้นักพัฒนา สามารถเรียกใช้ข้อมูลต่าง ๆ ภายในเว็บไซต์ Github ได้ เช่น ดึงข้อมูลของโครงการ รายการคอมมิตภายในโครงการ รวมถึงความคิดเห็นต่าง ๆ ภายในโครงการ เป็นต้น ตัวอย่างของเอพีไอบางส่วนจากเว็บไซต์ Github แสดงดังรูปที่ 4.10



List commits on a repository

GET /repos/:owner/:repo/commits

Parameters

Name	Type	Description
sha	string	SHA or branch to start listing commits from. Default: the repository's default branch (usually master).
path	string	Only commits containing this file path will be returned.
author	string	GitHub login or email address by which to filter by commit author.
since	string	Only commits after this date will be returned. This is a timestamp in ISO 8601 format: YYYY-MM-DDTHH:MM:SSZ.
until	string	Only commits before this date will be returned. This is a timestamp in ISO 8601 format: YYYY-MM-DDTHH:MM:SSZ.

รูปที่ 4.10 เอพีไอของเว็บไซต์ Github

โดยหลังจากเรียกใช้งานเอพีไอของเว็บไซต์ Github แล้ว ข้อมูลที่จะได้รับกลับมาจะอยู่ในรูปแบบเจสัน (Json) จากนั้นจะนำข้อมูลไปวิเคราะห์ผ่านเครื่องมือของโครงการนี้

4.4.3. การเรียกใช้ด้วยบรรทัดคำสั่ง (Command line)

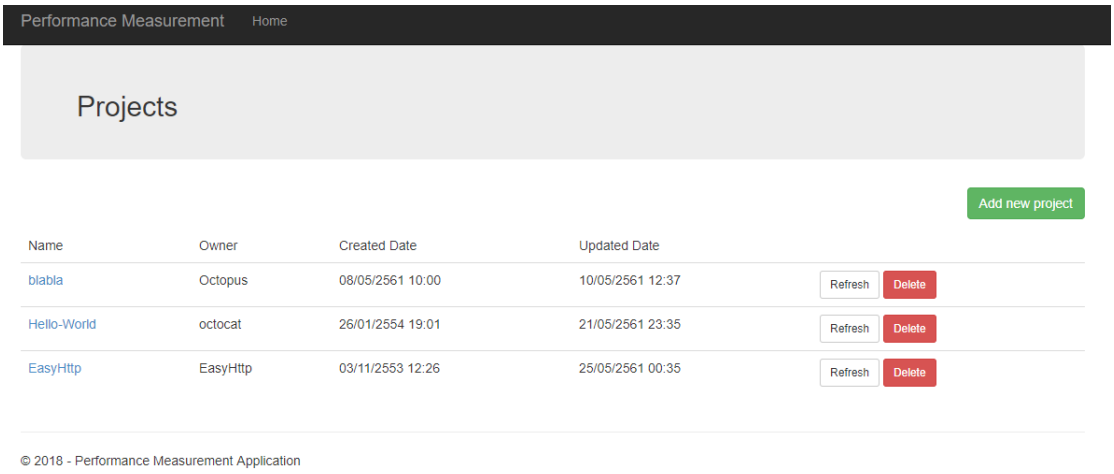
ในส่วนของการเรียกใช้ด้วยบรรทัดคำสั่งนั้น มีการใช้งานอยู่ 2 ส่วน ได้แก่ ส่วนของการดึงโค้ดจาก Github เพื่อนำมาเตรียมไว้สำหรับขั้นตอนการตรวจหาร่องรอยที่ไม่ดีในโค้ด และส่วนของการเรียก Designite เพื่อทำการตรวจหาร่องรอยที่ไม่ดีในโค้ดที่เตรียมไว้ โดยทั้ง 2 ส่วนนี้อยู่ในขั้นตอนของการวัดค่าประสิทธิภาพของนักพัฒนาซอฟต์แวร์ ตัวอย่างการเรียกใช้ด้วยบรรทัดคำสั่งดังรูปที่ 4.11

รูปที่ 4.11 การเรียกใช้ด้วยบรรทัดคำสั่ง

4.5 การพัฒนาส่วนต่อประสานผู้ใช้ของเครื่องมือ

การพัฒนาส่วนต่อประสานผู้ใช้ของเครื่องมือการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์โดยอิงร่องรอยที่ไม่ดีในโค้ดในระบบควบคุมเวอร์ชันแบบกระจายศูนย์นั้นพัฒนาขึ้นในรูปแบบเว็บแอปพลิเคชันซึ่งประกอบด้วย 4 ส่วนดังต่อไปนี้

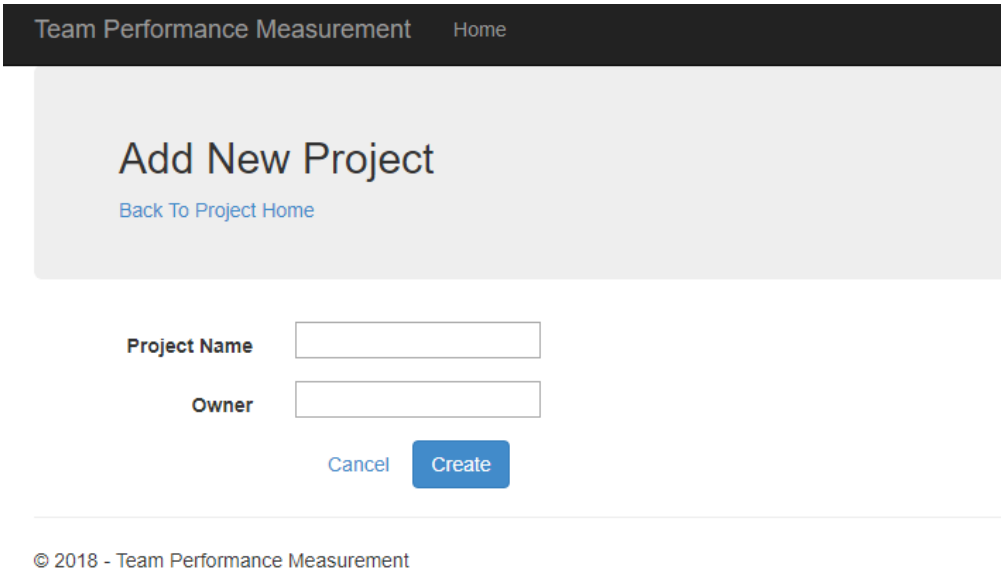
1) ส่วนของการจัดการโครงการที่จะนำมาวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์สำหรับทำการลบ/ปรับปรุงข้อมูลโครงการเป็นตามรูปที่ 4.12



รูปที่ 4.12 การออกแบบส่วนตัวประสานของส่วนการจัดการโครงการที่จะนำมาวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์

จากรูปที่ 4.12 การออกแบบส่วนตัวประสานของส่วนการจัดการโครงการที่จะนำมาวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์มีการนำโครงการในระบบมาแสดงซึ่งประกอบด้วย ชื่อโครงการ เจ้าของโครงการ วันที่สร้างโครงการ วันล่าสุดในการปรับปรุงโครงการ และยังมีส่วนของการจัดการแต่ละโครงการซึ่งได้แก่ การปรับปรุงโครงการ และการลบโครงการ ซึ่งนอกจากนี้ยังสามารถเพิ่มโครงการใหม่ที่ต้องการจะนำไปวัดประสิทธิภาพได้

2) ส่วนของการเพิ่มโครงการที่จะนำมาวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์ ทำหน้าที่เพิ่มข้อมูลโครงการตามรูปที่ 4.13

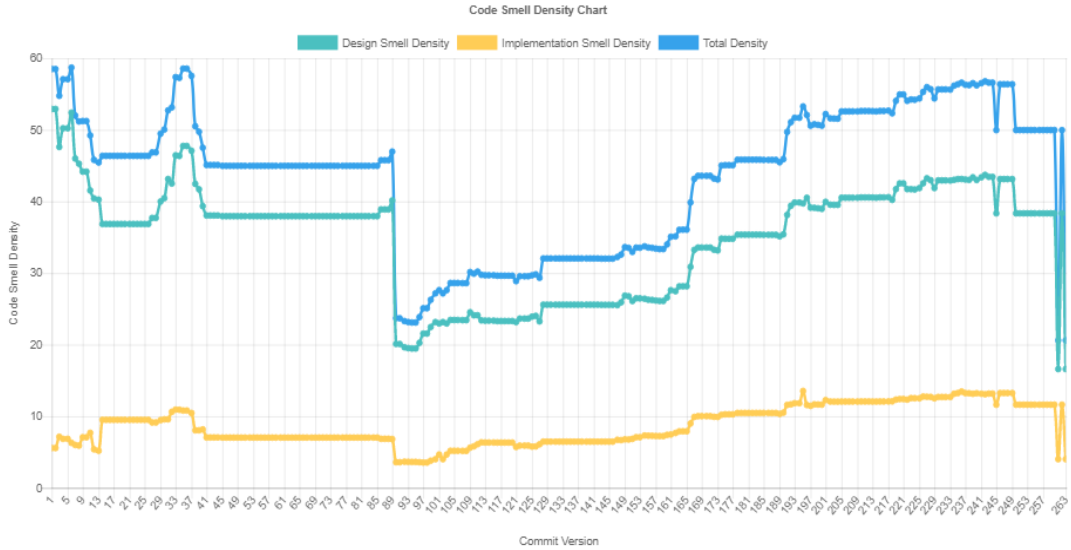


รูปที่ 4.13 การออกแบบส่วนตัวประสานของส่วนการเพิ่มโครงการที่จะนำมาวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์

จากรูปที่ 4.13 การออกแบบส่วนต่อประสานของส่วนการเพิ่มโครงการที่จะนำมาวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์มีส่วนของการกรอกข้อมูล ได้แก่ ชื่อโครงการ และเจ้าของโครงการ รวมถึงยังมีปุ่มสำหรับเพิ่มโครงการและยกเลิกการเพิ่มโครงการอีกด้วย

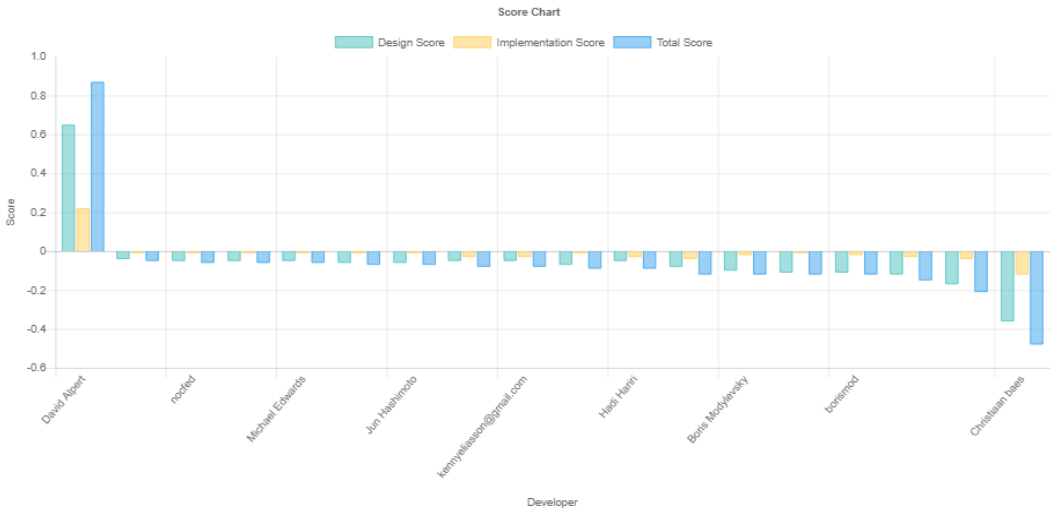
3) ส่วนของการแสดงผลของการวัดประสิทธิภาพของนักพัฒนาแต่ละรายในโครงการนั้น ๆ แบ่งเป็น 4 ส่วน ดังนี้

(1) แผนภูมิความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดในแต่ละคอมมิต โดยแกน X จะแสดงถึงเวอร์ชันคอมมิต และแกน Y จะแสดงถึงความหนาแน่นของร่องรอยที่ไม่ดีในโค้ด โดยจะแบ่งร่องรอยที่ไม่ดีในโค้ดออกเป็น 3 ประเภท ได้แก่ ร่องรอยที่ไม่ดีประเภทการพัฒนา ร่องรอยที่ไม่ดีประเภทการออกแบบ และผลรวมของร่องรอยที่ไม่ดีทั้งสองประเภท ตามรูปที่ 4.14



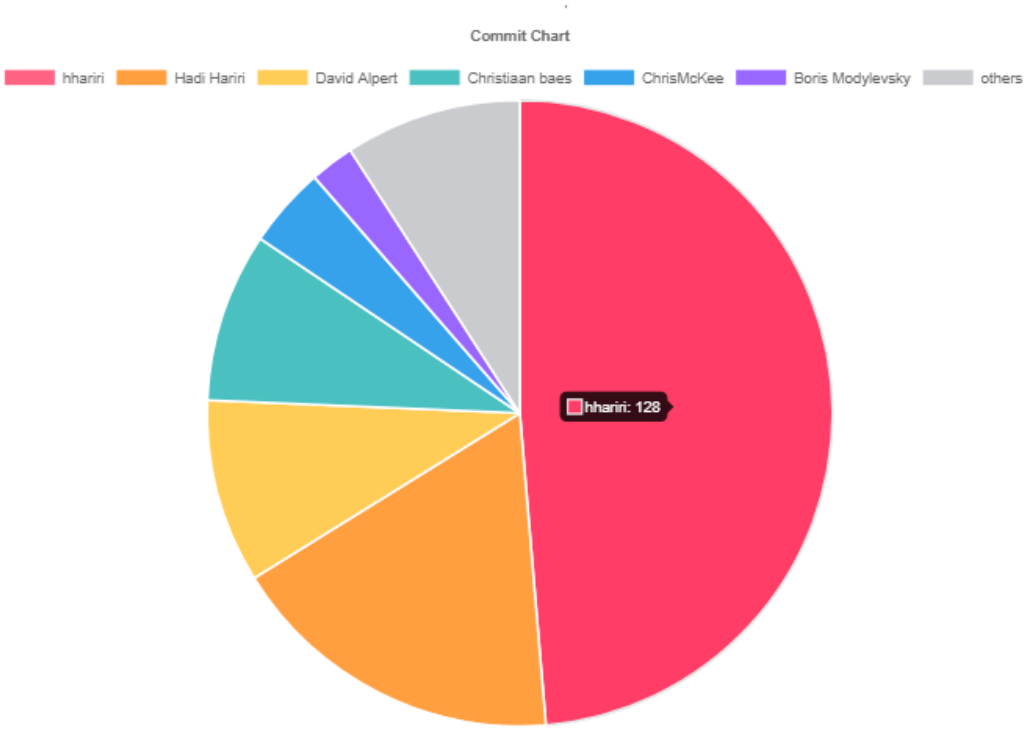
รูปที่ 4.14 การออกแบบส่วนตัวประสานของการแสดงแผนภูมิความหนาแน่นของร่องรอยที่ไม่ดีในโค้ดในแต่ละคอมมิต

(2) แผนภูมิคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์แต่ละรายในโครงการ โดยแกน X แสดงถึงชื่อของนักพัฒนาซอฟต์แวร์ และแกน Y แสดงถึงคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์ โดยจะแบ่งคะแนนประสิทธิภาพที่คำนวณจากร่องรอยที่ไม่ดีในโค้ดออกเป็น 3 ประเภท ได้แก่ ร่องรอยที่ไม่ดีประเภทการพัฒนา ร่องรอยที่ไม่ดีประเภทการออกแบบ และผลรวมของร่องรอยที่ไม่ดีทั้งสองประเภท ซึ่งเป็นค่าประสิทธิภาพโดยรวมของนักพัฒนาซอฟต์แวร์แต่ละรายโดยคำนวณจากคอมมิต โค้ดในเวอร์ชันปัจจุบัน ตามสมการที่ 6 ซึ่งจะแสดงตามรูปที่ 4.15



รูปที่ 4.15 การออกแบบส่วนตัวประสานของการแสดงแผนภูมิคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์แต่ละรายในโครงการ

(3) แผนภูมิจำนวนคอมมิตของนักพัฒนาซอฟต์แวร์ 6 อันดับที่มีการคอมมิตมากที่สุด และผลรวมของการคอมมิตของอันดับถัดไป ตามรูปที่ 4.16



รูปที่ 4.16 แผนภูมิจำนวนคอมมิตของนักพัฒนาซอฟต์แวร์

(4) ตารางคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์แต่ละราย โดยจะประกอบด้วย ชื่อของนักพัฒนา จำนวนคอมมิต คะแนนเฉลี่ยของร่องรอยที่ไม่ดีในโค้ดประเภทการพัฒนา คะแนนเฉลี่ยของร่องรอยที่ไม่ดีในโค้ดประเภทการออกแบบ และคะแนนเฉลี่ยของร่องรอยที่ไม่ดีในโค้ดรวมทั้งสองประเภท ตามรูปที่ 4.17

Committer	Number of commits	Performance Score (Design Smell)	Performance Score (Implementation Smell)	Performance Score (Total)
David Alpert	25	0.65	0.22	0.87
ChrisMcKee	11	-0.04	-0.01	-0.05
nocfed	2	-0.05	-0.01	-0.06
GitHub	4	-0.05	-0.01	-0.06
Michael Edwards	1	-0.05	-0.01	-0.06
Aaron Kor	1	-0.06	-0.01	-0.07
Jun Hashimoto	1	-0.06	-0.01	-0.07
Minglei Hou	2	-0.05	-0.03	-0.08
kennyeliasson@gmail.com	2	-0.05	-0.03	-0.08
Peter T. LaComb Jr	1	-0.07	-0.01	-0.09
Hadi Hariri	46	-0.05	-0.03	-0.09
Johan Leino	3	-0.08	-0.04	-0.12
Boris Modylevsky	6	-0.10	-0.02	-0.12

รูปที่ 4.17 ตารางคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์แต่ละราย

4) ส่วนของการแสดงผลของการวัดประสิทธิภาพจากทุกโครงการที่นักพัฒนาซอฟต์แวร์รายนั้น ๆ มีส่วนร่วมอยู่ในโครงการ โดยจะประกอบไปด้วยส่วนของแผนภูมิการเปรียบเทียบคะแนนของโครงการต่าง ๆ โดยแกน X แสดงถึงโครงการที่นำมาเปรียบเทียบ และแกน Y แสดงถึงคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์ โดยจะแบ่งคะแนนประสิทธิภาพที่คำนวณจากร่องรอยที่ไม่ดีในโค้ดออกเป็น 3 ประเภท ได้แก่ ร่องรอยที่ไม่ดีประเภทการพัฒนา ร่องรอยที่ไม่ดีประเภทการออกแบบ และผลรวมของร่องรอยที่ไม่ดีทั้งสองประเภท และส่วนของตารางคะแนนของทุกโครงการที่นักพัฒนาซอฟต์แวร์รายนั้น ๆ เข้าร่วม ตามรูปที่ 4.18

Project	Number of commit	Performance Score (Design Smell)	Performance Score (Implement Smell)	Performance Score (Total)	Compare
EasyHttp	25	0.65	0.22	0.87	☑
ExampleProject	8	0.83	-0.49	0.34	☑

รูปที่ 4.18 การออกแบบส่วนตัวประสานของการแสดงผลของการวัดประสิทธิภาพจากทุกโครงการที่นักพัฒนาซอฟต์แวร์รายนั้น ๆ มีส่วนร่วมอยู่ในโครงการ

จากรูปที่ 4.18 การออกแบบส่วนต่อประสานของการแสดงผลของการวัดประสิทธิภาพจากทุกโครงการที่นักพัฒนาซอฟต์แวร์รายนั้น ๆ มีส่วนร่วมอยู่ในโครงการมีส่วนของแผนภูมิการเปรียบเทียบคะแนนของโครงการต่าง ๆ และส่วนของตารางคะแนนของทุกโครงการที่นักพัฒนาซอฟต์แวร์รายนั้น ๆ เข้าร่วม โดยจะประกอบด้วย ชื่อโครงการ จำนวนคอมมิต คะแนนเฉลี่ยของร่องรอยที่ไม่ดีในโค้ดประเภทการพัฒนา คะแนนเฉลี่ยของร่องรอยที่ไม่ดีในโค้ดประเภทการออกแบบ และคะแนนเฉลี่ยของร่องรอยที่ไม่ดีในโค้ดรวมทั้งสองประเภท และยังสามารถเลือกเปรียบเทียบโครงการได้โดยการทำเครื่องหมายถูกในส่วนของการเปรียบเทียบ ซึ่งระบบจะนำคะแนนของโครงการไปเพิ่มในแผนภูมิการเปรียบเทียบคะแนนของโครงการต่าง ๆ ทั้งนี้ในการเปรียบเทียบผู้ใช้เครื่องมือจะต้องคำนึงถึงสภาพแวดล้อม ความซับซ้อน ขนาดของโครงการ และผู้ร่วมโครงการ ซึ่งควรจะเป็นแบบเดียวกันหรือใกล้เคียงกันจึงจะสามารถเปรียบเทียบกันได้ เนื่องจากปัจจัยเหล่านี้มีผลต่อคะแนนประสิทธิภาพของนักพัฒนาในโครงการนั้น ๆ ทั้งสิ้น โดยถ้าผู้ใช้เครื่องมือไม่คำนึงถึงความแตกต่างระหว่างปัจจัยเหล่านี้อาจทำให้การตีความผลการเปรียบเทียบระหว่างโครงการที่เลือกมาผิดพลาดได้

หากโครงการที่เลือกมาเปรียบเทียบกันได้แล้ว ถ้าคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์รายนี้ในโครงการใหม่มีค่าน้อยกว่าคะแนนในโครงการเก่า หมายความว่านักพัฒนาซอฟต์แวร์รายนี้มีการพัฒนาซอฟต์แวร์ที่มีคุณภาพลดลง ในทางตรงกันข้าม ถ้าคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์รายนี้ในโครงการใหม่มีค่ามากกว่าคะแนนในโครงการเก่า จะหมายความว่านักพัฒนาซอฟต์แวร์รายนี้มีการพัฒนาซอฟต์แวร์ที่มีคุณภาพมากขึ้น

บทที่ 5
การทดสอบและประเมินผล

การประเมินผลและทดสอบการทำงานของเครื่องมือวัดประสิทธิภาพนักพัฒนาซอฟต์แวร์รายคนโดยอิงร่องรอยที่ไม่ดีในโค้ดในระบบควบคุมเวอร์ชันแบบกระจายศูนย์มีดังนี้

5.1 การประเมินผลความสอดคล้องในการจัดอันดับประสิทธิภาพของนักพัฒนาซอฟต์แวร์

ในส่วนนี้จะใช้การเปรียบเทียบระหว่าง ผลลัพธ์จากการทดลองจัดอันดับตามคะแนนการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์โดยใช้เครื่องมือที่นำเสนอกับผลลัพธ์การจัดอันดับประสิทธิภาพของนักพัฒนาซอฟต์แวร์ที่ได้จากการตอบแบบสอบถามโดยผู้ประเมินที่เป็นนักพัฒนาซอฟต์แวร์ที่มีประสบการณ์จำนวน 4 คน ดังข้อมูลในตารางที่ 5.1 ผู้ประเมินแต่ละคนจะต้องตรวจสอบโค้ดของโครงการจำนวน 2 โครงการ ดังตารางที่ 5.2 และนำข้อมูลจากการตรวจสอบมาทำแบบสอบถามประสิทธิภาพของนักพัฒนาซอฟต์แวร์จำนวน 10 ข้อ แต่ละข้อมีคะแนน 1-5 ซึ่งแสดงถึงประสิทธิภาพของนักพัฒนาซอฟต์แวร์ในระดับต้องปรับปรุง-ระดับดีมาก จากนั้นทำการรวมคะแนนที่นักพัฒนาซอฟต์แวร์แต่ละคนได้แล้วนำมาจัดอันดับประสิทธิภาพของนักพัฒนาซอฟต์แวร์ในโครงการ (ดูรายละเอียดแบบสอบถามในภาคผนวก ก.)

ตารางที่ 5.1 ข้อมูลรายละเอียดของผู้ประเมินแบบสอบถาม

	ผู้ประเมินคนที่ 1	ผู้ประเมินคนที่ 2	ผู้ประเมินคนที่ 3	ผู้ประเมินคนที่ 4
ตำแหน่งงาน	Senior Programmer	Senior Programmer	Senior Programmer	Senior Programmer
ประสบการณ์ทำงาน	6 ปี	6 ปี	6 ปี	2 ปี

ตารางที่ 5.2 ข้อมูลรายละเอียดของโครงการที่นำมาประเมิน

	โครงการที่ 1	โครงการที่ 2
จำนวนคอมมิต	292 คอมมิต	82 คอมมิต
จำนวรบรรทัดของโค้ด	5482 บรรทัด	3396 บรรทัด
จำนวนผู้คอมมิต	18 คน	7 คน

การวัดผลความแม่นในการจัดอันดับประสิทธิภาพของนักพัฒนาซอฟต์แวร์จะใช้มาตรวัดสัมประสิทธิ์สหสัมพันธ์อันดับที่ของสเปียร์แมน (Spearman's Rank Correlation Coefficient) ซึ่งเป็นค่าดัชนีการวัดความสัมพันธ์ระหว่างการจัดอันดับสองชุดบนข้อมูลชุด

เดียวกันโดยจะวัดครั้งละคู่อันดับ ค่าจะอยู่ในช่วง -1 ถึง 1 โดย -1 หมายถึงการจัดอันดับสองชุดมีอันดับที่ตรงข้ามกัน และ 1 หมายถึงการจัดอันดับสองชุดมีลำดับที่ตรงกัน ในการจัดอันดับประสิทธิภาพของนักพัฒนาซอฟต์แวร์แต่ละรายในโครงการ i ค่าดัชนี ρ จะคำนวณได้จากสมการ (10)

$$\rho_i = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}} \quad (10)$$

โดยที่

x_i คือ อันดับที่ของคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์จากเครื่องมือ

y_i คือ อันดับที่ของคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์จากผู้ประเมิน

5.2 ผลการทดลอง

ผลการประเมินโดยเครื่องมือและผู้ประเมินทั้ง 4 ราย สำหรับโครงการที่ 1 และ 2 แสดงในตารางที่ 5.3 และ 5.4 ตามลำดับ ส่วนค่าสัมประสิทธิ์สหสัมพันธ์ที่วัดได้แสดงไว้ในตารางที่ 5.5

ตารางที่ 5.3 ผลการประเมินของโครงการที่ 1 (ดูรายละเอียดคะแนนการประเมินได้ในภาคผนวก ข)

โครงการที่ 1	เครื่องมือ		ผู้ประเมิน 1		ผู้ประเมิน 2		ผู้ประเมิน 3		ผู้ประเมิน 4	
นักพัฒนาซอฟต์แวร์	คะแนน	อันดับ	คะแนน	อันดับ	คะแนน	อันดับ	คะแนน	อันดับ	คะแนน	อันดับ
1	0.87	1	37	1	41	1	40	1	38	1
2	-0.0476	2	35	2	33	2	33	2	29	12
3	-0.0574	3	30	7	31	3	29	13.5	30	7
4	-0.0606	4	30	7	30	7	31	3.5	32	2
5	-0.0611	5	30	7	30	7	31	3.5	31	3
6	-0.0728	6.5	30	7	30	7	30	8	30	7
7	-0.0728	6.5	30	7	30	7	30	8	30	7
8	-0.0776	8	30	7	30	7	30	8	30	7
9	-0.0796	9	30	7	30	7	30	8	30	7
10	-0.0857	10	30	7	30	7	30	8	30	7
11	-0.0873	11	30	7	28	16	30	8	30	7
12	-0.1162	12	28	14	29	13	28	16.5	28	14

ตารางที่ 5.3 ผลการประเมินของโครงการที่ 1 (ดูรายละเอียดคะแนนการประเมินได้ในภาคผนวก ข) [ต่อ]

โครงการที่ 1	เครื่องมือ		ผู้ประเมิน 1		ผู้ประเมิน 2		ผู้ประเมิน 3		ผู้ประเมิน 4	
นักพัฒนาซอฟต์แวร์	คะแนน	อันดับ	คะแนน	อันดับ	คะแนน	อันดับ	คะแนน	อันดับ	คะแนน	อันดับ
13	-0.1172	13	28	14	29	13	29	13.5	29	12
14	-0.1232	14	28	14	29	13	29	13.5	29	12
15	-0.1236	15	27	17	27	17	28	16.5	27	15.5
16	-0.154	16	28	14	29	13	30	8	27	15.5
17	-0.2126	17	28	14	29	13	29	13.5	26	17
18	-0.4783	18	23	18	24	18	24	18	21	18

ตารางที่ 5.4 ผลการประเมินของโครงการที่ 2 (ดูรายละเอียดคะแนนการประเมินได้ในภาคผนวก ข)

โครงการที่ 2	เครื่องมือ		ผู้ประเมิน 1		ผู้ประเมิน 2		ผู้ประเมิน 3		ผู้ประเมิน 4	
นักพัฒนาซอฟต์แวร์	คะแนน	อันดับ	คะแนน	อันดับ	คะแนน	อันดับ	คะแนน	อันดับ	คะแนน	อันดับ
1	0.0149	1	38	1	37	1	37	1	37	1
2	0.0133	2	32	2.5	32	2.5	31	5	32	2.5
3	0.0099	4	30	7	31	5.5	31	5	31	5.5
4	0.0099	4	31	5	31	5.5	31	5	31	5.5
5	0.0099	4	31	5	31	5.5	31	5	31	5.5
6	0.0073	6	31	5	31	5.5	31	5	31	5.5
7	-0.0066	7	32	2.5	32	2.5	35	2	32	2.5

ตารางที่ 5.5 สัมประสิทธิ์สหสัมพันธ์ระหว่างการจัดอันดับโดยเครื่องมือและผู้ประเมินทั้ง 4 ราย

การจัดอันดับ	ค่า ρ ของโครงการที่ 1	ค่า ρ ของโครงการที่ 2
เครื่องมือ-ผู้ประเมิน 1	0.912	0.3689
เครื่องมือ-ผู้ประเมิน 2	0.911	0.4341
เครื่องมือ-ผู้ประเมิน 3	0.7538	0.1387
เครื่องมือ-ผู้ประเมิน 4	0.8256	0.4341
ค่าเฉลี่ย	0.8506	0.344

จากผลการทดลองแสดงให้เห็นว่าค่าสัมประสิทธิ์สหสัมพันธ์อันดับที่ของสเปียร์แมน ρ ของทั้งสองโครงการมีค่าเป็นบวก คือ 0.8506 และ 0.344 ซึ่งแสดงว่าการจัดอันดับของสองโครงการมีความสอดคล้องในเชิงบวกกับการจัดอันดับโดยผู้ประเมินที่เป็นนักพัฒนาซอฟต์แวร์ที่มีประสบการณ์ ทั้งนี้โครงการที่ 1 มีค่า ρ มากกว่าโครงการที่ 2 เนื่องจากโครงการที่ 1 มีนักพัฒนาซอฟต์แวร์หลายรายที่มีการคอมมิตน้อยและมีการเพิ่มหรือปรับปรุงโค้ดน้อย จึงทำให้ในแบบสอบถามผู้ประเมินทำการให้ค่าปานกลางแก่นักพัฒนาซอฟต์แวร์รายนั้น และทำให้เมื่อจัดอันดับนักพัฒนากลุ่มนี้จะอยู่ในอันดับกลาง ๆ ในขณะเดียวกันการวัดประสิทธิภาพนักพัฒนากลุ่มนี้โดยเครื่องมือจะจัดอันดับให้นักพัฒนากลุ่มนี้อยู่ในอันดับกลาง ๆ เช่นกัน เนื่องจากเมื่อมีการคอมมิตน้อยและมีการเพิ่มหรือปรับปรุงโค้ดน้อย ทำให้การเปลี่ยนแปลงของความหนาแน่นของร่องรอยที่ไม่ดีมีค่าน้อยไปด้วย ค่าเฉลี่ยเบย์เซียนจึงให้ค่าที่อยู่อันดับกลาง ๆ ไม่สูงหรือต่ำมาก เมื่อทำการคำนวณสัมประสิทธิ์สหสัมพันธ์จึงได้ค่าความสอดคล้องสูงในเชิงบวก สำหรับโครงการที่ 2 แม้ค่า ρ เป็นบวกแต่ก็มีค่าไม่สูงนัก ซึ่งอาจเป็นผลมาจากการที่จำนวนคอมมิตในโครงการมีค่อนข้างมาก ผู้ประเมินอาจไม่สามารถพิจารณาจากทุกคอมมิตในโครงการได้โดยละเอียด แต่เครื่องมือสามารถวิเคราะห์และคำนวณค่าประสิทธิภาพจากทุกคอมมิตได้เลยโดยตรง ผลการประเมินประสิทธิภาพจึงอาจมีความแตกต่างกันอยู่ ทำให้การคำนวณค่า ρ จึงได้ค่าที่ไม่สูงนัก อย่างไรก็ตามจากการที่ค่าเฉลี่ย ρ ของทั้งสองโครงการมีค่าบวก จึงสรุปได้ว่าวิธีการที่ผู้วิจัยเสนอมาสามารถใช้เป็นส่วนเสริมในการพิจารณาประสิทธิภาพของนักพัฒนาซอฟต์แวร์รายคนร่วมกับการพิจารณาด้วยวิธีอื่น ๆ ได้

บทที่ 6
บทสรุปโครงงานและข้อเสนอแนะ

6.1 สรุปผลโครงงานมหาบัณฑิต

โครงงานมหาบัณฑิตนี้ได้นำเสนอวิธีการและเครื่องมือสนับสนุนการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์รายคนโดยอิงร่องรอยที่ไม่ดีในโค้ดในระบบควบคุมเวอร์ชันแบบกระจายศูนย์ โดยเครื่องมือที่ผู้วิจัยพัฒนาสามารถวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์ของโครงการที่ผู้ใช้เครื่องมือเลือกมาจากเว็บไซต์ Github โดยระบบจะเรียกใช้เครื่องมือ Designite ในการวิเคราะห์หาค่าร่องรอยที่ไม่ดีในโค้ดในแต่ละเวอร์ชัน จากนั้นนำค่าที่ไม่ดีในโค้ดแต่ละเวอร์ชันมาหาค่าความหนาแน่นของร่องรอยที่ไม่ดีในโค้ด และนำไปเปรียบเทียบกับเวอร์ชันก่อนหน้าเพื่อที่จะได้ค่าที่เปลี่ยนแปลงในแต่ละเวอร์ชัน สุดท้ายจะนำค่าที่เปลี่ยนแปลงในแต่ละเวอร์ชันมาหาค่าประสิทธิภาพของนักพัฒนาซอฟต์แวร์แต่ละรายโดยใช้สมการหาค่าเฉลี่ยเบย์เซียน ซึ่งผลการทดลองพบว่าการคำนวณคะแนนประสิทธิภาพของนักพัฒนาซอฟต์แวร์แต่ละรายโดยอิงร่องรอยที่ไม่ดีในโค้ดในระบบควบคุมเวอร์ชันแบบกระจายศูนย์กับผลการประเมินของนักพัฒนาซอฟต์แวร์ที่มีประสบการณ์มีค่าสัมประสิทธิ์สหสัมพันธ์อันดับที่ของสเปียร์แมนของทั้งสองโครงการที่นำมาทดลองมีค่าเป็นบวก คือ 0.8506 กับ 0.344 ซึ่งแสดงว่าการจัดอันดับทั้งสองวิธีมีความสอดคล้องกันพอควร ผู้จัดการโครงการสามารถนำวิธีการและเครื่องมือมาใช้เป็นส่วนเสริมในการพิจารณาประสิทธิภาพของนักพัฒนาซอฟต์แวร์รายคนร่วมกับการพิจารณาด้วยวิธีอื่น

6.2 ปัญหาและข้อจำกัดในการทำโครงงาน

(1) เครื่องมือ Designite ไม่สามารถวิเคราะห์โครงการที่มีการนำไลบรารีที่พัฒนาขึ้นมาเองมาใช้ร่วมกับในโครงการได้

(2) เครื่องมือ Designite ไม่สามารถวิเคราะห์โค้ดที่มากกว่า 50,000 บรรทัดได้ในเวอร์ชันทดลอง ซึ่งเป็นเวอร์ชันที่ใช้ในโครงงานมหาบัณฑิตนี้

(3) เครื่องมือ Designite ไม่สามารถวิเคราะห์โค้ดที่มีการใช้ไลบรารีของ C# นอกเหนือจากเวอร์ชันที่ Designite สนับสนุนได้

(4) การคอมมิตอาจจะไม่ใช่การคอมมิตโค้ดอย่างเดียว อาจเป็นการคอมมิตเพื่อนำเอกสารต่าง ๆ (Documentation) เข้าสู่โครงการ จึงทำให้การดึงข้อมูลโครงการเข้ามาวัดประสิทธิภาพจะต้องคัดกรองการคอมมิตเอกสารออกไป ซึ่งทำให้คอมมิตที่นำมาคำนวณมีจำนวนลดลง ส่งผลให้จำนวนคอมมิตที่แสดงผลโดยเครื่องมือมีจำนวนน้อยกว่าที่ปรากฏจริงบน Github

(5) ในการวัดประสิทธิภาพนักพัฒนาซอฟต์แวร์ที่สนใจรายหนึ่งในหลายโครงการในระบบควบคุมเวอร์ชันแบบกระจายศูนย์ Github ผู้วิจัยพบว่ามีความเป็นไปได้ยากเนื่องจากข้อมูลใน Github บอกเพียงว่านักพัฒนารายนี้เป็นเจ้าของโครงการใดบ้าง แต่ไม่สามารถหาข้อมูลได้ว่านักพัฒนารายนี้เกี่ยวข้องกับโครงการใดบ้าง อีกทั้งโครงการต่าง ๆ ที่นักพัฒนารายนี้เป็นเจ้าของอาจใช้ภาษาอื่นที่ไม่ใช่ซีชาร์ป หรือมีการใช้ไลบรารีที่ Designite ไม่รองรับจึงทำให้ไม่มีข้อมูลโครงการเพียงพอสำหรับคำนวณเปรียบเทียบประสิทธิภาพของนักพัฒนารายนี้แบบข้ามโครงการได้ อย่างไรก็ตามหากผู้ใช้เครื่องมือมีข้อมูลของหลายโครงการบน Github เอง และมีทีมนักพัฒนาที่ร่วมอยู่ในหลายโครงการนี้ด้วยกัน ก็จะสามารถใช้เครื่องมือในการวัดและเปรียบเทียบประสิทธิภาพของนักพัฒนารายที่สนใจแบบข้ามโครงการได้

6.3 ข้อเสนอแนะ

(1) โครงงานมหาบัณฑิตนี้ได้นำเสนอการวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์โดยอิงร่องรอยที่ไม่ดีในโค้ดในระบบควบคุมเวอร์ชันแบบกระจายศูนย์ โดยนำเครื่องมือ Designite มาเป็นส่วนหนึ่งในการหาร่องรอยที่ไม่ดีในโค้ด จึงควรมีการพัฒนาส่วนของการหาร่องรอยที่ไม่ดีในโค้ดเพื่อแก้ไขปัญหาและข้อจำกัดของ Designite ที่มีอยู่ในโครงงานมหาบัณฑิตนี้

(2) วิธีหาค่าเฉลี่ยเอโล (Elo) [13] เป็นอีกวิธีหนึ่งที่ใช้วัดประสิทธิภาพนักพัฒนาซอฟต์แวร์ เนื่องจากค่าเฉลี่ยเอโลเป็นวิธีการคำนวณค่าเฉลี่ยโดยจะคำนวณเปรียบเทียบนักพัฒนาซอฟต์แวร์เป็นคู่จนได้ค่าเฉลี่ยของแต่ละราย จึงสามารถนำวิธีหาค่าเฉลี่ยเอโลนี้ไปพัฒนาเป็นอีกวิธีในการคำนวณหาค่าประสิทธิภาพของนักพัฒนาซอฟต์แวร์ได้

(3) การหาค่าประสิทธิภาพของนักพัฒนาซอฟต์แวร์สามารถพิจารณาจากค่าคุณลักษณะที่ดีต่าง ๆ ของนักพัฒนาซอฟต์แวร์ได้เช่นกัน เช่น อาจนำค่าคุณลักษณะที่ดีในด้านการเขียนโค้ด มาพิจารณาร่วมกับปริมาณความหนาแน่นของร่องรอยที่ไม่ดีที่เสนอ ซึ่งอาจช่วยให้ผลลัพธ์ของการคำนวณมีความเหมาะสมยิ่งขึ้น

ภาคผนวก

ภาคผนวก ก.
แบบทดสอบเพื่อวัดประสิทธิภาพของนักพัฒนาซอฟต์แวร์โดยอิงร่องรอยที่ไม่ดีในโค้ดในระบบควบคุมเวอร์ชันแบบกระจายศูนย์

วิธีทำแบบสอบถาม

ผู้ทำแบบสอบถามอ่านคำถามและนำไปพิจารณาร่วมกับโค้ดของนักพัฒนาซอฟต์แวร์ที่จะนำมาวัดประสิทธิภาพ โดยคำตอบมี 5 ตัวเลือก ซึ่งแต่ละคำตอบจะมีคะแนนแตกต่างกันดังนี้

ดีมาก	5 คะแนน
ดี	4 คะแนน
ปานกลาง	3 คะแนน
พอใช้	2 คะแนน
ปรับปรุง	1 คะแนน

หากผู้ทำแบบสอบถามไม่สามารถพิจารณาคำถามบางข้อ เนื่องมาจากข้อมูลที่จะนำมาพิจารณาไม่เพียงพอ ให้ผู้ทำแบบสอบถามให้คะแนน “ปานกลาง” สำหรับคำถามข้อนั้นๆ

ข้อมูลที่ผู้ตอบแบบสอบถามต้องระบุ

1. ชื่อ-นามสกุล
2. ตำแหน่งงาน
3. ประสบการณ์ทำงาน

คำถาม

1. มีการวางโครงสร้างของโครงการให้เข้าใจได้ง่าย
2. ตั้งชื่อคลาสและฟังก์ชันให้สื่อถึงการทำงานและเข้าใจง่าย
3. แยกการทำงานของแต่ละฟังก์ชันอย่างชัดเจน
4. มีการจัดการคลาสและฟังก์ชันให้เล็กและทำความเข้าใจง่าย
5. มีการจัดกลุ่มของฟังก์ชันที่มีการทำงานที่สอดคล้องกันให้อยู่ในรูปแบบคลาส
6. มีการจัดการโค้ดที่มีการทำไปใช้ซ้ำได้อย่างเหมาะสม
7. มีการจัดการในการกำหนดตัวแปรแบบคงที่
8. มีการอธิบายโค้ดที่ต้องทำความเข้าใจ
9. มีการจัดการคำโค้ดเก่า หรือโค้ดที่ไม่ได้ใช้
10. มีการจัดการกับโค้ดที่มีความเสี่ยงที่จะเกิดข้อผิดพลาด

ภาคผนวก ข.

คำตอบของแบบสอบถาม

คำตอบจากผู้ทำแบบสอบถาม

ผู้ทำแบบสอบถามคนที่ 1

ตำแหน่งงาน: Senior Programmer

ประสบการณ์ทำงาน: 6 ปี

โครงการที่ 1

คำถาม	คะแนนของนักพัฒนาซอฟต์แวร์แต่ละราย								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	5	3	3	3	3	3	3	3	3
2	4	3	2	3	3	3	3	3	3
3	4	3	3	3	3	3	3	3	3
4	2	3	2	3	3	3	3	3	3
5	4	3	3	3	3	3	3	3	3
6	4	4	3	3	3	3	3	3	3
7	2	4	3	3	3	3	3	3	3
8	4	3	3	3	3	3	3	3	3
9	4	5	4	3	3	3	3	3	3
10	4	4	4	3	3	3	3	3	3
คะแนนรวม	37	35	30	30	30	30	30	30	30

คำถาม	คะแนนของนักพัฒนาซอฟต์แวร์แต่ละราย								
	10	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
1	3	4	3	3	2	3	3	2	2
2	3	3	3	2	3	3	3	4	3
3	3	2	2	3	3	3	3	3	3
4	3	2	2	3	3	2	2	2	2
5	3	4	3	3	3	3	3	4	3
6	3	3	3	3	3	3	3	3	2
7	3	1	3	2	2	2	2	2	1
8	3	3	3	3	3	2	3	3	2
9	3	4	3	3	3	3	3	3	3
10	3	4	3	3	3	3	3	2	2
คะแนนรวม	30	30	28	28	28	27	28	28	23

โครงการที่ 2

คำถาม	คะแนนของนักพัฒนาซอฟต์แวร์แต่ละราย						
	1	**2**	**3**	**4**	**5**	**6**	**7**
1	3	3	3	3	3	3	4
2	4	3	3	3	3	3	3
3	4	3	3	3	3	3	3
4	4	3	3	3	3	3	3
5	3	3	3	3	3	3	3
6	4	3	3	3	3	3	3
7	3	3	3	4	4	4	3
8	4	4	3	3	3	3	4
9	4	3	3	3	3	3	3
10	5	4	3	3	3	3	3
คะแนนรวม	38	32	30	31	31	31	32

ผู้ทำแบบสอบถามคนที่ 2

ตำแหน่งงาน: Senior Programmer

ประสบการณ์ทำงาน: 6 ปี

โครงการที่ 1

คำถาม	คะแนนของนักพัฒนาซอฟต์แวร์แต่ละราย								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	5	3	3	3	3	3	3	3	3
2	5	3	3	3	3	3	3	3	3
3	4	3	3	3	3	3	3	3	3
4	3	3	2	3	3	3	3	3	3
5	5	3	3	3	3	3	3	3	3
6	4	3	3	3	3	3	3	3	3
7	3	4	3	3	3	3	3	3	3
8	5	3	3	3	3	3	3	3	3
9	3	4	4	3	3	3	3	3	3
10	4	4	4	3	3	3	3	3	3
คะแนนรวม	41	33	31	30	30	30	30	30	30

คำถาม	คะแนนของนักพัฒนาซอฟต์แวร์แต่ละราย								
	10	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
1	3	4	3	3	2	3	3	2	2
2	3	3	3	3	3	3	3	4	3
3	3	2	2	3	3	3	3	3	3
4	3	2	2	3	3	2	2	2	3
5	3	3	3	3	3	3	3	4	3
6	3	3	4	3	3	3	3	3	2
7	3	1	3	2	3	2	2	2	1
8	3	3	3	3	3	2	3	3	2
9	3	3	3	3	3	3	4	4	3
10	3	4	3	3	3	3	3	2	2
คะแนนรวม	30	28	29	29	29	27	29	29	24

โครงการที่ 2

คำถาม	คะแนนของนักพัฒนาซอฟต์แวร์แต่ละราย						
	1	**2**	**3**	**4**	**5**	**6**	**7**
1	3	3	3	3	3	3	5
2	4	3	3	3	3	3	3
3	4	3	3	3	3	3	2
4	4	3	3	3	3	3	3
5	3	3	3	3	3	3	3
6	3	3	3	3	3	3	3
7	3	3	3	4	4	4	3
8	4	4	3	3	3	3	4
9	4	3	4	3	3	3	3
10	5	4	3	3	3	3	3
คะแนนรวม	37	32	31	31	31	31	32

ผู้ทำแบบสอบถามคนที่ 3

ตำแหน่งงาน: Senior Programmer

ประสบการณ์ทำงาน: 6 ปี

โครงการที่ 1

คำถาม	คะแนนของนักพัฒนาซอฟต์แวร์แต่ละราย								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	5	3	3	3	3	3	3	3	3
2	5	3	2	3	3	3	3	3	3
3	4	3	3	3	3	3	3	3	3
4	3	4	2	3	4	3	3	3	3
5	4	3	3	3	3	3	3	3	3
6	4	3	3	3	3	3	3	3	3
7	2	4	3	3	3	3	3	3	3
8	5	3	3	3	3	3	3	3	3
9	4	3	4	3	3	3	3	3	3
10	4	4	3	4	3	3	3	3	3
คะแนนรวม	40	33	29	31	31	30	30	30	30

คำถาม	คะแนนของนักพัฒนาซอฟต์แวร์แต่ละราย								
	10	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
1	3	4	3	3	1	3	3	2	2
2	3	3	3	3	3	3	3	4	3
3	3	2	3	3	3	3	3	3	3
4	3	2	2	3	3	2	3	2	2
5	3	4	3	3	3	3	3	4	3
6	3	3	3	3	3	3	3	3	2
7	3	2	3	2	3	2	2	2	2
8	3	3	2	3	3	3	3	3	2
9	3	3	3	3	3	3	4	3	3
10	3	4	3	3	3	3	3	3	2
คะแนนรวม	30	30	28	29	29	28	30	29	24

โครงการที่ 2

คำถาม	คะแนนของนักพัฒนาซอฟต์แวร์แต่ละราย						
	1	**2**	**3**	**4**	**5**	**6**	**7**
1	3	3	3	3	3	3	5
2	4	3	3	3	3	3	3
3	4	3	3	3	3	3	3
4	4	3	3	3	3	3	3
5	3	3	3	3	3	3	3
6	4	3	3	4	4	4	3
7	3	3	3	3	3	3	4
8	4	4	3	3	3	3	5
9	4	3	4	3	3	3	3
10	4	3	3	3	3	3	3
คะแนนรวม	37	31	31	31	31	31	35

ผู้ทำแบบสอบถามคนที่ 4

ตำแหน่งงาน: Senior Programmer

ประสบการณ์ทำงาน: 2 ปี

โครงการที่ 1

คำถาม	คะแนนของนักพัฒนาซอฟต์แวร์แต่ละราย								
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	5	3	3	3	3	3	3	3	3
2	4	3	3	3	3	3	3	3	3
3	3	3	3	3	3	3	3	3	3
4	3	3	2	4	3	3	3	3	3
5	4	3	3	3	3	3	3	3	3
6	4	4	3	3	3	3	3	3	3
7	1	4	3	3	3	3	3	3	3
8	5	4	3	3	3	3	3	3	3
9	5	4	4	3	3	3	3	3	3
10	4	4	3	4	4	3	3	4	3
คะแนนรวม	38	29	30	32	31	30	30	30	30

คำถาม	คะแนนของนักพัฒนาซอฟต์แวร์แต่ละราย								
	10	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**
1	3	3	3	3	1	3	3	2	2
2	3	4	3	3	3	3	3	4	3
3	3	2	3	3	3	3	3	3	3
4	3	2	2	3	3	2	2	1	1
5	3	4	3	3	3	3	3	4	3
6	3	3	3	3	3	3	3	2	1
7	3	2	3	2	3	2	1	2	1
8	3	3	3	3	3	2	3	3	2
9	3	3	3	3	3	3	3	3	3
10	3	4	2	3	4	3	3	2	2
คะแนนรวม	30	30	28	29	29	27	27	26	21

โครงการที่ 2

คำถาม	คะแนนของนักพัฒนาซอฟต์แวร์แต่ละราย						
	1	**2**	**3**	**4**	**5**	**6**	**7**
1	3	3	3	3	3	3	4
2	4	3	3	3	3	3	3
3	4	3	3	3	3	3	3
4	4	3	3	3	3	3	2
5	3	3	3	3	3	3	3
6	4	3	3	3	3	3	3
7	3	3	3	4	4	4	3
8	4	4	3	3	3	3	4
9	4	3	4	3	3	3	3
10	4	4	3	3	3	3	4
คะแนนรวม	37	32	31	31	31	31	32

บรรณานุกรม

1. Aiko Yamashita, S.C., *Code smells as system-level in dictators of maintainability: An empirical study.* 2013: p. 2639-2640.
2. *Refactoring for Architecture Smells: An Introduction*. Available from: http://www.designsmells.com/articles/refactoring-for-architecture-smells-an-introduction/.
3. *Documentation: Getting Started - About Version Control*. Available from: https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control.
4. Sharma, T., *Designite: A Customizable Tool for Smell Mining in C# Repositories*, in *10th Seminar on Advanced Techniques and Tools for Software Evolution*. 2017: Madrid, Spain.
5. G. Suryanarayana, G.S., and T. Sharma, *Refactoring for Software Design Smells: Managing Technical Debt*, M. Kaufmann, Editor.
6. Fowler, M., *Refactoring: Improving the Design of Existing Programs*, ed. A.-W. Professional. 1999.
7. *Bayesian Rating – how to implement a weighted rating system*. Available from: http://www.thebroth.com/blog/118/bayesian-rating.
8. *Collective Choice: Rating Systems*. Available from: http://www.lifewithalacrity.com/2005/12/collective_choi.html.
9. J. Garcia, D.P., G. Edwards, and N. Medvidovic, *Identifying Architectural Bad Smells*, in *European Conference on Software Maintenance and Reengineering*. 2009 p. 255-258.
10. Tusher Sharma, M.F.a.D.S., *House of Cards: Code Smells in Open-source C# Repositories*, in *Empirical Software Engineering and Measurement*. 2017.
11. Shu Li, H.T., and Kosuke Takano, *Analysis of Software Developer Activity on a Distributed Version Control System*, in *Advanced Information Networking and Applications Workshops (WAINA)*. 2016. p. 701-707.
12. Ting, C., *The Application of the Function Point Analysis in Software Developers' Performance Evaluation*, in *Wireless Communications, Networking and Mobile*

Computing. 2008. p. 1-4.

13. *TopCoder*. Available from: https://www.topcoder.com/community/how-it-works/.

ประวัติผู้เขียน

ชื่อ-สกุล	ณทัศน์ จงประสิทธิ์
วัน เดือน ปี เกิด	12 กรกฎาคม 2534
สถานที่เกิด	โรงพยาบาลรามาธิบดี
วุฒิการศึกษา	วิทยาศาสตรบัณฑิต
ที่อยู่ปัจจุบัน	273 หมู่บ้านอยู่เจริญ ซอยลาดพร้าว 101 ถนนลาดพร้าว เขตวังทองหลาง แขวงคลองเจ้าคุณสิงห์ กรุงเทพฯ 10310