กรอบงานสำหรับตรวจจับและจัดลำดับข้อบกพร่องของซอฟต์แวร์ในชุดข้อมูลไม่สมดุล

นายธีรวิทย์ เชยกีวงศ์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2558

A Framework to Detect and Rank Software Defects in Imbalanced Data Sets

Mr. Teerawit Choeikiwong

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science Program in Software Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2015

หัวข้อวิทยานิพนธ์	กรอบงานสำหรับตรวจจับและจัดลำดับข้อบกพร่องของซอฟต์แวร์ในชุดข้อมูลไม่สมดุล
โดย	นายธีรวิทย์ เชยกีวงศ์
สาขาวิชา	วิศวกรรมซอฟต์แวร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	ดร. พีรพล เวทีกูล

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์
(รองศาสตราจารย์ ดร. สุพจน์ เตชวรสินสกุล)

คณะกรรมการสอบวิทยานิพนธ์

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

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(ดร. พีรพล เวทีกูล)

.....กรรมการภายนอกมหาวิทยาลัย
(ผู้ช่วยศาสตราจารย์ ดร. มฑุปายาส ทองมาก)

ธีรวิทย์ เชยกีวงศ์ : กรอบงานสำหรับตรวจจับและจัดลำดับข้อบกพร่องของซอฟต์แวร์ในชุดข้อมูลไม่สมดุล (A Framework to Detect and Rank Software Defects in Imbalanced Data Sets) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: ดร. พีรพล เวทีกูล, 106 หน้า.

ในกระบวนการประกันคุณภาพซอฟต์แวร์ จำเป็นอย่างยิ่งที่จะป้องกันไม่ให้ผลิตภัณฑ์ที่มีข้อบกพร่องส่งมอบให้กับลูกค้า เนื่องจากจะช่วยประหยัดต้นทุนในการบำรุงรักษา ทั้งยังส่งผลให้ซอฟต์แวร์มีคุณภาพและความน่าเชื่อถืออีกด้วย การตรวจจับข้อบกพร่องจึงถือเป็นกระบวนการสำคัญที่พยายามตรวจจับข้อบกพร่องที่มีโอกาสเกิดขึ้นในซอฟต์แวร์ได้โดยอัตโนมัติ โดยหลังจากพบข้อบกพร่องแล้ว จำเป็นอย่างยิ่งที่จะต้องระบุระดับความรุนแรง เพื่อหลีกเลี่ยงผลกระทบที่อาจก่อให้เกิดความล้มเหลวของระบบ งานวิจัยจำนวนมากได้ประยุกต์ใช้การทำเหมืองข้อมูลสำหรับตรวจจับข้อบกพร่อง อย่างไรก็ตามประสิทธิภาพการตรวจจับยังคงไม่เป็นที่น่าพอใจ เนื่องจากปัญหาความไม่สมดุลของคลาสและจำนวนข้อมูลไม่เพียงพอ ซึ่งเป็นสาเหตุให้ประสิทธิภาพในการตรวจจับและระบุระดับความรุนแรงของข้อบกพร่องลดลง

งานวิจัยนี้จึงนำเสนอกรอบงานสำหรับตรวจจับและจัดลำดับข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์ โดยมุ่งเน้นที่งานสองส่วน ส่วนแรกเป็นการตรวจจับข้อบกพร่องโดยการประยุกต์ใช้วิธีอาร์เอสวีเอ็ม ซึ่งลดไบแอสของคลาสที่เป็นเสียงข้างมากโดยใช้การปรับแก้ค่าขีดแบ่ง ในส่วนที่สองจะนำเสนอการจัดลำดับข้อบกพร่องตามระดับความรุนแรงด้วยวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์ (เอเอชพี) ที่พิจารณาจากค่าความเชื่อมั่น และระดับความรุนแรงของข้อบกพร่อง นอกจากนี้ยังได้นำเสนออัลกอริทึม OS-YATSI ที่รวมกันระหว่างวิธีการเรียนรู้แบบกึ่งมีผู้สอนและเทคนิคการสุ่มเพิ่มตัวอย่างในการปรับปรุงประสิทธิภาพของการจัดลำดับข้อบกพร่องตามระดับความรุนแรง โดยทำการทดลองกับซอฟต์แวร์ภาษาจาวาจำนวน 15 โปรแกรม ซึ่งผลการทดลองพบว่า กรอบงานที่นำเสนอให้ผลดีกว่าวิธีการจำแนกประเภทแบบดั้งเดิม โดยในส่วนของการทำนายข้อบกพร่อง วิธีอาร์เอสวีเอ็มให้ผลดีกว่าวิธีการแบบดั้งเดิมอย่างมีนัยสำคัญจำนวน 6 และ 5 โปรแกรมในเทอมของมาตรวัดมัชฌิมฮาร์โมนิคและมัชฌิมเรขาคณิตตามลำดับ และในส่วนของการจัดลำดับข้อบกพร่อง วิธีเอเอชพีที่นำเสนอให้ประสิทธิภาพดีกว่าการพิจารณาระดับความรุนแรงเพียงอย่างเดียวจำนวน 10 จากทั้งหมด 15 โปรแกรม ในเทอมของมาตรวัดอัตราเฉลี่ยประสิทธิภาพ นอกจากนี้ วิธี OS-YATSI ให้ผลดีกว่าวิธีการแบบดั้งเดิมอย่างมีนัยสำคัญโดยสามารถปรับปรุงประสิทธิภาพการทำนายเฉลี่ยเพิ่มขึ้น 28.79% ในเทอมของมาตรวัดมัชฌิมฮาร์โมนิคแบบค่าเฉลี่ยมหภาค

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมซอฟต์แวร์ ลายมือชื่อ อ.ที่ปรึกษาหลัก .....

ปีการศึกษา 2558

5670234021 : MAJOR SOFTWARE ENGINEERING
KEYWORDS: SOFTWARE DEFECT PREICTION / SEVERITY CATEGORIZATION / IMBALANCED ISSUE / SEMI-SUPERVISED LEARNING

TEERAWIT CHOEIKIWONG: A Framework to Detect and Rank Software Defects in Imbalanced Data Sets. ADVISOR: PEERAPON VATEEKUL, Ph.D., 106 pp.

In the software assurance process, it is crucial to prevent software with defected modules to be published to users since it can save the maintenance cost and increase software quality and reliability. Software defect prediction is recognized as an important process to automatically predict the possibility of having a defect in the software. After defects are detected, it is then needed to identify their severity levels to avoid any effects that may obstruct the whole system. However, most defect prediction studies are often faced with an imbalanced issue and scarcity of data, which causes decreased prediction performance.

This research presents a framework to detect and rank defects in software. The Framework focuses on two tasks. First, we will capture defects by applying an unbiased SVM called “R-SVM,” which reduces a bias of the majority class by using the concept of threshold adjustment. Second, the detected modules will be ranked according to their severity levels by using Analytic Hierarchy Process (AHP) that considers a confidence value and severity level of the defects. Furthermore, we also propose a novel algorithm called “OS-YATSI,” that combines semi-supervised learning and oversampling strategy to improve ranking performance. The experiment was conducted on 15 Java programs. The result showed that the proposed framework outperformed all of the traditional approaches. In the defect prediction model, R-SVM significantly outperformed others on 6 programs in terms of F1. In the defect ranking model, the AHP method is the winner on 10 out of 15 programs in terms of Average Performance Rate (APR). In addition, OS-YATSI significantly outperformed all baseline classifiers on all programs at an average of 28.79% improvement in term of macro F1.

Department: Computer Engineering Student's Signature .....
Field of Study: Software Engineering Advisor's Signature .....
Academic Year: 2015

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงได้อย่างสมบูรณ์ด้วยความอนุเคราะห์อย่างดียิ่งจาก ดร. พีรพล เวทีกูล อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่สละเวลาในการให้คำแนะนำและคำปรึกษาที่เป็นประโยชน์ในการทำวิจัย รวมถึงแนวทางการแก้ไขปัญหาที่เกิดขึ้นในระหว่างการทำวิจัยและความรู้ทางวิชาการต่างๆ นอกจากนี้ยังคอยสนับสนุนและผลักดันผู้วิจัยอย่างเต็มกำลัง เพื่อให้งานวิจัยมีความก้าวหน้า ทำให้ผู้วิจัยสามารถพัฒนางานวิจัยออกมาได้อย่างมีคุณภาพ ขอกราบขอบพระคุณอาจารย์เป็นอย่างสูง

ขอกราบขอบพระคุณคณะกรรมการสอบวิทยานิพนธ์ทั้งสองท่าน ได้แก่ รองศาสตราจารย์ ดร. ทวีตีย์ เสนีย์วงศ์ ณ อยุธยา ประธานกรรมการสอบวิทยานิพนธ์ และผู้ช่วยศาสตราจารย์ ดร. มฑุปายาส ทองมาก กรรมการผู้ทรงคุณวุฒิภายนอกมหาวิทยาลัย ที่กรุณาสละเวลาในการให้คำแนะนำและเสนอแนะแนวทางการแก้ไขปัญหาที่เป็นประโยชน์อย่างดียิ่งกับงานวิจัย อีกทั้งยังพิจารณาความถูกต้องของเนื้อหาวิทยานิพนธ์อย่างละเอียดถี่ถ้วน เพื่อให้วิทยานิพนธ์ฉบับนี้มีคุณภาพและมีความสมบูรณ์มากยิ่งขึ้น

ขอกราบขอบพระคุณคณาจารย์ภาควิชาวิศวกรรมคอมพิวเตอร์ทุกท่านที่ให้ความรู้ด้านวิชาการต่างๆและคอยอบรมสั่งสอนผู้วิจัย โดยเฉพาะอย่างยิ่ง ผู้ช่วยศาสตราจารย์ นครทิพย์ พร้อมพูล ที่กรุณาสละเวลาในการให้คำปรึกษาและให้คำแนะนำที่เป็นประโยชน์ในการทำวิจัย และขอขอบคุณบุคลากรภาควิชาวิศวกรรมคอมพิวเตอร์ทุกท่านที่คอยให้ความช่วยเหลือผู้วิจัยในเรื่องต่างๆ ด้วยความเต็มใจเสมอมา

ขอขอบคุณพี่ๆ เพื่อนๆและน้องๆ ทั้งใน SE Lab และใน MIND Lab สำหรับคำแนะนำและคำปรึกษาต่างๆ ทั้งในด้านการเรียนและการทำวิจัย รวมถึงความช่วยเหลือ กำลังใจและความอบอุ่นที่มีให้แก่ผู้วิจัยเสมอมา

สุดท้ายนี้ ขอกราบขอบพระคุณบิดา มารดา และสมาชิกทุกคนในครอบครัว ที่มอบโอกาสที่ดีในการศึกษาต่อในระดับปริญญามหาบัณฑิต และคอยสนับสนุนผู้วิจัยในด้านต่างๆ อีกทั้งยังคอยให้กำลังใจและอยู่เคียงข้างผู้วิจัย ทั้งในยามทุกข์และยามสุขเสมอมา ซึ่งเป็นสิ่งที่สำคัญที่เป็นแรงบันดาลใจให้ผู้วิจัยตั้งใจทำงานวิจัยฉบับนี้ให้สำเร็จลุล่วงอย่างสมบูรณ์ ผู้วิจัยขอมอบความภาคภูมิใจและความสำเร็จให้กับผู้มีส่วนร่วมทุกท่าน และหวังเป็นอย่างยิ่งว่า วิทยานิพนธ์ฉบับนี้จะสร้างแรงบันดาลใจ และผลักดันให้เกิดการพัฒนาทางด้านวิศวกรรมซอฟต์แวร์ต่อไป ขอบคุณครับ

สารบัญ

หน้า

หน้า

หน้า

หน้า

สารบัญตาราง

หน้า

สารบัญภาพ

หน้า

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญของปัญหา

ข้อบกพร่องของซอฟต์แวร์ คือ สิ่งผิดปกติ (anomaly) ในซอฟต์แวร์ที่มีสาเหตุมาจากข้อผิดพลาดของมนุษย์ เช่น พิมพ์ชื่อตัวแปรผิด ความเข้าใจผิดในสัญลักษณ์ที่ใช้ในการออกแบบ เป็นต้น ซึ่งอาจเป็นสาเหตุให้ซอฟต์แวร์ทำงานไม่ถูกต้องหรือไม่เป็นไปตามข้อกำหนด (specifications) ข้อบกพร่องเหล่านี้จะกลายเป็นความล้มเหลวของระบบซอฟต์แวร์ก็ต่อเมื่อซอฟต์แวร์ส่วนที่เกิดข้อบกพร่องนั้นถูกกระตุ้นให้ทำงาน โดยสามารถพบข้อบกพร่องได้ในรหัสต้นฉบับ (source code) หรือผลิตภัณฑ์งานอื่นๆ เช่น ข้อกำหนดความต้องการ เอกสารการออกแบบ ฯลฯ ข้อบกพร่องในเอกสารเหล่านี้สามารถค้นพบด้วยการทบทวน (review) ส่วนข้อบกพร่องที่พบในรหัสต้นฉบับบางครั้ง เรียกว่า “บั๊ก” (bugs) จะถูกค้นพบในกระบวนการทบทวน หรือการทดสอบ (testing) จากที่กล่าวมาข้างต้นจะเห็นได้ว่าการสร้างผลิตภัณฑ์ซอฟต์แวร์โดยปราศจากข้อบกพร่องหรือบั๊ก เป็นสิ่งที่ทำได้ยาก เนื่องจากมนุษย์เป็นผู้พัฒนาจึงทำให้เกิดข้อผิดพลาด

ในปัจจุบันองค์กรที่พัฒนาซอฟต์แวร์ต่างให้ความสำคัญกับกระบวนการผลิตและการประกันคุณภาพซอฟต์แวร์ เพื่อให้ได้ซอฟต์แวร์ที่มีคุณภาพ สามารถตอบสนองต่อความต้องการของลูกค้าและสามารถใช้งานได้จริง ทั้งนี้การจะได้มาซึ่งซอฟต์แวร์ที่มีคุณภาพนั้นจำเป็นที่จะต้องมีการตรวจหาข้อบกพร่อง (Defect Detection) ซึ่งถือเป็นกระบวนการที่สำคัญในงานทางด้านวิศวกรรมซอฟต์แวร์ เนื่องจากสามารถตรวจจับและวิเคราะห์โอกาสที่จะเกิดข้อบกพร่องขึ้นภายในซอฟต์แวร์ได้โดยอัตโนมัติ ซึ่งช่วยให้นักพัฒนาหรือโปรแกรมเมอร์ทราบถึงสาเหตุของปัญหาที่เกิดขึ้น ทำให้สามารถแก้ไขปัญหาได้อย่างถูกต้อง รวดเร็ว และอยู่ภายในระยะเวลาที่กำหนด ส่งผลให้ซอฟต์แวร์มีคุณภาพและมีความน่าเชื่อถือมากยิ่งขึ้น โดยปัจจัยเสี่ยงที่ทำให้ซอฟต์แวร์เกิดข้อบกพร่องก็คือ ขนาดและความซับซ้อนของซอฟต์แวร์ ซึ่งเป็นคุณลักษณะของซอฟต์แวร์ที่สะท้อนให้เห็นถึงความเชื่อที่ว่า ถ้าซอฟต์แวร์ใดๆมีขนาดใหญ่แสดงว่าซอฟต์แวร์นั้นจะต้องมีการทำงานที่ซับซ้อนมาก ซึ่งการทำงานที่ซับซ้อนนี้เองเป็นสาเหตุให้ซอฟต์แวร์มีโอกาสเกิดข้อบกพร่องขึ้นได้

ที่ผ่านมาได้มีการนำเทคนิคการทำเหมืองข้อมูลและการเรียนรู้เครื่องจักรมาประยุกต์ใช้ในงานวิจัยด้านการตรวจจับข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์แบบอัตโนมัติ ในปี ค.ศ. 2008 K.O. Elish และคณะ [1] ได้ประยุกต์ใช้ Support Vector Machines (SVM) ในการทำนายข้อบกพร่องบนชุดข้อมูล MDP ต่อมาในปี ค.ศ. 2010 N. Seliya และคณะ [2] ได้เสนอวิธีการ RBBag ในการทำนาย

ข้อบกพร่องในระบบซอฟต์แวร์ที่มีการประกันภัยสูง และในปีเดียวกัน D. Gray และคณะ [3] ได้ประยุกต์ใช้ SVM ในการทำนายข้อบกพร่องโดยวิเคราะห์จากคุณลักษณะภายในของซอฟต์แวร์ นอกจากนี้ในปี ค.ศ. 2013 W. Shuo และคณะ [4] ได้ประยุกต์ใช้ Ensemble Algorithm และการสุ่มตัวอย่างในการตรวจจับข้อบกพร่อง ซึ่งผลการทดลองจากงานวิจัยเหล่านี้พบว่า ประสิทธิภาพในการตรวจจับข้อบกพร่องมีค่าค่อนข้างต่ำ เนื่องจากไม่ได้พิจารณาคุณลักษณะของข้อมูลที่เกิดปัญหาความไม่สมดุลของคลาส (Class imbalanced problem) ซึ่งเป็นปัจจัยสำคัญที่ส่งผลให้ประสิทธิภาพการทำนายลดลง

ปัญหาความไม่สมดุลของคลาส เป็นปัญหาใหญ่ในงานด้านการทำเหมืองข้อมูล เนื่องจากการประยุกต์ใช้เทคโนโลยีมีความหลากหลายและยังคงเพิ่มขึ้นเรื่อยๆ ทำให้ข้อมูลมีขนาดใหญ่ขึ้น จึงทำให้การจำแนกประเภทของข้อมูลทำได้ยาก ซึ่งปัญหาความไม่สมดุลของข้อมูล เป็นปัญหาที่เกิดขึ้นกับข้อมูลทุกๆประเภท โดยทั่วไปจะประกอบด้วยคลาส 2 กลุ่ม คือ คลาสที่มีเสียงข้างมาก (majority class) และคลาสที่มีเสียงข้างน้อย (minority class) โดยที่คลาสที่มีเสียงข้างมากจะมีสัดส่วนมากกว่าคลาสที่มีเสียงข้างน้อย ตัวจำแนกประเภทส่วนใหญ่มักที่จะสนใจการจำแนกประเภทคลาสที่มีเสียงข้างมาก ในขณะที่เราสนใจคลาสที่มีเสียงข้างน้อย ดังนั้นตัวจำแนกประเภทจึงให้ผลลัพธ์ที่ดีกับคลาสที่มีเสียงข้างมาก แต่จะให้ผลลัพธ์ที่แย่กับคลาสที่มีเสียงข้างน้อย นำมาซึ่งการจำแนกข้อมูลในคลาสที่ผิดพลาด ตัวอย่างเช่น ชุดข้อมูลเอ็มดีพี (MDP) ขององค์การบริหารอวกาศและการบินแห่งชาติ (NASA) [5] ประกอบด้วยโครงการทั้งหมด 12 โครงการ โดยมีมอดูลข้อบกพร่องเฉลี่ยร้อยละ 20 ของทั้งหมด 12 โครงการ ซึ่งในโครงการพีซี2 (PC2) พบมอดูลข้อบกพร่องน้อยที่สุดเพียงร้อยละ 2.5 เท่านั้น หมายความว่าโครงการพีซี2 มีจำนวนคลาสที่มีเสียงข้างมาก (มอดูลที่ไม่มีข้อบกพร่อง) ร้อยละ 97.5 และมีจำนวนคลาสที่มีเสียงข้างน้อย (มอดูลที่มีข้อบกพร่อง) เพียงร้อยละ 2.5 เท่านั้น จะเห็นว่าชุดข้อมูลดังกล่าวเป็นชุดข้อมูลที่เกิดปัญหาความไม่สมดุลของคลาส เนื่องจากมีสัดส่วนระหว่างคลาสทั้ง 2 กลุ่มแตกต่างกันอย่างชัดเจน ซึ่งเป็นสาเหตุที่ทำให้ประสิทธิภาพในการจำแนกประเภทลดลง

นอกจากปัญหาความไม่สมดุลของคลาสในชุดข้อมูล การตรวจจับข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์ดังที่ได้กล่าวไปข้างต้นทำได้เฉพาะการระบุคลาสที่มีปัญหาเท่านั้น แต่ไม่ได้ให้คำแนะนำในการแก้ไขข้อบกพร่องที่เกิดขึ้นว่าควรจะแก้ปัญหาข้อบกพร่องใดก่อน ดังนั้น เพื่อให้นักพัฒนาสามารถตัดสินใจแก้ไขปัญหาของข้อบกพร่องที่เกิดขึ้นได้อย่างเหมาะสม การจัดลำดับข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์ตามระดับความรุนแรงจึงเป็นกระบวนการสำคัญที่ช่วยระบุได้ว่าข้อบกพร่องที่เกิดขึ้นนั้นส่งผลกระทบต่อซอฟต์แวร์มากน้อยเพียงใด

งานวิจัยที่ผ่านมาได้นำเสนอแบบจำลองการทำนายระดับความรุนแรงของข้อบกพร่องจากรายงานข้อบกพร่องของซอฟต์แวร์ โดยในปี ค.ศ. 2008 T. Menzies และคณะ [6] ได้เสนอวิธี

SEVERIS ที่ประยุกต์ใช้การทำเหมืองข้อความ (text mining) ในการสกัดและวิเคราะห์รายละเอียดจากรายงานข้อบกพร่อง ต่อมาในปี ค.ศ. 2010 A. Lamkanfi และคณะ [7] ได้ประยุกต์ใช้วิธี Naïve Bayes (NB) ในการทำนายระดับความรุนแรงบนชุดข้อมูลจาก Bugzilla นอกจากนี้ในปี ค.ศ. 2011 [8] ยังได้ทำการเปรียบเทียบ NB กับวิธีการจำแนกประเภทอื่นๆอีกด้วย ถัดมาในปี ค.ศ. 2012 K. K. Chaturvedi และคณะ [9] ได้ทำการเปรียบเทียบแบบจำลองที่สร้างด้วยวิธีการจำแนกประเภทแบบต่างๆ และในปีเดียวกัน Y. Cheng-Zen และคณะ [10] ได้ประยุกต์ใช้วิธีการคัดเลือกคุณลักษณะ เพื่อช่วยในการปรับปรุงประสิทธิภาพของการทำนาย หลังจากนั้นในปี ค.ศ. 2014 N.K. Singha Roy และคณะ [11] ได้เสนอแบบจำลองการทำนายระดับความรุนแรงโดยประยุกต์ใช้วิธี Bi-grams และการคัดเลือกคุณลักษณะ จะเห็นว่าการทำนายระดับความรุนแรงของข้อบกพร่องเหล่านี้เป็นแนวทางที่เป็นประโยชน์สำหรับนักพัฒนาในการตัดสินใจแก้ไขปัญหาที่เกิดขึ้น อย่างไรก็ตามงานวิจัยที่กล่าวมาทั้งหมดยังขาดการพิจารณาปัญหาความไม่สมดุลของคลาส ซึ่งเป็นสาเหตุให้ประสิทธิภาพการทำนายของแบบจำลองลดลง นอกจากนี้การจัดลำดับของคลาสที่เกิดข้อบกพร่องมีการพิจารณาจากระดับความรุนแรงเพียงอย่างเดียวเท่านั้น แต่ในความเป็นจริงแล้วคลาสที่มีระดับความรุนแรงสูง อาจมีโอกาสเกิดข้อบกพร่องน้อยมาก หรือแทบไม่มีโอกาสเกิดขึ้นเลยก็เป็นได้ ซึ่งหากมีการพิจารณาค่าความมั่นใจของการเกิดข้อบกพร่องร่วมกันกับระดับความรุนแรง จะทำให้การจัดลำดับมีประสิทธิภาพมากยิ่งขึ้น เพราะสามารถบ่งชี้ถึงคลาสที่มีระดับความรุนแรงในขณะเดียวกันก็สามารถบอกโอกาสของการเกิดข้อบกพร่องขึ้นในคลาสนั้นด้วย

ดังนั้น งานวิจัยนี้จึงมีจุดประสงค์ที่จะนำเสนอกรอบงานที่สามารถตรวจจับข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์โดยประยุกต์ใช้วิธีการจำแนกประเภทแบบ SVM ที่มีชื่อเรียกว่า R-SVM [12] ซึ่งสามารถจัดการกับปัญหาความไม่สมดุลของคลาสในชุดข้อมูลได้ นอกจากนี้กรอบงานที่นำเสนอยังสามารถระบุถึงระดับความรุนแรงของข้อบกพร่อง และจัดลำดับความสำคัญของข้อบกพร่องที่เกิดขึ้นตามระดับความรุนแรงที่ได้นิยามขึ้นตามบริบทของงานวิจัย โดยที่นิยามของความรุนแรง คือ ระดับของผลกระทบการขึ้นต่อกันของคลาสที่เกิดข้อผิดพลาดที่ส่งผลกระทบต่อการทำงานของซอฟต์แวร์ โดยประยุกต์ใช้วิธีกระบวนการลำดับชั้นเชิงวิเคราะห์ [13] เพื่อช่วยในการตัดสินใจหาข้อบกพร่องที่เหมาะสมที่สุดที่ควรทำการแก้ไขก่อน ซึ่งจะพิจารณาจากเกณฑ์ตัดสินใจ 2 เกณฑ์ คือ ค่าความเชื่อมั่นของการเกิดข้อบกพร่อง และระดับความรุนแรงของข้อบกพร่อง นอกจากนี้กรอบงานยังนำเสนอวิธีการ OS-YATSI ที่มีการประยุกต์ใช้วิธี YATSI [14] และการสุ่มตัวอย่างแบบ SMOTE [15] ในการปรับปรุงประสิทธิภาพของการจัดลำดับของข้อบกพร่องตามระดับความรุนแรง โดยอนุญาตให้นักพัฒนาหรือผู้ใช้กรอกระดับความรุนแรงของข้อบกพร่องเพียงบางส่วน และนำข้อมูลเหล่านี้ไปสร้างแบบจำลองเพื่อช่วยระบุระดับความรุนแรงของข้อบกพร่องส่วนที่เหลือ ทั้งนี้ผลลัพธ์ที่ได้สามารถใช้

เป็นข้อเสนอแนะให้กับผู้พัฒนาในการแก้ไขปัญหาตามระดับความรุนแรงของข้อบกพร่องที่ส่งผลกระทบต่อซอฟต์แวร์ พร้อมทั้งพัฒนาเครื่องมือต้นแบบที่สามารถตรวจจับข้อบกพร่องของซอฟต์แวร์และจัดลำดับความสำคัญโดยให้ข้อเสนอแนะในการแก้ไขปัญหาตามระดับความรุนแรงของข้อบกพร่องได้

1.2 วัตถุประสงค์ของงานวิจัย

1) นำเสนอแนวคิดในการสร้างกรอบงานสำหรับตรวจจับข้อบกพร่องและจัดลำดับข้อบกพร่องตามระดับความรุนแรง ซึ่งประกอบด้วย 2 แบบจำลอง คือ

(1) แบบจำลองการทำนายข้อบกพร่องของซอฟต์แวร์โดยใช้วิธีการจำแนกประเภทที่สามารถแก้ไขปัญหาความไม่สมดุลของคลาสได้ (Software Defect Prediction Model)

(2) แบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์ตามระดับความรุนแรงที่นิยามขึ้นตามบริบทของงานวิจัย (Software Defect Severity Ranking Model)

2) เปรียบเทียบแบบจำลองการทำนายข้อบกพร่องของซอฟต์แวร์ (Comparison of Software Defect Prediction Model)

3) เปรียบเทียบแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์ตามระดับความรุนแรง (Comparison of Software Defect Severity Ranking Model)

พัฒนาเครื่องมือต้นแบบที่สนับสนุนกรอบงานสำหรับตรวจจับข้อบกพร่องและจัดลำดับข้อบกพร่องตามระดับความรุนแรงอย่างอัตโนมัติ (Prototype Tool of Software Defect Prediction and Ranking System)

1.3 ขอบเขตของงานวิจัย

1) งานวิจัยนี้ทำการเสนอกรอบงานสำหรับการตรวจจับข้อบกพร่องและจัดลำดับข้อบกพร่องตามระดับความรุนแรง โดยจะประกอบด้วย 2 แบบจำลอง คือ

(1) แบบจำลองการทำนายข้อบกพร่องของซอฟต์แวร์จากรหัสต้นฉบับโดยใช้วิธีการจำแนกประเภทที่สามารถแก้ไขปัญหาความไม่สมดุลของคลาสได้

(2) แบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์ตามระดับความรุนแรง

2) การทำนายข้อบกพร่องของซอฟต์แวร์ในงานวิจัยนี้ จะพิจารณาข้อบกพร่องที่เกิดขึ้นในระดับคลาสเท่านั้น และไม่สามารถระบุประเภทของข้อบกพร่องที่เกิดขึ้นได้

3) การจัดลำดับความสำคัญของข้อบกพร่องที่เกิดขึ้นตามระดับความรุนแรงในงานวิจัยนี้ จะพิจารณาจากระดับความรุนแรงที่ถูกนิยามขึ้นตามบริบทของงานวิจัยเท่านั้น

4) แบบจำลองในการทำนายข้อบกพร่องของซอฟต์แวร์และแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์ในงานวิจัยนี้ ใช้สำหรับซอฟต์แวร์เชิงวัตถุที่พัฒนาด้วยภาษาจาวาเท่านั้น

5) พัฒนาเครื่องมือสำหรับการทำนายข้อบกพร่องของซอฟต์แวร์ ซึ่งจะแสดงผลการทำนายและจัดลำดับความสำคัญของข้อบกพร่องที่เกิดขึ้นตามระดับความรุนแรงเท่านั้น มิได้เสนอแนวทางในการปรับปรุงแก้ไข โดยพัฒนาขึ้นด้วยภาษาจาวา และทดสอบบนระบบปฏิบัติการวินโดวส์ (Windows)

1.4 ประโยชน์ของงานวิจัย

1) ได้กรอบงานที่สามารถนำไปใช้ในการทำนายข้อบกพร่องของซอฟต์แวร์และสามารถชี้บอกถึงระดับความรุนแรงของข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์ได้

2) ได้เครื่องมือที่สนับสนุนการทำนายข้อบกพร่องของซอฟต์แวร์และจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรง เพื่อเป็นแนวทางให้ผู้พัฒนาซอฟต์แวร์พิจารณาแก้ไขซอฟต์แวร์โดยมุ่งเน้นในส่วนที่เกิดข้อบกพร่องโดยตรง ซึ่งจะช่วยลดระยะเวลาและความพยายามที่ใช้ในการปรับปรุงแก้ไขซอฟต์แวร์

1.5 ขั้นตอนและวิธีการดำเนินการวิจัย

1) ศึกษาความรู้และทฤษฎีที่เกี่ยวข้อง

(1) วิธีการจำแนกประเภททางด้านการเรียนรู้เครื่องและการทำเหมืองข้อมูล

(2) มาตรวัดทางด้านซอฟต์แวร์และคุณลักษณะที่เหมาะสมสำหรับการทำนายข้อบกพร่อง

(3) กระบวนการลำดับชั้นเชิงวิเคราะห์

(4) ศึกษานิยาม ประเภท และระดับความรุนแรงของข้อบกพร่องตามมาตรฐาน IEEE 610.12-1990 และ IEEE 1044-2009

2) วิเคราะห์และออกแบบวิธีการทำนายข้อบกพร่องของซอฟต์แวร์ที่จะใช้ในงานวิจัยนี้

(1) ออกแบบขั้นตอนของการสกัดมาตรวัดจากรหัสต้นฉบับ

(2) ออกแบบขั้นตอนการสร้างแบบจำลองด้วยวิธีการจำแนกประเภท

(3) ทดลองและประเมินผลประสิทธิภาพการทำนายด้วยตัวชี้วัดที่เกี่ยวข้อง

3) วิเคราะห์และออกแบบวิธีการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงที่จะใช้ในงานวิจัยนี้

(1) ออกแบบขั้นตอนการสร้างแบบจำลองด้วยวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์

(2) ออกแบบขั้นตอนการสร้างแบบจำลองด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน

(3) ทดลองและประเมินผลประสิทธิภาพการระบุระดับความรุนแรงด้วยตัวชี้วัดที่เกี่ยวข้อง

4) ออกแบบและพัฒนาเครื่องมือที่สนับสนุนแนวคิดและวิธีวิจัยที่นำเสนอ

5) จัดทำบทความวิชาการ และเผยแพร่บทความวิชาการ

6) สรุปผล ข้อเสนอแนะ งานวิจัยในอนาคตและเรียบเรียงวิทยานิพนธ์

1.6 โครงสร้างของเนื้อหาในวิทยานิพนธ์

เนื้อหาของวิทยานิพนธ์ฉบับนี้แบ่งออกเป็น 6 บท คือ บทที่ 1 เป็นบทนำ บทที่ 2 กล่าวถึงทฤษฎีและงานวิจัยที่เกี่ยวข้อง บทที่ 3 นำเสนอแนวคิดของงานวิจัยและขั้นตอนในการสร้างแบบจำลองสำหรับการทำนายข้อบกพร่องและการจัดลำดับความสำคัญของข้อบกพร่อง ซึ่งประกอบด้วยขั้นตอนต่างๆในการดำเนินการวิจัย รวมทั้งแนวคิดและรายละเอียดในการสร้างและการประเมินแบบจำลอง บทที่ 4 เป็นบทที่แสดงผลการทดลองและการประเมินผลแบบจำลองที่สร้างขึ้นพร้อมทั้งเปรียบเทียบแบบจำลองที่ได้นำเสนอกับแบบจำลองอื่นๆ บทที่ 5 เป็นบทที่อธิบายการออกแบบและพัฒนาเครื่องมือต้นแบบตามแบบจำลองที่นำเสนอ และบทที่ 6 เป็นบทสุดท้าย ซึ่งจะเป็นบทสรุปของงานวิจัย ข้อจำกัด รวมทั้งข้อเสนอแนะ งานวิจัยในอนาคต และบทความวิชาการที่ตีพิมพ์

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ทฤษฎีที่เกี่ยวข้อง

2.1.1 มาตรวัดซอฟต์แวร์ (Software Metrics)

การวัด (Measurement) คือ การกำหนดค่าตัวเลขหรือตัวอักษรให้กับคุณลักษณะของสิ่งที่เราสนใจ (Entity) เพื่อที่จะอธิบายคุณลักษณะ (Attribute) นั้นๆ ซึ่งสิ่งที่เราสนใจอาจเป็นวัตถุ เหตุการณ์ หรือ การออกแบบซอฟต์แวร์ โดยเราจะเรียกคุณลักษณะของสิ่งที่สนใจว่า ตัววัด (Metrics) หรือ มาตรวัด

การวัดถือเป็นองค์ประกอบที่มีความสำคัญในกระบวนการพัฒนาซอฟต์แวร์ เนื่องจากสามารถช่วยบอกให้ผู้พัฒนาทราบว่าซอฟต์แวร์ที่พัฒนานั้นเป็นไปตามเป้าหมายและกำหนดการที่วางไว้หรือไม่ การวัดช่วยในการประมาณค่าใช้จ่าย ต้นทุน กำลังคน และระยะเวลาของการพัฒนาซอฟต์แวร์ การวัดจำนวนข้อผิดพลาดของซอฟต์แวร์ช่วยให้ผู้พัฒนาทราบถึงความเชื่อถือได้ของซอฟต์แวร์และสามารถวางแผนการทดสอบได้อย่างเหมาะสม สำหรับการสร้างระบบหรือผลิตภัณฑ์หนึ่งๆ จะใช้การวัดเพื่อประเมินคุณภาพของผลิตภัณฑ์และกำหนดคุณสมบัติของผลิตภัณฑ์ ซึ่งมาตรวัดที่ได้จะสะท้อนให้เห็นถึงประโยชน์ต่างๆ และหนึ่งประโยชน์หลักที่มีความสำคัญก็คือ การให้ข้อมูลในการทำนายข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์

ในปัจจุบันได้มีการประยุกต์ใช้มาตรวัดต่างๆ สำหรับการทำนายข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์ การวัดในงานวิจัยนี้มีวัตถุประสงค์ เพื่อชี้ให้เห็นว่ามาตรวัดขนาดและโครงสร้างของผลิตภัณฑ์ซอฟต์แวร์ สามารถสะท้อนให้เห็นถึงแนวโน้มของการเกิดข้อบกพร่องในซอฟต์แวร์ได้ ผู้วิจัยได้ศึกษาและรวบรวมมาตรวัดจากงานวิจัยที่ผ่านมาในอดีต [16-20] และเลือกใช้มาตรวัดที่บ่งชี้ถึงขนาดและโครงสร้างของผลิตภัณฑ์ซอฟต์แวร์ที่นิยมใช้งานกันอย่างแพร่หลาย จำนวนทั้งหมด 17 มาตรวัด โดยสกัดค่าจากรหัสต้นฉบับด้วยเครื่องมือ CKJM [21] สามารถแสดงรายละเอียดของแต่ละมาตรวัดได้ดังตารางที่ 1

ตารางที่ 1 มาตรวัดทั้งหมดที่ใช้ในงานวิจัย

มาตรวัด	คำอธิบาย	แหล่งอ้างอิง
Weight Method per Class (WMC)	มาตรวัดที่ใช้วัดความซับซ้อนของคลาสซึ่งเกิดจาก ผลรวมของความซับซ้อนของเมท็อดทั้งหมดภายในคลาส	C&K [16]

ตารางที่ 1 มาตรวัดทั้งหมดที่ใช้ในงานวิจัย (ต่อ)

มาตรวัด	คำอธิบาย	แหล่งอ้างอิง
Depth of Inheritance Tree (DIT)	มาตรวัดที่ใช้วัดความยาวสูงสุดจากคลาสที่สนใจไปยังรากของต้นไม้การสืบทอด	C&K [16]
Number of Children (NOC)	มาตรวัดที่ใช้ในการวัดจำนวนคลาสลูก	C&K [16]
Coupling Between Object classes (CBO)	มาตรวัดที่ใช้ในการนับจำนวนคลาสที่มีความขึ้นต่อกันของเมท็อดหรือตัวแปรที่อยู่คนละคลาส	C&K [16]
Response for a class (RFC)	มาตรวัดที่ใช้ในการนับจำนวนเมท็อดของคลาสและจำนวนเมท็อดที่ถูกเรียกใช้งานโดยเมท็อดของคลาสนี้	C&K [16]
Lack of Cohesion in Methods (LCOM)	มาตรวัดที่ใช้ในการวัดจำนวนเซ็ตที่ไม่ยึดเหนี่ยวกันของเมท็อดภายในคลาส	C&K [16]
Afferent couplings (Ca)	มาตรวัดที่วัดความซับซ้อนของการไหลของข้อมูลเข้า	Martin [17]
Efferent couplings (Ce)	มาตรวัดที่วัดความซับซ้อนของการไหลของข้อมูลออก	Martin [17]
Number of Public Methods (NPM)	มาตรวัดที่วัดจำนวนของเมท็อดของคลาสที่มีขอบเขตการเข้าถึงเป็นแบบสาธารณะ	QMOOD [18]
Data Access Metric (DAM)	มาตรวัดที่วัดการเข้าถึงข้อมูลของคลาส	QMOOD [18]
Measure of Aggregation (MOA)	มาตรวัดที่วัดจำนวนคุณลักษณะทั้งหมดที่ประกาศไว้ภายในคลาส	QMOOD [18]
Measure of Functional Abstraction (MFA)	มาตรวัดที่วัดอัตราส่วนของจำนวนเมท็อดที่ถูกโอเวอร์ไรด์ต่อจำนวนรวมของเมท็อดที่สามารถเข้าถึงได้โดยสมาชิกของเมท็อดภายในคลาส	QMOOD [18]
Cohesion Among Methods of Class (CAM)	มาตรวัดที่วัดการยึดเหนี่ยวของการเรียกใช้คุณลักษณะภายในคลาส	QMOOD [18]
Coupling Between Methods (CBM)	มาตรวัดที่วัดการเข้าคู่กันระหว่างเมท็อด	Tang [19]

ตารางที่ 1 มาตรวัดทั้งหมดที่ใช้ในงานวิจัย (ต่อ)

มาตรวัด	คำอธิบาย	แหล่งอ้างอิง
Average Method Complexity (AMC)	มาตรวัดที่วัดค่าเฉลี่ยของขนาดของเมท็อดในแต่ละคลาส	Tang [19]
McCabe's cyclomatic complexity (CC)	มาตรวัดที่วัดค่าความซับซ้อนของโปรแกรม	McCabe [20]
Lines of Code (LOC)	มาตรวัดที่นับจำนวนบรรทัดทุกบรรทัดทั้งหมดของโปรแกรม	McCabe [20]

2.1.2 วิธีการจำแนกประเภทด้วยวิธีการเรียนรู้แบบมีผู้สอน (Supervised Learning Techniques)

การจำแนกประเภทด้วยวิธีการเรียนรู้แบบมีผู้สอนเป็นงานทำเหมืองข้อมูลประเภทหนึ่ง โดยตัวอย่างในชุดข้อมูลสอน (training set) ที่ใช้ จะมีคุณลักษณะหนึ่งซึ่งบอกค่าประเภทของตัวอย่างนั้น เราเรียกค่าคุณลักษณะนี้ว่า ฉลากประเภท (class label) ซึ่งเป็นค่าข้อมูลแบบไร้ลำดับ (categorical) โดยวิธีการจำแนกประเภทสามารถแบ่งออกได้เป็น 2 กลุ่ม คือ แบบจำแนกสองฉลากประเภท (binary classification) และจำแนกหลายฉลากประเภท (multiclass classification) ซึ่งในงานวิจัยนี้ได้ศึกษาวิธีการจำแนกประเภทจำนวน 6 วิธี ดังต่อไปนี้

1) ต้นไม้ตัดสินใจ (Decision Tree: DT)

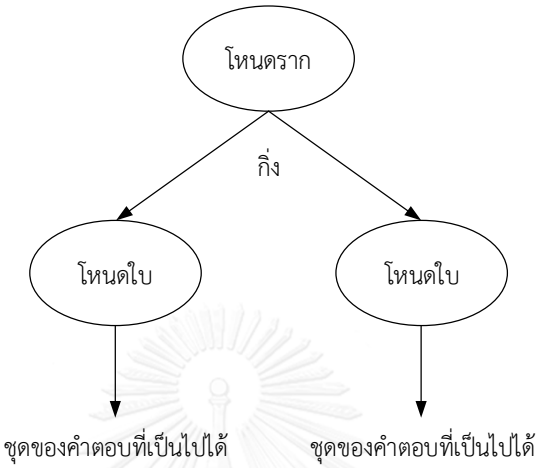
ต้นไม้ตัดสินใจ [22] เป็นการเรียนรู้โดยการจำแนกข้อมูลออกเป็นคลาสต่างๆ โดยใช้คุณลักษณะของข้อมูลในการจำแนกประเภท ต้นไม้ตัดสินใจที่ได้จากการเรียนรู้ทำให้ทราบว่าคุณลักษณะใดของข้อมูลที่เป็นตัวกำหนดการจำแนกประเภท และคุณลักษณะแต่ละตัวมีความสำคัญมากน้อยต่างกันอย่างไรต่อการจำแนกประเภท ซึ่งเป็นประโยชน์ช่วยให้ผู้ใช้สามารถวิเคราะห์ข้อมูลและตัดสินใจได้ถูกต้องมากยิ่งขึ้น

การแทนต้นไม้ตัดสินใจ (Decision Tree Representation) ผลลัพธ์จากการเรียนรู้ต้นไม้ตัดสินใจจะแสดงอยู่ในรูปของต้นไม้ดังแสดงในรูปที่ 1 ซึ่งประกอบด้วยส่วนต่างๆ ดังนี้

- โหนดภายใน (internal node) คือ คุณลักษณะต่างๆ ของข้อมูล ซึ่งเมื่อข้อมูลใดๆตกลงมาที่โหนด จะใช้คุณลักษณะนี้เป็นตัวตัดสินใจว่าข้อมูลจะไปในทิศทางใด โดยโหนดภายในที่เป็นจุดเริ่มต้นข้องต้นไม้ เรียกว่า โหนดราก (root node)

● กิ่ง (branch, link) เป็นค่าของคุณลักษณะในโหนดภายในที่แตกกิ่งนี้ออกมา ซึ่งโหนดภายในจะแตกกิ่งออกมาเป็นจำนวนเท่ากับจำนวนค่าของคุณลักษณะในโหนดภายในเส้น

● โหนดใบ (leaf node) คือ กลุ่มต่างๆ ซึ่งเป็นผลลัพธ์ในการจำแนกประเภทข้อมูล



รูปที่ 1 ส่วนประกอบต่างๆของต้นไม้ตัดสินใจ

2) การเรียนรู้เบย์แบบง่าย (Naïve Bayes: NB)

การเรียนรู้เบย์แบบง่าย [22] เป็นวิธีการจำแนกประเภทข้อมูลที่ง่าย รวดเร็ว และมีประสิทธิภาพวิธีหนึ่ง โดยผลลัพธ์ที่ได้นั้นเทียบได้กับผลลัพธ์จากอัลกอริทึมที่มีความซับซ้อนกว่า เช่น อัลกอริทึมต้นไม้ตัดสินใจ (C4.5) การเรียนรู้เบย์แบบง่ายมีพื้นฐานมาจากทฤษฎีของเบย์ ซึ่งใช้คำนวณหาค่าความน่าจะเป็นที่ข้อมูลน่าจะถูกจำแนกอยู่ในประเภทใด แต่จะลดความซับซ้อนลงโดยเพิ่มสมมติฐานที่ว่า คุณลักษณะต่างๆ ของข้อมูลจะไม่ขึ้นต่อกัน หรือกล่าวได้ว่า ความน่าจะเป็นของข้อมูล X ที่มีคุณลักษณะ n ตัว หรือ $X=\{A_1,\ldots,A_n\}$ จะถูกจำแนกออกเป็นกลุ่ม (class) C_i มีค่าเท่ากับ

$$P(C_i|A_1,\ldots A_n)=\frac{P(A_1,\ldots A_n|C_i)\times P(C_i)}{P(A_1,\ldots A_n)} \qquad \text{[ทฤษฎีของเบย์]} \qquad (1)$$

ขั้นตอนวิธีการเรียนรู้เบย์แบบง่าย (Naïve Bayes Algorithm) ประกอบด้วยขั้นตอนหลัก 2 เฟส คือ เฟสการเรียนรู้ และเฟสการจำแนกประเภท โดยสามารถอธิบายรายละเอียดได้ดังนี้

● เฟสการเรียนรู้ จะทำการประมาณค่าความน่าจะเป็นต่างๆจากชุดข้อมูลสอน ได้แก่ ค่าประมาณความน่าจะเป็นของประเภทต่างๆ หรือ $P'(C_j)$ และค่าประมาณความน่าจะเป็นของแต่ละค่าคุณลักษณะเมื่อรู้ประเภท หรือ $P'(A_j=a_j|C_i)$

ถ้ากำหนดให้ S คือ ขนาดของชุดข้อมูลตัวอย่าง เราสามารถประมาณค่า $P(C_i)$ ด้วย s_i / S โดยที่ s_i เท่ากับจำนวนตัวอย่างที่จัดอยู่ในกลุ่ม C_i และประมาณค่า $P(A_j = a_j | C_i)$ ด้วย s_{ij} / S โดยที่ s_{ij} เท่ากับจำนวนตัวอย่างที่จัดอยู่ในกลุ่ม C_i ซึ่งมีค่าคุณลักษณะ $A_j = a_j$

- เฟสการจำแนกประเภทเป็นการเลือกค่าความน่าจะเป็นมากที่สุดระหว่าง C_i ของตัวอย่างใหม่ที่ทราบค่าคุณลักษณะต่างๆ โดยใช้ทฤษฎีของเบย์ดังสมการ (1) และค่าประมาณต่างๆ ที่คำนวณไว้ล่วงหน้าในเฟสการเรียนรู้บนสมมติฐาน Conditional Independence

3) การเรียนรู้ด้วยเพื่อนบ้านใกล้ที่สุดหรือเคเอ็นเอ็น (k-Nearest Neighbor: k-NN)

การเรียนรู้ด้วยเพื่อนบ้านใกล้ที่สุด หรือ เคเอ็นเอ็น [23] เป็นวิธีการเรียนรู้เชิงอินสแตนซ์ (Instant-Based Learning) หรือ ตัวเรียนรู้เกียจคร้าน (Lazy) วิธีหนึ่ง ซึ่งการเรียนรู้วิธีดังกล่าวจะมองว่าตัวอย่างคือจุดในปริภูมิ n มิติ การจำแนกประเภทตัวอย่างใหม่จะลำเอียงใช้ประเภทของตัวอย่างละแวกใกล้เคียงวัดโดยระยะห่างยูคลิเดียน (Euclidean distance)

นิยามตัวอย่าง x ด้วยเวกเตอร์คุณลักษณะ $< a_1(x), a_2(x), \ldots, a_n(x) >$ โดยที่ $a_r(x)$ แทนค่าคุณลักษณะตัวที่ r ของตัวอย่าง x

นิยามระยะห่างระหว่างตัวอย่าง x_i และ x_j แสดงดังสมการ (2)

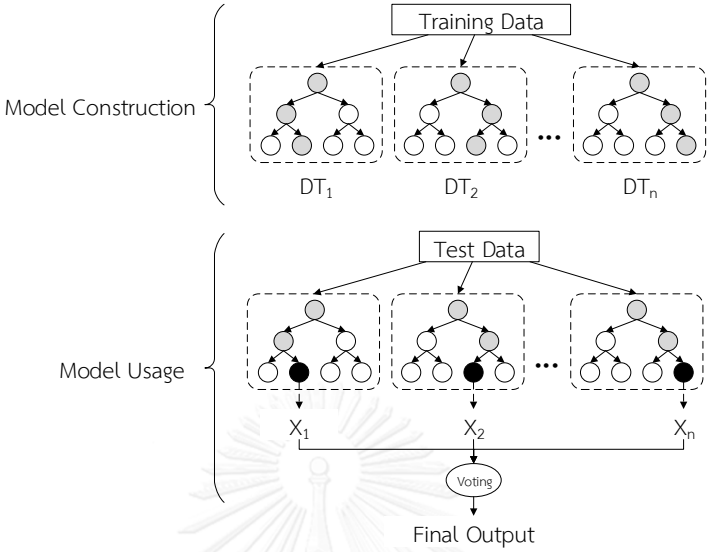
$$d(x_i, x_j) = \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2} \qquad (2)$$

วิธีการจำแนกประเภทเพื่อนบ้านใกล้ที่สุดเหมาะกับชุดข้อมูลสอนที่มีปริมาณมากและตัวอย่างมีคุณลักษณะไม่เกิน 20 คุณลักษณะ เวลาที่ใช้สอนจะรวดเร็วเนื่องจากเป็นการเรียนรู้แบบเกียจคร้าน ต่างกับวิธีการจำแนกประเภทอื่นซึ่งจะใช้เวลานาน แต่ก็มีความอ่อนไหวต่อปัญหาของมิติข้อมูล (Curse of Dimensionality) กล่าวคือ เมื่อชุดตัวอย่างมีคุณลักษณะจำนวนมาก การหาตัวอย่างละแวกใกล้เคียงกับตัวอย่างค้นถามอาจผิดพลาด อันเนื่องมาจากการคำนวณระยะห่างเพื่อหาตัวอย่างใกล้เคียง อาจถูกครอบงำหรือบิดเบือนด้วยค่าคุณลักษณะที่ไม่เกี่ยวข้องจำนวนมาก

4) การเรียนรู้ป่าแบบสุ่ม (Random Forest: RF)

การเรียนรู้แบบป่าสุ่ม [24] เป็นอีกวิธีการหนึ่งที่ได้รับความนิยม เนื่องจากเป็นวิธีการที่ใช้งานง่าย มีความยืดหยุ่นในการจัดการข้อมูลที่มีความหลากหลาย การทำงานจะมีลักษณะคล้ายกับวิธีการ Bagging คือ จะสร้างแบบจำลองที่มีความแตกต่างกันโดยใช้การสุ่มข้อมูลตัวอย่างจากชุดข้อมูลสอน จำนวนหลายๆ ชุด แล้วสร้างแบบจำลองด้วยวิธีการต้นไม้ตัดสินใจ แต่สิ่งที่แตกต่างจากวิธีการ Bagging คือ จะมีการสุ่มคุณลักษณะ (random attribute subsets) เพื่อให้แบบจำลองมี

ความหลากหลายมากขึ้น โดยผลลัพธ์ที่ได้อย่างอิสระจากแบบจำลองแต่ละอัน จะถูกนำมาคิดเป็นผลโหวต โดยผลโหวตที่มากที่สุดจะถูกใช้เป็นคำตอบ ดังแสดงในรูปที่ 2



รูปที่ 2 ขั้นตอนการทำงานของวิธีการเรียนรู้ป่าแบบสุ่ม

5) ซัพพอร์ตเวกเตอร์แมชชีน (Support Vector Machine: SVM)

ซัพพอร์ตเวกเตอร์แมชชีน หรือ เอสวีเอ็ม [23] เป็นหนึ่งในวิธีการจำแนกประเภทที่ได้รับความนิยมมากที่สุด ที่ให้ความถูกต้องและแม่นยำสูงกว่าวิธีการจำแนกประเภทอื่นๆ โดยเฉพาะอย่างยิ่งในงานด้านการจัดหมวดหมู่เอกสาร (text categorization) ซึ่งมักจะนำมาใช้กับข้อมูลที่เป็นเชิงเส้น หลักการของซัพพอร์ตเวกเตอร์แมชชีนก็คือ จะสร้างโมเดลการจำแนกประเภทเชิงเส้นโดยหาระนาบแบ่งเขตข้อมูลที่เหมาะสม (The optimal separating hyperplane) ซึ่งมีระยะห่างระหว่างคลาสสองคลาสที่เราสนใจมากที่สุด ตัวอย่างสอนที่อยู่บริเวณขอบของการกระจายตัวของคลาสบนพื้นที่ข้อมูลคุณลักษณะ (feature space) จะถูกเรียกว่า ซัพพอร์ตเวกเตอร์ (support vector) ดังแสดงในรูปที่ 3

จากสมการ (3) แสดง optimization function ในการสร้างระนาบแบ่งเขตข้อมูลที่เหมาะสม โดยที่ w คือ เวกเตอร์ค่าน้ำหนักของ output และ C คือ ค่าคงที่ที่ใช้สำหรับกำหนดค่าความผิดพลาดในการจำแนกกลุ่มข้อมูล และค่า ξ_i (slack variable) เป็นตัวแปรที่กำหนดขึ้นเพื่อยอมรับค่าความผิดพลาดที่คลาดเคลื่อนไปจากตำแหน่งที่เหมาะสม จากสมการจะพบว่าสมการประกอบด้วยสองพจน์หลัก โดยในพจน์แรกจะเป็นการเพิ่มระยะระหว่างคลาสสองคลาสให้มากที่สุด และพจน์ที่สองเป็นการลดข้อผิดพลาดจากการสอนให้ต่ำที่สุด

$$\phi(w,\xi) = \frac{1}{2} w^T w + C\sum_{i=1}^{n} \xi_i \qquad (3)$$

ในกรณีที่ข้อมูลไม่เป็นเชิงเส้น ซัพพอร์ตเวกเตอร์แมชชีนสามารถแก้ไขปัญหาได้โดยทำการเปลี่ยนแปลงมิติของข้อมูลจากพื้นที่มิติต่ำ (lower dimensional space) ไปยังพื้นที่มิติสูง (higher dimensional space) โดยสร้างฟังก์ชันวัดความคล้ายที่เรียกว่า เคอร์เนลฟังก์ชัน (Kernel Function) ดังแสดงในสมการ (4) เพื่อให้สามารถจำแนกข้อมูลแบบเชิงเส้นได้

$$K(x_i, x_j) = \phi(x_i)\phi(x_j) \tag{4}$$

โดยเคอร์เนลฟังก์ชันที่นิยมใช้มีอยู่ 4 ประเภท ดังแสดงในสมการที่ (5) (6) (7) และ (8) ดังนี้

- เชิงเส้น (Linear):

$$K(x_i, x_j) = x_i \cdot x_j \tag{5}$$

- โพลิโนเมียล (Polynomial):

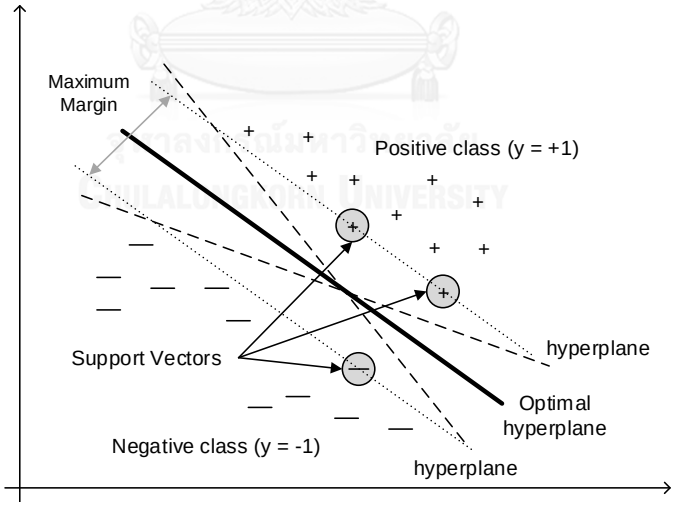
$$K(x_i, x_j) = (x_i \cdot x_j + 1)^d \tag{6}$$

- เกาเซียนเรเดียลเบสิสฟังก์ชัน (Gaussian Radial Basis Function-RBF):

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2) \tag{7}$$

- ซิกมอยด์ (Sigmoid):

$$K(x_i, x_j) = \tanh(\gamma(x_i \cdot x_j) - r) \tag{8}$$



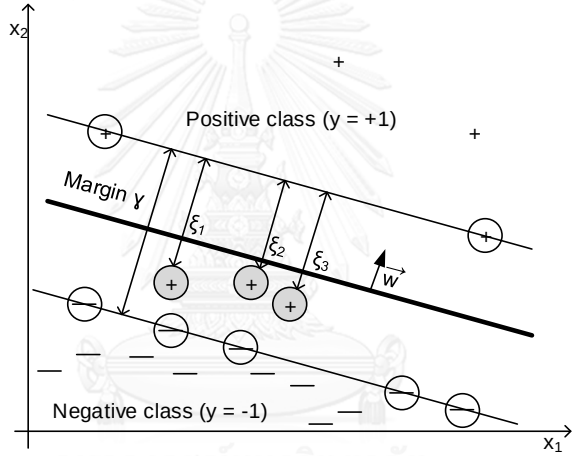
รูปที่ 3 ระนาบแบ่งเขตข้อมูลที่เหมาะสม (Optimal separating hyperplane)

6) อันไบแอสซัพพอร์ตเวกเตอร์แมชชีน (Unbiased Support Vector Machine: R-SVM)

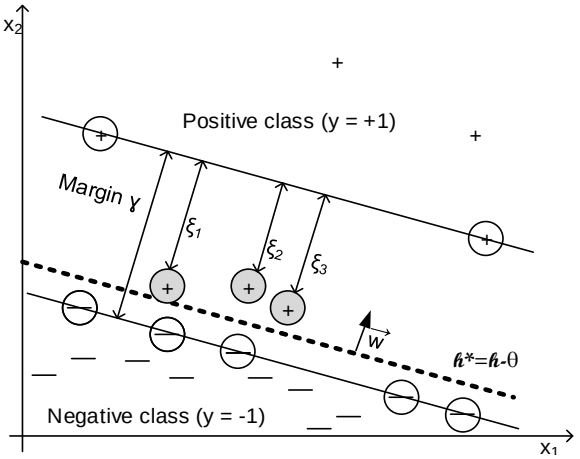
ซัพพอร์ตเวกเตอร์แมชชีนพื้นฐานได้แสดงให้เห็นถึงประสิทธิภาพการทำนายที่ดีในชุดข้อมูลที่สมดุล แต่จะให้ประสิทธิภาพต่ำในกรณีที่ชุดข้อมูลไม่สมดุล อันไบแอสซัพพอร์ตเวกเตอร์แมชชีน หรืออาร์เอสวีเอ็ม [12] เป็นวิธีการที่สามารถจัดการกับปัญหาความไม่สมดุลของชุดข้อมูลได้ ซึ่ง

นำแนวคิดของการสุ่มตัวอย่าง (resampling) มาประยุกต์ใช้กับ R-SVM (R ย่อมาจาก resampling) โดยใช้การปรับแก้ค่าขีดแบ่ง (threshold adjustment) เพื่อลดจำนวนข้อมูลของคลาสที่เป็นเสียงข้างมาก (majority class) ซึ่งจะปรับเลื่อนระนาบแบ่งเขตข้อมูลเพื่อให้ได้รับประสิทธิภาพสูง โดยที่ทิศทางของระนาบแบ่งเขตข้อมูลยังคงไม่เปลี่ยนแปลง รูปที่ 4 แสดงการเลื่อน (shifting) ระนาบแบ่งเขตข้อมูลอันไบแอสเพื่อจำแนกข้อมูลคลาสบวก (positive class) จำนวน 3 ตัวอย่างให้ถูกต้อง ซึ่งระนาบแบ่งเขตข้อมูลแบบเก่า(ไบแอส) จำแนกประเภทผิด

อย่างไรก็ตามการปรับปรุงระนาบตัดสินใจให้ได้รับประสิทธิภาพดีขึ้นสามารถแก้ไขปัญหาความแม่นยำที่เกินความจริง (overfitting) ได้ ซึ่งเป็นปัญหาที่มักพบเจอบ่อยๆ เกิดจากการที่โมเดลจดจำรูปแบบของข้อมูลสอนมากเกินไปจนไม่สามารถทำนายข้อมูลที่ไม่เคยเจอมาก่อน (unknown data) ได้



(ก) ระนาบแบ่งเขตข้อมูลก่อนปรับแก้ค่าขีดแบ่ง



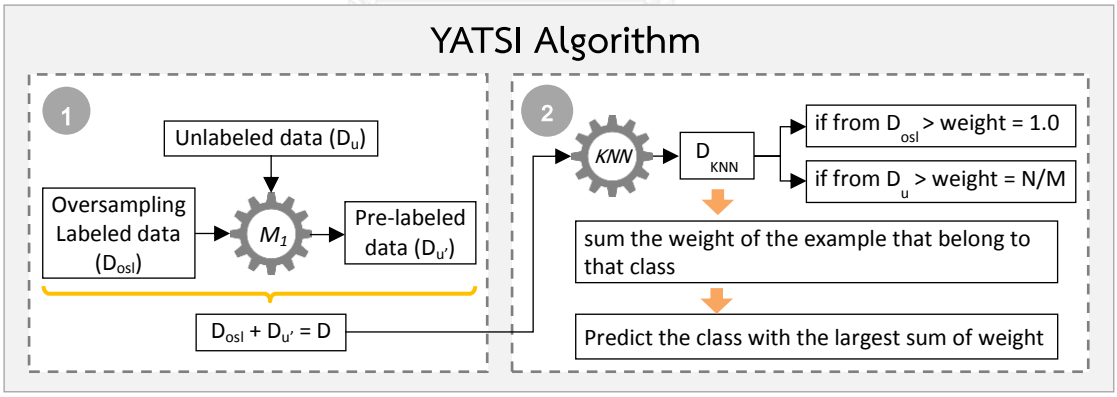
(ข) ระนาบแบ่งเขตข้อมูลหลังปรับแก้ค่าขีดแบ่ง

รูปที่ 4 ระนาบแบ่งเขตข้อมูล (ก) ก่อน และ (ข) หลัง ปรับแก้ค่าขีดแบ่งของการจำแนกข้อมูลตัวอย่างจำนวน 3 ตัวอย่างให้ถูกต้อง [12]

2.1.3 วิธีการจำแนกประเภทด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน (Semi-Supervised Learning Techniques)

การจำแนกประเภทด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอนเป็นงานทำเหมืองข้อมูลที่มีจุดมุ่งหมายเพื่อปรับปรุงประสิทธิภาพการทำนายให้มีความถูกต้องและแม่นยำเพิ่มมากขึ้นโดยการใช้ประโยชน์จากข้อมูลที่มีฉลากประเภทและไม่มีฉลากประเภท กระบวนการทำงานจะเรียนรู้จากข้อมูลสอนที่มีการระบุฉลากประเภทเพียงบางส่วนจากข้อมูลสอนทั้งหมด สำหรับส่วนที่ไม่มีฉลากประเภทนั้นจะใช้กระบวนการเรียนรู้เพื่อระบุฉลากประเภทและปรับความถูกต้องให้กับการเรียนรู้ต่อไป จากการศึกษางานวิจัยที่ผ่านมา [25] พบว่าวิธีการเรียนรู้แบบกึ่งมีผู้สอนแบบดั้งเดิม สามารถแบ่งออกเป็น 4 กลุ่มใหญ่ๆ ประกอบด้วย 1) Self-Training 2) Co-Training 3) Density-based และ 4) Graph-based methods ซึ่งความสำเร็จของวิธีการเรียนรู้แบบกึ่งมีผู้สอนนั้นขึ้นอยู่กับสมมติฐานที่ซ่อนเร้น (underlying assumptions) ของแต่ละแบบจำลอง

งานวิจัยนี้ใช้วิธีการเรียนรู้แบบกึ่งมีผู้สอนที่จัดอยู่ในกลุ่ม Self-Training เนื่องจากเป็นกลุ่มที่ได้รับความนิยมอย่างแพร่หลาย ใช้งานง่าย มีความยืดหยุ่นและสามารถประยุกต์ใช้งานกับวิธีการจำแนกประเภทอื่นๆได้เป็นอย่างดี โดยประยุกต์ใช้อัลกอริทึม YATSI [14] ในการสร้างแบบจำลองการจัดลำดับข้อบกพร่องของซอฟต์แวร์ตามระดับความรุนแรง ซึ่งมีกระบวนการทำงาน 2 ขั้นตอน ดังแสดงในรูปที่ 5 สามารถอธิบายรายละเอียดในแต่ละขั้นตอนได้ดังนี้



รูปที่ 5 ขั้นตอนการทำงานของอัลกอริทึม YATSI

1) สร้างแบบจำลองเริ่มต้น M_1 โดยใช้ข้อมูลที่มีฉลากประเภท (labeled data) ที่ผ่านการสุ่มตัวอย่างแล้วจากขั้นตอนที่ (1) จากนั้นใช้แบบจำลอง M_1 ทำนายหาฉลากประเภทเทียม (pre-labeled data) ของข้อมูลที่ไม่มีฉลากประเภททุกตัว (unlabeled data) แล้วรวมข้อมูลนี้กับข้อมูลที่มีฉลากประเภทที่ผ่านการสุ่มตัวอย่างแล้ว เป็นข้อมูลใหม่ (D) เพื่อใช้ในการหาฉลากประเภทที่แท้จริงในขั้นตอนต่อไป

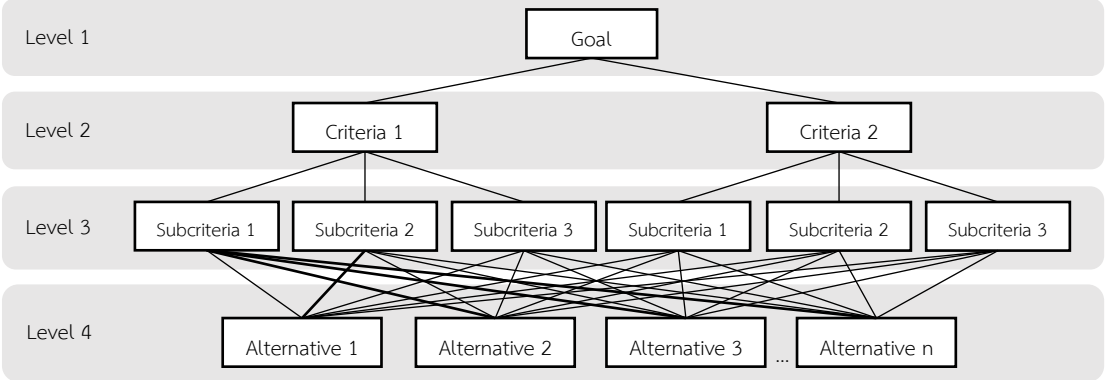
2) สำหรับทุกๆตัวอย่างในข้อมูลที่มีระดับความรุนแรงเทียม ($D_{u'}$) จะคำนวณหาเพื่อนบ้านที่ใกล้เคียงที่สุดกับตัวอย่างในข้อมูลใหม่ (D) เพื่อสร้างกลุ่มข้อมูล D_{KNN} สำหรับทำนายระดับความรุนแรงที่แท้จริงของข้อมูลที่ไม่มีฉลากประเภท โดยมีการให้น้ำหนักตามเงื่อนไข คือ หากตัวอย่างในข้อมูล D_{KNN} มาจากข้อมูลตั้งต้นที่มีฉลากประเภท (D_{osl}) ให้ค่าน้ำหนักเท่ากับ 1 แต่ถ้าตัวอย่างในข้อมูล D_{KNN} มาจากข้อมูลที่ไม่มีฉลากประเภท (D_u) ให้ค่าน้ำหนักเท่ากับ N/M โดยที่ N คือ จำนวนข้อมูลที่มีฉลากประเภท และ M คือ จำนวนข้อมูลที่ไม่มีฉลากประเภท หลังจากนั้นสามารถหาระดับความรุนแรงที่แท้จริงของแต่ละตัวอย่างในข้อมูลที่ไม่มีฉลากประเภทได้ โดยคำนวณหาผลรวมของค่าน้ำหนักของความรุนแรงทุกๆระดับในข้อมูล D_{KNN} ตามเงื่อนไขที่ได้กล่าวไปข้างต้น หากระดับความรุนแรงใดที่มีค่าผลรวมของน้ำหนักมากที่สุด หมายความว่า ระดับความรุนแรงนั้นคือระดับความรุนแรงที่แท้จริงของตัวอย่างนั้นๆ

2.1.4 กระบวนการลำดับชั้นเชิงวิเคราะห์ (Analytic Hierarchy Process)

กระบวนการลำดับชั้นเชิงวิเคราะห์ (Analytic Hierarchy Process) [13] หมายถึง กระบวนการที่ใช้ในการวัดค่าระดับของการตัดสินใจในเรื่องต่างๆ โดยการหาคำตอบที่เหมาะสมที่สุดและตรงตามเป้าหมายของการตัดสินใจ ซึ่งเป็นวิธีการที่นิยมใช้งานอย่างกว้างขวาง เนื่องจากมีความง่ายและสะดวกในการใช้งาน โดยขั้นตอนการทำงานของกระบวนการลำดับชั้นเชิงวิเคราะห์ประกอบด้วย 3 ขั้นตอนหลัก ดังต่อไปนี้

1) กำหนดโครงสร้างลำดับชั้นในการวิเคราะห์ (Defining hierarchical structure)

ในการวิเคราะห์เพื่อตัดสินใจเลือกทางเลือกที่ดีที่สุด จะแบ่งการวิเคราะห์ออกเป็นลำดับชั้น โดยในแต่ละระดับอาจมีหลายเกณฑ์การตัดสินใจ และในแต่ละเกณฑ์การตัดสินใจอาจมีหลายเกณฑ์การตัดสินใจย่อย ดังแสดงในรูปที่ 6 ซึ่งสามารถอธิบายรายละเอียดของลำดับชั้นได้ดังต่อไปนี้



รูปที่ 6 แผนภูมิโครงสร้างลำดับชั้นเชิงวิเคราะห์

- ระดับที่ 1 เป้าหมาย (Goal) คือ การแจ้งถึงวัตถุประสงค์ของปัญหาการตัดสินใจ
- ระดับที่ 2 เกณฑ์การตัดสินใจ (Criteria) คือ สิ่งที่สามารถทำให้เป้าหมายบรรลุตามวัตถุประสงค์ที่กำหนดขึ้น
- ระดับที่ 3 เกณฑ์การตัดสินใจย่อย (Sub criteria) คือ สิ่งที่สามารถทำให้เป้าหมายบรรลุตามวัตถุประสงค์ที่กำหนดขึ้น เพื่อใช้ตัดสินใจหาทางเลือกที่เหมาะสมที่สุดสำหรับวัตถุประสงค์นั้นๆ
- ระดับที่ 4 ทางเลือก (Alternatives) คือ สิ่งที่ต้องการเลือกเพื่อให้เกิดประโยชน์สูงสุดและตรงตามเป้าหมายของการตัดสินใจ

2) คำนวณหาลำดับความสำคัญ (Calculating priority of the hierarchy)

หลังจากมีการกำหนดโครงสร้างลำดับชั้นเรียบร้อยแล้วต้องมีการประเมินความสำคัญของทางเลือกและเกณฑ์การตัดสินใจต่างๆ โดยใช้การเปรียบเทียบความสัมพันธ์ทีละคู่ (Pairwise Comparison) จากปัจจัยที่มีผลกระทบต่อเกณฑ์การตัดสินใจในแต่ละระดับชั้นจากบนลงล่าง (Top-Down) จนครบทุกเกณฑ์ โดยมาตราส่วนในการเปรียบเทียบความสำคัญ (Measurement Scale) แบ่งออกเป็น 9 ระดับ ดังแสดงในตารางที่ 2

ตารางที่ 2 มาตราส่วนในการเปรียบเทียบความสำคัญ [26]

ระดับ	ความหมาย
1	เท่ากัน (Equal Importance)
2	เท่ากันถึงปานกลาง (Weak or slight)
3	ปานกลาง (Moderate Importance)
4	ปานกลางถึงค่อนข้างมาก (Moderate Plus)
5	ค่อนข้างมาก (Strong Importance)
6	ค่อนข้างมากถึงมากกว่า (Strong Plus)
7	มากกว่า (Demonstrated Importance)
8	มากกว่าถึงมากที่สุด (Demonstrated Plus)
9	มากที่สุด (Extreme Importance)

การคำนวณหาค่าลำดับความสำคัญของเกณฑ์การตัดสินใจแต่ละเกณฑ์ก่อนที่จะประเมินความสำคัญของทางเลือก จะประกอบด้วยขั้นตอนย่อย คือ

(1) การให้ค่าน้ำหนักความสำคัญของเกณฑ์การตัดสินใจ

สามารถทำได้โดยการสร้างตารางเมตริกซ์เปรียบเทียบเกณฑ์การตัดสินใจแบบทีละคู่ (Pairwise Comparison Matrix) ดังตารางที่ 3

ตารางที่ 3 ตารางเมตริกซ์เปรียบเทียบเกณฑ์การตัดสินใจทีละคู่

เป้าหมาย		เกณฑ์การตัดสินใจ				
		A_1	A_2	A_3	...	A_n
เกณฑ์การตัดสินใจ	A_1	a_{11}	a_{12}	a_{13}	...	a_{1n}
	A_2	a_{21}	a_{22}	a_{23}	...	a_{2n}
	A_3	a_{31}	a_{32}	a_{33}	...	a_{3n}
	⋮	⋮	⋮	⋮	⋱	⋮
	A_n	a_{1n}	a_{2n}	a_{3n}	...	a_{nn}

โดยที่ a_{ij} คือ สมาชิกในแถวที่ i หลักที่ j ของเมตริกซ์ หมายถึง ผลการเปรียบเทียบความสำคัญระหว่างเกณฑ์การตัดสินใจ A_i และ A_j ซึ่งสามารถกำหนดมาตราส่วนในการเปรียบเทียบความสำคัญให้แต่ละเกณฑ์ตัดสินใจได้โดยพิจารณาค่าระดับจากตารางที่ 2

<u>ตัวอย่างการให้ค่าน้ำหนักความสำคัญของเกณฑ์การตัดสินใจ</u>

สมมติให้ ระดับความรุนแรงมีเกณฑ์การตัดสินใจย่อย 2 เกณฑ์ คือ Low และ High และมีมาตราส่วนในการเปรียบเทียบความสำคัญอยู่ในระดับ 7 (มากกว่า)

ดังนั้น สามารถสร้างตารางเมตริกซ์เปรียบเทียบเกณฑ์การตัดสินใจทีละคู่ได้ดังตารางที่ 4

ตารางที่ 4 ตัวอย่างตารางเมตริกซ์เปรียบเทียบเกณฑ์การตัดสินใจทีละคู่

Severity		Criteria	
		Low	High
Criteria	Low	1	1/7
	High	7	1

- ถ้าเกณฑ์การตัดสินใจ A_i (แถว) และ A_j (คอลัมน์) มีความสำคัญเท่ากัน จะได้ $a_{ij}=1$
- ถ้าเกณฑ์การตัดสินใจ A_i (แถว) มีความสำคัญมากกว่า A_j (คอลัมน์) จะได้ $a_{ij}=7$
- ถ้าเกณฑ์การตัดสินใจ A_i (แถว) มีความสำคัญน้อยกว่า A_j (คอลัมน์) จะได้ $a_{ij}=\frac{1}{7}$

(2) คำนวณหาค่าน้ำหนักความสำคัญของเกณฑ์การตัดสินใจ

- ปรับผลรวมของแต่ละคอลัมน์ให้มีค่าเท่ากับ 1 (Normalize) โดยนำตัวเลขในแต่ละคอลัมน์หารด้วยผลรวมของทุกคอลัมน์นั้นๆ
- คำนวณหาค่าน้ำหนัก (Eigenvector) ของเกณฑ์การตัดสินใจ โดยหาผลรวมในแต่ละแถวและหารผลรวมดังกล่าวด้วยจำนวนเกณฑ์ที่ใช้ในการตัดสินใจ

(3) เปรียบเทียบทางเลือกที่กำหนดไว้ผ่านเกณฑ์การตัดสินใจ เพื่อจัดลำดับความสำคัญของทางเลือก

คำนวณหาค่าน้ำหนักความสำคัญโดยรวมของแต่ละทางเลือกที่กำหนดไว้ โดยคำนวณจากผลรวมของผลคูณระหว่างค่าน้ำหนักของแต่ละทางเลือกในแต่ละเกณฑ์การตัดสินใจกับค่าน้ำหนักของเกณฑ์การตัดสินใจย่อยนั้นๆ

ค่าน้ำหนักความสำคัญโดยรวม = ผลรวมของ (ค่าน้ำหนักของเกณฑ์การตัดสินใจ × ค่าน้ำหนักของเกณฑ์การตัดสินใจย่อย)

ดังนั้น สามารถเรียงลำดับความสำคัญของทางเลือกทั้งหมดจากมากไปน้อย ซึ่งทางเลือกที่มีค่าน้ำหนักความสำคัญมากที่สุดจะเป็นทางเลือกที่ดีที่สุด

3) ตรวจสอบความสอดคล้องของข้อมูล (Validating consistency of the data)

ในการวิเคราะห์เปรียบเทียบความสัมพันธ์แบบทีละคู่ อาจเกิดข้อผิดพลาดหรือความไม่สมเหตุสมผลกันในการให้ค่าน้ำหนักเกณฑ์การตัดสินใจ จึงจำเป็นต้องมีการตรวจสอบความสอดคล้องกันของข้อมูล โดยการพิจารณาอัตราส่วนความสอดคล้องของข้อมูล (Consistency Ratio: C.R.) ซึ่งเป็นอัตราส่วนระหว่างดัชนีความสอดคล้องของข้อมูล (Consistency Index: CI) และดัชนีความสอดคล้องของข้อมูลเชิงสุ่ม (Random Consistency Index: RI) ดังแสดงในสมการ (9)

$$CR = \frac{CI}{RI} \tag{9}$$

โดยที่ ดัชนีความสอดคล้องของข้อมูลสามารถคำนวณได้ดังสมการ (10)

$$CI = \frac{\lambda_{\max} - n}{n-1} \tag{10}$$

เมื่อ λ_{max} คือ ผลรวมของผลคูณระหว่างค่าน้ำหนักความสำคัญของเกณฑ์การตัดสินใจในแต่ละคอลัมน์จากตารางเมตริกซ์เปรียบเทียบเกณฑ์การตัดสินใจแบบทีละคู่กับค่าน้ำหนัก (Eigenvector) ของเกณฑ์การตัดสินใจในแต่ละแถว

n คือ จำนวนเกณฑ์การตัดสินใจ

ส่วนค่าดัชนีความสอดคล้องของข้อมูลเชิงสุ่มจะขึ้นอยู่กับขนาดของตารางเมตริกซ์เปรียบเทียบเกณฑ์การตัดสินใจแบบทีละคู่ ดังแสดงในตารางที่ 5

ตารางที่ 5 ดัชนีความสอดคล้องของข้อมูลเชิงสุ่ม (Random Consistency Index: RI)

n	RI	n	RI	n	RI
1	0.00	6	1.24	11	1.51
2	0.00	7	1.32	12	1.48
3	0.58	8	1.41	13	1.56
4	0.90	9	1.45	14	1.57
5	1.12	10	1.49	15	1.59

อัตราส่วนความสอดคล้องของข้อมูลที่ได้จะถูกพิจารณาว่าค่าน้ำหนักความสำคัญที่กำหนดให้กับเกณฑ์การตัดสินใจมีความสมเหตุสมผลหรือไม่ตามเงื่อนไขดังต่อไปนี้

- ถ้า $CR < 0.1$ แสดงว่าค่าน้ำหนักความสำคัญของเกณฑ์การตัดสินใจมีความสอดคล้องกัน
- ถ้า $CR > 0.1$ แสดงว่าค่าน้ำหนักความสำคัญของเกณฑ์การตัดสินใจไม่มีความสอดคล้องกัน จะต้องปรับค่าน้ำหนักความสำคัญใหม่

การตรวจสอบความสอดคล้องของข้อมูล จะทำการตรวจสอบทุกระดับชั้นจนถึงระดับสุดท้ายเพื่อให้มั่นใจได้ว่าน้ำหนักความสำคัญที่กำหนดให้กับเกณฑ์การตัดสินใจมีความน่าเชื่อถือ

2.1.5 มาตรฐาน IEEE 610.12 อภิธานศัพท์ทางด้านวิศวกรรมซอฟต์แวร์ (IEEE 610.12 Standard Glossary of Software Engineering Terminology)

มาตรฐาน IEEE 610.12 [27] เป็นมาตรฐานที่ระบุคำศัพท์ที่นิยมใช้ในงานทางด้านวิศวกรรมซอฟต์แวร์ในปัจจุบันและให้คำนิยามมาตรฐานของคำศัพท์เหล่านี้ เพื่อให้ผู้ใช้สามารถสื่อสารและนำไปใช้งานได้อย่างถูกต้อง

ในงานวิจัยนี้ได้สนใจนิยามของคำศัพท์ คือ ข้อบกพร่อง (Defect) และความรุนแรง (Severity) จึงได้มีการนิยามคำศัพท์ตามบริบทของงานวิจัยที่สอดคล้องกับนิยามตามมาตรฐาน IEEE 610.12 ดังแสดงในตารางที่ 6

ตารางที่ 6 นิยามของคำศัพท์ตามมาตรฐาน IEEE 610.12 [27] และบริบทที่ใช้ในงานวิจัยนี้

คำศัพท์	นิยามตามมาตรฐาน IEEE 610.12	นิยามตามบริบทของงานวิจัย
1. ข้อบกพร่อง (Defect)	- ข้อบกพร่อง (Defect, Fault) ได้มีการนิยามไว้ 2 ความหมาย ดังนี้ 1) ข้อบกพร่องในอุปกรณ์หรือส่วนประกอบฮาร์ดแวร์ ตัวอย่างเช่น ไฟฟ้าลัดวงจร 2) ขั้นตอน กระบวนการ หรือนิยามข้อมูลที่ไม่ถูกต้องในโปรแกรมคอมพิวเตอร์	- ข้อบกพร่องที่เกิดขึ้นในรหัสต้นฉบับ (Source Code) ในกระบวนการพัฒนาซอฟต์แวร์
2. ความรุนแรง (Severity)	- ระดับความรุนแรง (Severity, Criticality) คือ ระดับของผลกระทบของข้อบกพร่องที่ส่งผลกระทบกับความต้องการ โมดูล ข้อผิดพลาด ข้อบกพร่อง ความล้มเหลว หรือรายการอื่นๆในการพัฒนาหรือการทำงานของระบบ	- ระดับของผลกระทบการขึ้นต่อกันของคลาสที่เกิดข้อผิดพลาดที่ส่งผลกระทบต่อการทำงานของซอฟต์แวร์

2.1.6 มาตรฐาน IEEE 1044 การจำแนกประเภทสำหรับสิ่งผิดปกติในซอฟต์แวร์ (IEEE 1044 Standard Classification for Software Anomalies)

มาตรฐาน IEEE 1044 [28] เป็นมาตรฐานที่ระบุถึงการจำแนกประเภทของความล้มเหลวและข้อบกพร่องโดยมีการกำหนดชุดของคุณลักษณะของความล้มเหลวและข้อบกพร่องที่มีความสำคัญเพื่ออำนวยความสะดวกให้ผู้ใช้สามารถจัดการกับปัญหาที่เกิดขึ้นในกระบวนการพัฒนาและบำรุงรักษาซอฟต์แวร์ได้อย่างมีประสิทธิภาพ

ในงานวิจัยนี้สนใจคุณลักษณะของข้อบกพร่อง คือ ระดับความรุนแรงของข้อบกพร่อง (Defect Severity) จึงได้มีการนิยามระดับความรุนแรงของข้อบกพร่องตามบริบทของงานวิจัยที่สอดคล้องกับนิยามตามมาตรฐาน IEEE 1044 ดังแสดงรายละเอียดในตารางที่ 7

จากตารางที่ 7 จะพบว่านิยามระดับความรุนแรงของข้อบกพร่องตามมาตรฐาน IEEE 1044 ได้แบ่งระดับความรุนแรงออกเป็น 5 ระดับ คือ ไม่รุนแรง รุนแรงน้อย รุนแรง วิกฤต และบล็อกกิ้ง ซึ่งแตกต่างจากงานวิจัยนี้ที่แบ่งระดับความรุนแรงออกเป็น 3 ระดับ โดยการพิจารณาจากค่าผลกระทบการขึ้นต่อกันของคลาส ซึ่งมีช่วงของค่าผลกระทบการขึ้นต่อกันของคลาสของความรุนแรงแต่ละระดับที่แตกต่างกัน คือ ระดับความรุนแรงน้อย จะมีค่าผลกระทบการขึ้นต่อกันของคลาสอยู่ในช่วง 0.00 - 0.33 ระดับความรุนแรงปานกลาง จะมีค่าผลกระทบการขึ้นต่อกันของคลาสอยู่ในช่วง 0.34 - 0.66

และระดับความรุนแรงมาก จะมีค่าผลกระทบการขึ้นต่อกันของคลาสอยู่ในช่วง 0.67 – 1.00 นอกจากนี้สาเหตุอีกประการที่แบ่งความรุนแรงออกเป็น 3 ระดับ คือ เพื่อให้ง่ายต่อการตัดสินใจในการแก้ไขและสอดคล้องกับชุดข้อมูลที่ใช้ในการจัดลำดับความสำคัญของข้อบกพร่องด้วยวิธีการเรียนรู้แบบกึ่งผู้สอนที่มีข้อมูลระดับความรุนแรง 3 ระดับเช่นกัน

ตารางที่ 7 นิยามระดับความรุนแรงของข้อบกพร่องตามมาตรฐาน IEEE 1044 [28] และบริบทที่ใช้ในงานวิจัยนี้

คุณลักษณะ	นิยามตามมาตรฐาน IEEE 1044	นิยามตามบริบทของงานวิจัย
ระดับความรุนแรงของข้อบกพร่อง (defect Severity)	- มีการนิยามไว้ 5 ระดับ ดังนี้ 1) ไม่รุนแรง หมายถึง ไม่มีผลกระทบอย่างมีนัยสำคัญในการดำเนินงาน 2) รุนแรงน้อย หมายถึง การดำเนินงานที่ไม่จำเป็นจะหยุดชะงัก 3) รุนแรง หมายถึง การดำเนินงานที่จำเป็นได้รับผลกระทบ แต่สามารถดำเนินการต่อไปได้ 4) วิกฤต หมายถึง การดำเนินงานที่จำเป็นจะหยุดชะงักอย่างหลีกเลี่ยงไม่ได้ ความปลอดภัยลดลง 5) บล็อกกิ้ง หมายถึง การทดสอบจะถูกระงับหรือรอการแก้ไขปัญหาอย่างเหมาะสม	- มีการนิยามไว้ 3 ระดับ ดังนี้ 1) รุนแรงน้อย หมายถึง ผลกระทบการขึ้นต่อกันของคลาสที่เกิดข้อผิดพลาดมีค่าอยู่ในช่วง 0.00 - 0.33 2) รุนแรงปานกลาง หมายถึง ผลกระทบการขึ้นต่อกันของคลาสที่เกิดข้อผิดพลาดมีค่าอยู่ในช่วง 0.34 - 0.66 3) รุนแรงมาก หมายถึง ผลกระทบการขึ้นต่อกันของคลาสที่เกิดข้อผิดพลาดมีค่าอยู่ในช่วง 0.67 – 1.00

2.1.7 มาตรวัดประสิทธิภาพการทำนาย (Prediction Performance Metrics)

ในหัวข้อนี้จะกล่าวถึงมาตรวัดประสิทธิภาพการทำนาย ซึ่งใช้ในการประเมินประสิทธิภาพของแบบจำลองการทำนายข้อบกพร่องและแบบจำลองการจัดลำดับข้อบกพร่องด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน โดยในการวัดประสิทธิภาพของแบบจำลองการจำแนกประเภทข้อมูลนั้น จะอาศัยเมตริกซ์ความสับสน (Confusion Matrix) ในการเก็บข้อมูลเกี่ยวกับการจำแนกข้อมูลจริงกับข้อมูลที่เกิดจากการทำนาย ดังแสดงในตารางที่ 8

ตารางที่ 8 เมตริกซ์ความสับสน (Confusion Matrix)

		ค่าที่ทำนาย (Predicted)	
		จริง (Positive)	เท็จ (Negative)
ค่าจริง (Actual)	จริง (Positive)	TP	FN
	เท็จ (Negative)	FP	TN

โดยที่ค่าต่างๆในเมตริกซ์ความสับสนมีความหมายดังนี้

- True Positive (TP) คือ ค่าที่โปรแกรมทำนายว่าจริง และค่าจริงเป็นจริง
- True Negative (TN) คือ ค่าที่โปรแกรมทำนายว่าเท็จ และค่าจริงเป็นเท็จ
- False Positive (FP) คือ ค่าที่โปรแกรมทำนายว่าจริง และค่าจริงเป็นเท็จ
- False Negative (FN) คือ ค่าที่โปรแกรมทำนายว่าเท็จ และค่าจริงเป็นจริง

ในงานวิจัยนี้ ค่าจากเมตริกซ์ความสับสนเหล่านี้จะถูกนำมาใช้คำนวณหาค่ามาตรวัดประสิทธิภาพการทำนาย ซึ่งประกอบไปด้วย ความเที่ยงตรง (Precision), ความครบถ้วน (Probability of Detection), ความน่าจะเป็นของการเตือนผิดพลาด (Probability of False Alarm), อัตราความถูกต้องเชิงลบ (True Negative Rate) [29], มัชฌิมฮาร์โมนิค (F-measure) [16] และมัชฌิมเรขาคณิต (G-mean) [30], ซึ่งเป็นมาตรวัดที่เหมาะสมที่นิยมใช้ในการแก้ไขปัญหาความไม่สมดุลของคลาสในชุดข้อมูล ดังแสดงรายละเอียดในตารางที่ 9

ตารางที่ 9 นิยามและสูตรการคำนวณของมาตรวัดประสิทธิภาพการทำนาย

มาตรวัด	นิยาม	สูตรการคำนวณ
Precision (Pr)	ค่าที่บ่งบอกว่าโปรแกรมทำนายว่าจริงถูกต้องเท่าไร	$\frac{TP}{TP+FP}$
Probability of Detection (PD), Recall, TPR	ค่าที่บ่งบอกว่าโปรแกรมทำนายว่าจริง เป็นอัตราส่วนเท่าไรของจริงทั้งหมด	$\frac{TP}{TP+FN}$
Probability of False Alarm (PF), FPR	ค่าที่บ่งบอกว่าโปรแกรมทำนายว่าจริง เป็นอัตราส่วนเท่าไรของไม่จริงทั้งหมด	$\frac{FP}{TN+FP}$
True Negative Rate (TNR)	ค่าที่บ่งบอกว่าโปรแกรมทำนายว่าไม่จริงเป็นอัตราส่วนเท่าไรของจริงทั้งหมด	$\frac{TN}{TN+FP}$
F-measure	ค่าเฉลี่ยที่ให้ความสำคัญกับค่า precision และ recall เท่าๆกัน	$\frac{2\times Pr\times Re}{Pr+Re}$
G-mean	รากของผลคูณระหว่าง TPR และ TNR	$\sqrt{(TPR)\times(TNR)}$

จากตารางที่ 9 สามารถรวมมาตรวัดเหล่านี้เข้าด้วยกัน ซึ่งสามารถแบ่งได้ 2 กลุ่ม คือ ค่าเฉลี่ยมหภาค (Macro-averaging) และ ค่าเฉลี่ยจุลภาค (Micro-averaging) [31] ค่าเฉลี่ยมหภาค (Macro-averaging) เป็นการวัดประสิทธิภาพด้วยค่าเฉลี่ยแบบให้น้ำหนักทุกคลาสเท่ากัน โดยจะคำนวณหาค่าความถูกต้องของแต่ละคลาสก่อน จากนั้นจึงนำมาเฉลี่ยเพื่อให้ได้ค่าความถูกต้องโดยรวม ในขณะที่ค่าเฉลี่ยจุลภาค (Micro-averaging) เป็นการวัดประสิทธิภาพด้วยค่าเฉลี่ยแบบให้น้ำหนักทุกตัวอย่างเท่ากัน โดยจะคำนวณหาค่าความถูกต้องจากจำนวนตัวอย่าง ซึ่งจะนำจำนวนตัวอย่างของแต่ละคลาสมารวมกัน เพื่อคำนวณหาค่าความถูกต้อง ดังแสดงรายละเอียดในตารางที่ 10

ตารางที่ 10 สมการแสดงการคำนวณค่าความเที่ยงตรง ความครบถ้วน และมัชฌิมฮาร์โมนิค แบบค่าเฉลี่ยมหภาคและจุลภาค

มาตรวัด	ค่าเฉลี่ยมหภาค	ค่าเฉลี่ยจุลภาค
Precision	$MaPr = \frac{1}{\lvert L \rvert}\sum_{i=1}^{\lvert L \rvert} Pr_i$	$MiPr = \frac{\sum_{i=1}^{\lvert L \rvert} tp_i}{\sum_{i=1}^{\lvert L \rvert}(tp_i + fp_i)}$
Recall	$MaRe = \frac{1}{\lvert L \rvert}\sum_{i=1}^{\lvert L \rvert} Re_i$	$MiRe = \frac{\sum_{i=1}^{\lvert L \rvert} tp_i}{\sum_{i=1}^{\lvert L \rvert}(tp_i + fn_i)}$
F_β-measure	$MaF_\beta = \frac{1}{\lvert L \rvert}\sum_{i=1}^{\lvert L \rvert} F_{\beta,i}$	$MiF_\beta = \frac{(\beta^2+1)\times MiPr \times MiRe}{\beta^2 \times MiPr + MiRe}$

* โดยที่ i คือ ดัชนีของคลาส (class index)

2.2 งานวิจัยที่เกี่ยวข้อง

ในหัวข้อนี้จะอธิบายเกี่ยวกับการศึกษาค้นคว้างานวิจัยที่เกี่ยวข้อง ซึ่งเกี่ยวข้องกับการทำนายข้อบกพร่องและทำนายระดับความรุนแรงของข้อบกพร่องในซอฟต์แวร์ เพื่อศึกษาแนวคิดและวิธีการของงานวิจัยเหล่านี้ และนำมาประยุกต์ใช้เป็นแนวทางในการสร้างกรอบงานสำหรับตรวจจับและจัดลำดับข้อบกพร่องที่ได้นำเสนอให้มีประสิทธิภาพมากยิ่งขึ้น โดยสามารถแบ่งงานวิจัยที่ได้ศึกษาออกเป็น 2 กลุ่ม คือ กลุ่มงานวิจัยด้านการทำนายข้อบกพร่องขอซอฟต์แวร์ และกลุ่มงานวิจัยด้านการทำนายระดับความรุนแรงของข้อบกพร่องในซอฟต์แวร์ นอกจากนี้ยังได้เปรียบเทียบความแตกต่างของงานวิจัยที่เกี่ยวข้องกับงานวิจัยที่นำเสนอ โดยอธิบายรายละเอียดได้ดังนี้

2.2.1 กลุ่มงานวิจัยด้านการทำนายข้อบกพร่องของซอฟต์แวร์

งานวิจัยกลุ่มนี้นำเสนอวิธีการในการสร้างแบบจำลองสำหรับทำนายข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์ โดยประยุกต์ใช้การทำเหมืองข้อมูลและการเรียนรู้ของเครื่องจักรด้วยวิธีการต่างๆ ซึ่งสามารถอธิบายรายละเอียดแต่ละงานวิจัยได้ดังนี้

2.2.1.1 การทำนายแนวโน้มการเกิดข้อบกพร่องในซอฟต์แวร์โมดูลโดยใช้เทคนิคซัพพอร์ตเวกเตอร์แมชชีน (Predicting defect-prone software modules using support vector machines)

งานวิจัยนี้ [1] มีจุดประสงค์เพื่อประเมินความสามารถของวิธีการซัพพอร์ตเวกเตอร์แมชชีนในการทำนายข้อบกพร่องในซอฟต์แวร์โมดูล โดยผู้วิจัยได้เปรียบเทียบประสิทธิภาพการทำนายระหว่างวิธีซัพพอร์ตเวกเตอร์แมชชีนกับวิธีจำแนกประเภทอื่นๆอีกจำนวน 8 วิธี ได้แก่ ต้นไม้ตัดสินใจป่าสุ่ม การเรียนรู้เบย์แบบง่าย ข่ายงานความเชื่อเบย์ เคเนียร์เรสเนเบอร์ การถดถอยโลจิสติก ข่ายงานประสาทเทียม และเพอร์เซปตรอนหลายชั้น โดยทำการทดลองบนชุดข้อมูลเอ็มดีพี ซึ่งผลการทดลองแสดงให้เห็นว่าวิธีซัพพอร์ตเวกเตอร์แมชชีนให้ประสิทธิภาพดีกว่าวิธีการจำแนกประเภทอื่นๆที่นำมาเปรียบเทียบ จากงานวิจัยแสดงให้เห็นว่าวิธีซัพพอร์ตเวกเตอร์แมชชีนเป็นวิธีการที่มีประสิทธิภาพในการทำนายข้อบกพร่องบนชุดข้อมูลเอ็มดีพี

2.2.1.2 การทำนายข้อบกพร่องในซอฟต์แวร์การประกันภัยสูง (Predicting Faults in High Assurance Software)

งานวิจัยนี้ [2] มีจุดประสงค์เพื่อลดจำนวนข้อบกพร่องในระบบซอฟต์แวร์การประกันภัยสูง โดยได้เสนออัลกอริทึมที่เรียกว่า รัฟลีบาลานซ์แบกกิ้ง หรือ อาร์บีแบก (Roughly Balanced Bagging: RBBag) ซึ่งทำงานโดยอาศัยแนวคิดของการทำแบกกิ้งกับตัวจำแนกประเภทสองชนิด คือ การเรียนรู้เบย์แบบง่าย และต้นไม้ตัดสินใจ โดยมีมาตรวัดประสิทธิภาพการทำนาย คือ ความถูกต้อง ความเที่ยงตรง ความครบถ้วน มัชฌิมฮาร์โมนิค มัชฌิมเรขาคณิต พื้นที่ใต้โค้งอาร์โอซี และพื้นที่ใต้โค้งความเที่ยงตรงและความครบถ้วน ผลการทดลองพบว่า วิธีการที่มีการประยุกต์ใช้อาร์บีแบกให้ประสิทธิภาพดีกว่าวิธีการแบบดั้งเดิมที่ไม่ได้มีการใช้แนวคิดอาร์บีแบก ยิ่งไปกว่านั้น วิธีการเรียนรู้เบย์แบบง่ายที่มีการประยุกต์ใช้อาร์บีแบกให้ประสิทธิภาพดีกว่าวิธีต้นไม้ตัดสินใจ จากงานวิจัยนี้ทำเกิดแนวคิดในการนำมาตรวัดประสิทธิภาพการทำนายดังกล่าวมาใช้ในงานวิจัย

2.2.1.3 การทำนายข้อบกพร่องในซอฟต์แวร์โดยใช้มาตรวัดรหัสคงที่ประเมินค่าแนวโน้มข้อบกพร่อง (Software Defect Prediction Using Static Code Metrics Underestimates Defect-Proneness)

งานวิจัยนี้ [3] มีจุดประสงค์เพื่อศึกษาว่ามีความสอดคล้องกับความเชื่อทางด้านวิศวกรรมซอฟต์แวร์ในปัจจุบัน กล่าวคือ ถ้าซอฟต์แวร์ยิ่งมีขนาดใหญ่ แสดงว่าโครงสร้างภายในของซอฟต์แวร์จะต้องมีความซับซ้อนมาก ซึ่งความซับซ้อนนี้เองที่มีแนวโน้มที่จะทำให้ซอฟต์แวร์เกิดข้อบกพร่องขึ้น โดยผู้วิจัยได้ประยุกต์ใช้วิธีซัพพอร์ตเวกเตอร์แมชชีนในการทำนายข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์กับชุดข้อมูลเอ็มดีพี โดยวิเคราะห์ผลลัพธ์จากการทำนายของวิธีการซัพพอร์ตเวกเตอร์แมชชีน ซึ่งผล

จากการทำนายแสดงให้เห็นว่า ถ้าโมดูลที่ทดสอบมีค่าตัดสินใจ (decision value) สูงแสดงว่าโมดูลนั้นมีโอกาสที่จะเกิดข้อบกพร่องขึ้น ซึ่งสอดคล้องกับความเชื่อทางด้านวิศวกรรมซอฟต์แวร์ดังกล่าว จากงานวิจัยนี้ทำเกิดแนวคิดในการนำมาตรวัดซอฟต์แวร์ที่วัดจากขนาดและโครงสร้างของซอฟต์แวร์ดังกล่าวมาใช้ในงานวิจัย

2.2.1.4 การใช้การเรียนรู้ความไม่สมดุลของคลาสสำหรับการทำนายข้อบกพร่องของซอฟต์แวร์ (Using Class Imbalance Learning for Software Defect Prediction)

งานวิจัยนี้ [4] มีจุดประสงค์เพื่อต้องการแก้ไขปัญหาความไม่สมดุลของคลาสในข้อมูลและเพิ่มประสิทธิภาพของการทำนายข้อบกพร่อง ผู้วิจัยจึงได้คำนึงถึงปัญหาความไม่สมดุลของคลาสในชุดข้อมูล โดยศึกษากลยุทธ์ต่างๆเพื่อหาวิธีแก้ไขปัญหาดังกล่าว ซึ่งประกอบไปด้วย เทคนิคการสุ่มตัวอย่าง การย้ายเทรสโฮลด์ และอัลกอริทึมแบบรวมกลุ่ม โดยมีมาตรวัดประสิทธิภาพการทำนาย คือ มัชฌิมเรขาคณิต พื้นที่ใต้โค้งอาร์โอซี และความสมดุล ซึ่งผลจากการทดลองพบว่า เทคนิคเอดาบูทเอ็นซี (AdaBoost.NC) ให้ประสิทธิภาพดีที่สุดและมากกว่าวิธีการเรียนรู้เบย์แบบง่ายและป่าสุ่ม นอกจากนี้ วิธีการเอดาบูทเอ็นซีแบบไดนามิกที่ผู้วิจัยได้นำเสนอยังสามารถปรับปรุงประสิทธิภาพการทำนายให้ดีขึ้นกว่าวิธีเอดาบูทเอ็นซีแบบดั้งเดิมอีกด้วย จากงานวิจัยดังกล่าวพบว่ามีวิธีการต่างๆที่สามารถแก้ไขปัญหาความไม่สมดุลของคลาสในชุดข้อมูลได้และทำให้เกิดแนวคิดในการนำเทคนิคการสุ่มตัวอย่างมาใช้ในงานวิจัย

2.2.2 กลุ่มงานวิจัยด้านการทำนายระดับความรุนแรงของข้อบกพร่องในซอฟต์แวร์

งานวิจัยกลุ่มนี้นำเสนอวิธีการในการสร้างแบบจำลองสำหรับทำนายระดับความรุนแรงของข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์ โดยประยุกต์ใช้การทำเหมืองข้อความ การคัดเลือกคุณลักษณะและวิธีการเรียนรู้ของเครื่องจักรด้วยวิธีการต่างๆ ซึ่งสามารถอธิบายรายละเอียดแต่ละงานวิจัยได้ดังนี้

2.2.2.1 การประเมินความรุนแรงอัตโนมัติของรายงานข้อบกพร่องของซอฟต์แวร์ (Automated Severity Assessment of Software Defect Reports)

งานวิจัยนี้ [6] มีจุดประสงค์เพื่อนำเสนอวิธีการประเมินความรุนแรงของรายงานข้อบกพร่องของซอฟต์แวร์แบบอัตโนมัติ โดยประยุกต์ใช้การทำเหมืองข้อความ และการเรียนรู้ของเครื่องจักร ผู้วิจัยได้เสนอวิธีการที่ชื่อว่า เซเวอริส (SEVERIS) ที่ใช้ในการสกัดและวิเคราะห์รายละเอียดของข้อความจากรายงานข้อบกพร่อง ซึ่งจะใช้เทคนิคการทำเหมืองข้อความ ได้แก่ การสกัดข้อความ (tokenization) การกำจัดคำหยุด (stop word removal) และการหารากศัพท์ (stemming) ในการสกัดคุณลักษณะที่เกี่ยวข้องกันของแต่ละรายงาน และใช้วิธีการเรียนรู้ของเครื่องจักรในการกำหนดคุณลักษณะระดับความรุนแรงที่เหมาะสม โดยได้ทำกรณีศึกษากับชุดข้อมูลจากระบบติดตามโครงการ

และปัญหาของนาซ่า (PITS) จำนวน 5 โครงการ และประเมินผลลัพธ์ด้วยค่ามัชฌิมฮาร์โมนิค จากผลการทดลองพบว่า ผลลัพธ์จะให้ค่าอยู่ในช่วง 65 - 98 % ซึ่งขึ้นอยู่กับระดับความรุนแรงที่แตกต่างกัน จากงานวิจัยจะพบว่าเป็นการทำนายระดับความรุนแรงของข้อบกพร่องโดยการวิเคราะห์จากรายงานข้อบกพร่อง ซึ่งต่างจากงานวิจัยนี้ที่วิเคราะห์จากมาตรวัดซอฟต์แวร์

2.2.2.2 การทำนายระดับความรุนแรงของบั๊กที่ถูกรายงาน (Predicting the Severity of a Reported Bug)

งานวิจัยนี้ [7] มีจุดประสงค์เพื่อนำเสนอการทำนายระดับความรุนแรงของข้อบกพร่องโดยประยุกต์ใช้วิธีการทำเหมืองข้อความในการวิเคราะห์รายละเอียดของข้อความจากรายงานข้อบกพร่องบนชุดข้อมูลจาก Bugzilla จำนวน 3 โครงการ คือ Mozilla Eclipse และ GNOME นอกจากนี้ยังมีการกำหนดขนาดที่เพียงพอให้กับข้อมูลสอน โดยกำหนดให้มี 500 รายงานต่อ 1 ระดับความรุนแรง ซึ่งจากผลการทดลองพบว่าผลลัพธ์จะให้ค่าความเที่ยงตรงและความครบถ้วน โดยโครงการ Mozilla และ Eclipse มีค่าอยู่ในช่วง 65 - 75 % ส่วนโครงการ GNOME มีค่าอยู่ในช่วง 70 - 85 % จากงานวิจัยจะพบว่าเป็นการทำนายระดับความรุนแรงของข้อบกพร่องโดยการวิเคราะห์จากรายงานข้อบกพร่องเช่นเดียวกับงานวิจัยในหัวข้อ 2.2.2.1 แต่มีการกำหนดปริมาณข้อมูลของแต่ละระดับความรุนแรงเพื่อให้เพียงพอต่อการทำนาย

2.2.2.3 การเปรียบเทียบอัลกอริทึมเหมืองข้อมูลสำหรับการทำนายระดับความรุนแรงของบั๊กที่ถูกรายงาน (Comparing Mining Algorithms for Predicting the Severity of a Reported Bug)

งานวิจัยนี้ [8] เป็นงานเพิ่มเติมจากงานวิจัยในหัวข้อ 2.2.2.2 ซึ่งมีจุดประสงค์เพื่อเปรียบเทียบผลลัพธ์จากแบบจำลองในงานวิจัยก่อนหน้าที่สร้างด้วยวิธีการเรียนรู้เบย์แบบง่าย กับแบบจำลองอื่นๆ จำนวน 4 แบบจำลอง ได้แก่ การเรียนรู้เบย์แบบมัลติโนเมียล เคเนียร์เรสเนเบอร์ และ ซัพพอร์ตเวกเตอร์แมชชีน จากผลการทดลองแสดงให้เห็นว่า แบบจำลองที่สร้างด้วยวิธีการเรียนรู้เบย์แบบมัลติโนเมียลให้ประสิทธิภาพในการทำนายสูงที่สุด ยิ่งไปกว่านั้นยังใช้เวลาในการทำนายและข้อมูลสอนน้อยกว่าวิธีการอื่นๆอีกด้วย

2.2.2.4 การกำหนดความรุนแรงของข้อบกพร่องโดยใช้เทคนิคการเรียนรู้ของเครื่องจักร (Determining Bug Severity using Machine Learning Techniques)

งานวิจัยนี้ [9] มีจุดประสงค์เพื่อนำเสนอการทำนายความรุนแรงของบั๊กโดยใช้เทคนิคการเรียนรู้ของเครื่องจักร โดยผู้วิจัยได้ทำการเปรียบเทียบเทคนิคการเรียนรู้ของเครื่องจักรจำนวน 6 ชนิด ได้แก่ การเรียนรู้เบย์แบบง่าย เคเนียร์เรสเนเบอร์ การเรียนรู้เบย์แบบมัลติโนเมียล ซัพพอร์ตเวกเตอร์

แมชชีน ต้นไม้ตัดสินใจ และริปเปอร์ (RIPPER) ซึ่งทำการทดลองกับข้อมูลชุดเดียวกันกับงานวิจัยในหัวข้อ 2.2.2.1 จากผลการทดลองพบว่าตัวจำแนกประเภทสามารถจำแนกประเภทได้ถูกต้องที่สุดที่ 125 คุณลักษณะ

2.2.2.5 การศึกษาเชิงประจักษ์ในการปรับปรุงการทำนายระดับความรุนแรงของรายงานข้อบกพร่องโดยใช้การคัดเลือกคุณลักษณะ (An Empirical Study on Improving Severity Prediction of Defect Reports using Feature Selection)

งานวิจัยนี้ [10] มีจุดประสงค์เพื่อวิเคราะห์ว่าการคัดเลือกคุณลักษณะสามารถช่วยให้การทำนายระดับความรุนแรงของข้อบกพร่องดีขึ้นได้หรือไม่ โดยได้ประยุกต์ใช้วิธีการคัดเลือกคุณลักษณะจำนวน 3 วิธี คือ Information Gain, Chi-Square และ Correlation Coefficient ร่วมกับแบบจำลองที่สร้างด้วยวิธีการเรียนรู้เบย์แบบง่าย กับชุดข้อมูลจำนวน 4 โครงการจาก Eclipse และ Mozilla จากผลการทดลองแสดงให้เห็นว่าการคัดเลือกคุณลักษณะทั้งสามวิธีสามารถปรับปรุงประสิทธิภาพในการทำนายได้เกินครึ่งหนึ่งของโครงการทั้งหมด

2.2.2.6 การปรับปรุงการจำแนกประเภทระดับความรุนแรงของบั๊ก (Towards an Improvement of Bug Severity Classification)

งานวิจัยนี้ [11] มีจุดประสงค์เพื่อวิเคราะห์ข้อบกพร่องว่ารุนแรงหรือไม่รุนแรง โดยประยุกต์ใช้วิธี bi-grams ร่วมกับการคัดเลือกคุณลักษณะในการวิเคราะห์รายละเอียดข้อความจากรายงานข้อบกพร่อง และสร้างแบบจำลองด้วยวิธีการเรียนรู้เบย์แบบง่าย โดยใช้ชุดข้อมูลจาก Eclipse และ Mozilla จากผลการทดลองแสดงให้เห็นว่าการประยุกต์ใช้ bi-grams สามารถปรับปรุงประสิทธิภาพการทำนายได้เพียงเล็กน้อยเท่านั้น แต่การคัดเลือกคุณลักษณะทำให้ประสิทธิภาพการทำนายดีขึ้น เนื่องจากสามารถคัดเลือก informative term ได้ถูกต้องมากยิ่งขึ้น ซึ่งผลลัพธ์ที่ได้จะแตกต่างกันไปขึ้นอยู่กับชุดข้อมูล

2.2.3 เปรียบเทียบความแตกต่างของงานวิจัยที่เกี่ยวข้องกับวิธีการที่นำเสนอ

จากการศึกษางานวิจัยที่เกี่ยวข้องทั้งหมดที่กล่าวถึงข้างต้น ทุกงานวิจัยมีจุดประสงค์เพื่อนำเสนอแบบจำลองในการทำนายข้อบกพร่องและทำนายระดับความรุนแรงของข้อบกพร่อง แต่ก็ยังมีความแตกต่างกัน คือ วิธีการสร้างแบบจำลอง มาตรวัดและการประเมินผลแบบจำลอง จึงได้เปรียบเทียบความแตกต่างระหว่างงานวิจัยที่เกี่ยวข้องทั้ง 2 กลุ่มกับงานวิจัยที่นำเสนอ ดังแสดงในตารางที่ 11

ตารางที่ 11 เปรียบเทียบความแตกต่างระหว่างงานวิจัยที่เกี่ยวข้องกับงานวิจัยที่นำเสนอ

งานวิจัย	วัตถุประสงค์	เทคนิคที่ใช้	มาตรวัดที่ใช้	มาตรวัดที่ใช้ในการประเมินผล
กลุ่มงานวิจัยด้านการทำนายข้อบกพร่องของซอฟต์แวร์				
2.2.1.1 [1]	- ประเมินความสามารถของวิธีการ SVM ในการ ทำนายข้อบกพร่อง โดยเปรียบเทียบกับวิธีการจำแนกประเภทอื่น	- Decision Tree - Random Forest - Naïve Bayes - Bayesian Belief Network - k-Nearest Neighbors - Logistic Regression - Neural Network - Multilayer Perceptron	- McCabe - Halstead - Line Count	- Accuracy - Precision - Recall - F-measure
2.2.1.2 [2]	- นำเสนอการทำนายข้อบกพร่องในระบบซอฟต์แวร์ที่มีการประกันภัยสูง โดยเปรียบเทียบวิธีการจำแนกประเภทอื่น จำนวน 3 วิธี	- RBBag - Naïve Bayes - Decision Tree	- McCabe - Halstead - Line Count	- Accuracy - Precision - Recall - F-measure - G-mean - AUC - AUPR
2.2.1.3 [3]	- นำเสนอการทำนายข้อบกพร่อง เพื่อแสดงให้เห็นว่าสอดคล้องกับความเชื่อทางวิศวกรรมซอฟต์แวร์ในปัจจุบัน	- Support Vector Machines	- McCabe - Halstead - Line Count	- Decision Value
2.2.1.4 [4]	- นำเสนอการทำนายข้อบกพร่อง โดยทำการเปรียบเทียบเทคนิค Resampling เทคนิค Threshold Moving และเทคนิค Ensemble Algorithm	- Naïve Bayes - Random Forest - Resampling - Threshold Moving - SMOTEBoost - AdaBoost.NC	- McCabe - Halstead - Line Count	- Precision - Recall - Balance - F-measure - AUC - G-mean

ตารางที่ 11 เปรียบเทียบความแตกต่างระหว่างงานวิจัยที่เกี่ยวข้องกับงานวิจัยที่นำเสนอ (ต่อ)

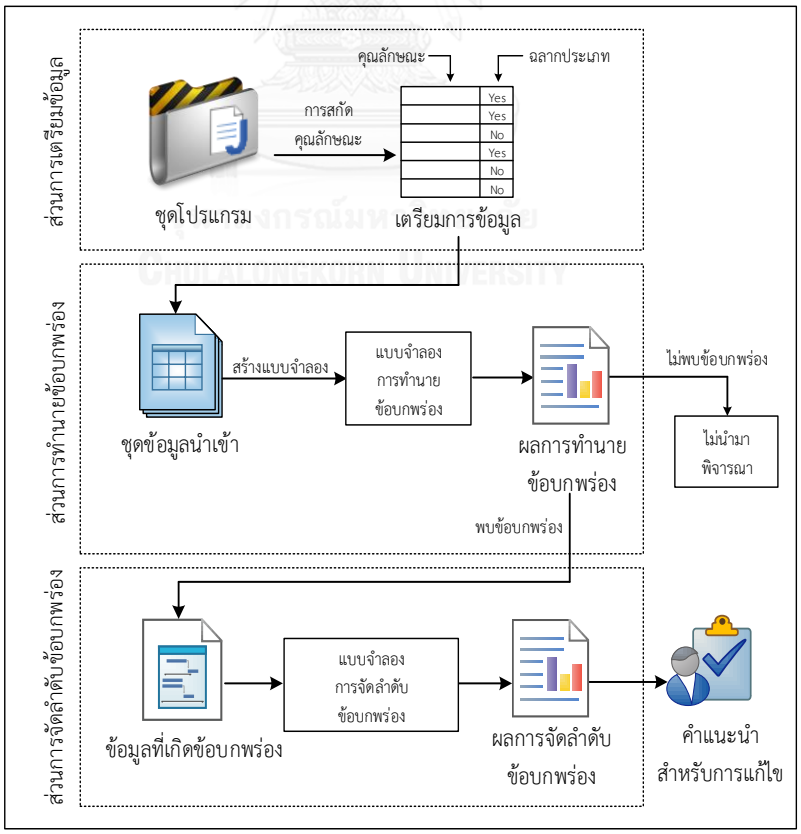
งานวิจัย	วัตถุประสงค์	เทคนิคที่ใช้	มาตรวัดที่ใช้	มาตรวัดที่ใช้ในการประเมินผล
กลุ่มงานวิจัยด้านการทำนายระดับความรุนแรงของข้อบกพร่องในซอฟต์แวร์				
2.2.2.1 [6]	- นำเสนอวิธีการประเมินความรุนแรงของรายงานข้อบกพร่องของซอฟต์แวร์ โดยประยุกต์ใช้วิธี Text Mining และวิธีการจำแนกประเภท - นำเสนอเครื่องมือที่สนับสนุนแนวคิดที่ได้นำเสนอ	- Tokenization - Stop Word Removal - Stemming - RIPPER	- Defect Description	- Precision - Recall - F-measure
2.2.2.2 [7]	- นำเสนอการทำนายระดับความรุนแรงของข้อบกพร่องโดยประยุกต์ใช้วิธีการทำเหมืองข้อความในการวิเคราะห์รายละเอียดของข้อความจากรายงานข้อบกพร่อง	- Naïve Bayes - Decision Tree - Support Vector Machines -Text Mining	- Defect Description	- Precision - Recall
2.2.2.3 [8]	- เปรียบเทียบผลลัพธ์จากแบบจำลองในงานวิจัยก่อนหน้า (งานวิจัย 2.2.2.2) กับแบบจำลองอื่นๆ จำนวน 4 แบบจำลอง	- Naïve Bayes - Naïve Bayes Multinomial - k-Nearest Neighbor - Support Vector Machines -Text Mining	- Defect Description	- ROC - AUC
2.2.2.4 [9]	- นำเสนอวิธีการทำนายความรุนแรงของข้อบกพร่อง โดยใช้วิธีการจำแนกประเภท	- Naïve Bayes - k-Nearest Neighbor - Naïve Bayes Multinomial - Support Vector Machines - Decision Tree - RIPPER	- Defect Description	- Accuracy - Precision - Recall - F-measure

ตารางที่ 11 เปรียบเทียบความแตกต่างระหว่างงานวิจัยที่เกี่ยวข้องกับงานวิจัยที่นำเสนอ (ต่อ)

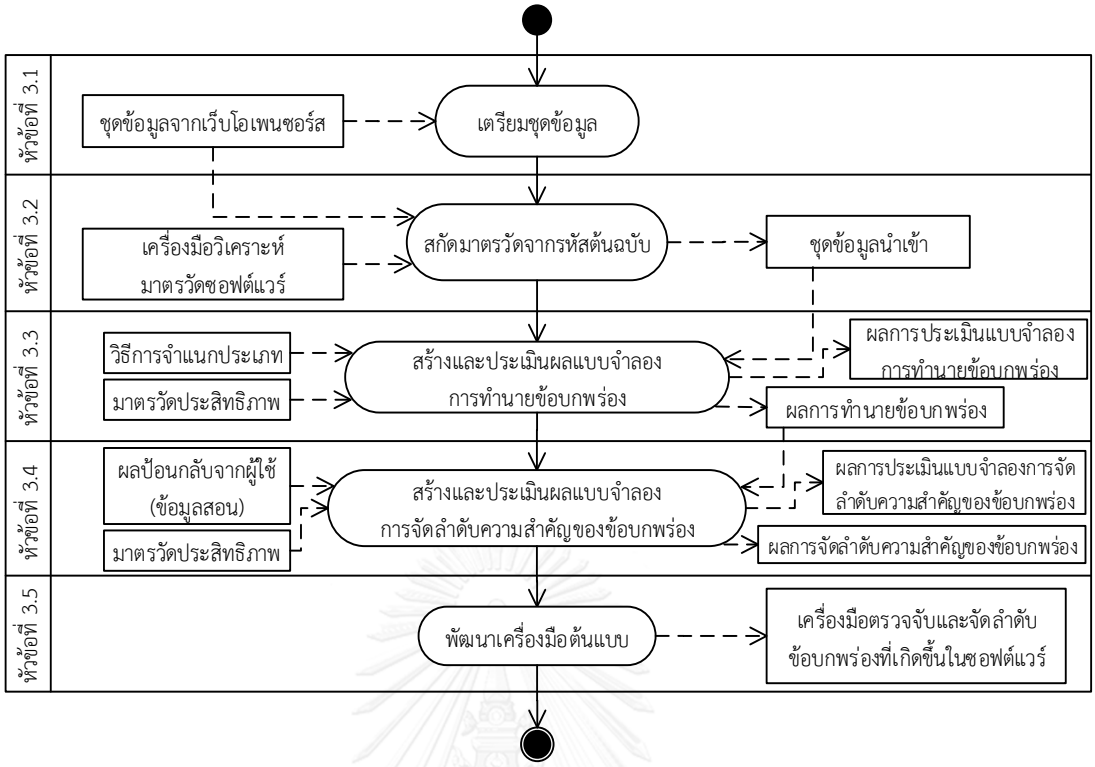
งานวิจัย	วัตถุประสงค์	เทคนิคที่ใช้	มาตรวัดที่ใช้	มาตรวัดที่ใช้ในการประเมินผล
2.2.2.5 [10]	- วิเคราะห์ว่าการคัดเลือกคุณลักษณะสามารถช่วยให้การทำนายระดับความรุนแรงของข้อบกพร่องดีขึ้นได้หรือไม่ โดยได้ประยุกต์ใช้วิธีการคัดเลือกคุณลักษณะ จำนวน 3 วิธี ร่วมกับแบบจำลองที่สร้างด้วยวิธี NB	- Information Gain - Chi-Square - Correlation Coefficient - Naïve Bayes - Text Mining	- Defect Description	- ROC - AUC
2.2.2.6 [11]	- วิเคราะห์ข้อบกพร่องว่ารุนแรงหรือไม่รุนแรง โดยประยุกต์ใช้วิธี bi-grams ร่วมกับการคัดเลือกคุณลักษณะ และสร้างแบบจำลองด้วยวิธี NB	- Text Mining - Bi-grams - Naïve Bayes	- Defect Description	- Accuracy - Precision - Recall - ROC - AUC
งานวิจัยที่นำเสนอ	- นำเสนอกรอบงานสำหรับการตรวจจับข้อบกพร่องของซอฟต์แวร์และการจัดลำดับของข้อบกพร่องตามระดับความรุนแรงในชุดข้อมูลไม่สมดุล - เปรียบเทียบประสิทธิภาพของแบบจำลองการตรวจจับข้อบกพร่องของซอฟต์แวร์ - เปรียบเทียบประสิทธิภาพของแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องในซอฟต์แวร์ตามระดับความรุนแรงของวิธีการจำแนกประเภทแบบกึ่งมีผู้สอนและเปรียบเทียบประสิทธิภาพของวิธีการ AHP - นำเสนอเครื่องมือต้นแบบที่สนับสนุนกรอบงานที่ได้นำเสนอ	- Decision Tree - Random Forest - Naïve Bayes - k-Nearest Neighbor - Support Vector Machines - Unbiased Support Vector Machines - Analytic Hierarchy Process - YATSI - OS-YATSI	- Chidamber and Kemerer - Martin - QMOOD - Tang - McCabe	- Precision - Probability of Detection - Probability of False Alarm - True Negative Rate - F-measure - G-mean - Average Precision -Average Dependency - Average Performance Rate

บทที่ 3
การสร้างแบบจำลองสำหรับทำนายและจัดลำดับความสำคัญของข้อบกพร่อง

การตรวจจับข้อบกพร่องของซอฟต์แวร์ คือ ความสามารถในการตรวจจับและวิเคราะห์โอกาสที่จะเกิดข้อบกพร่องขึ้นภายในซอฟต์แวร์ ซึ่งถือเป็นกระบวนการที่สำคัญในงานทางด้านวิศวกรรมซอฟต์แวร์ ที่ช่วยให้นักพัฒนาทราบถึงสาเหตุของปัญหาที่เกิดขึ้นว่ามาจากส่วนใดของซอฟต์แวร์ นอกจากนี้การจัดลำดับของข้อบกพร่องที่เกิดขึ้นตามระดับความรุนแรง ยังช่วยให้นักพัฒนาสามารถตัดสินใจแก้ไขปัญหาได้อย่างถูกต้องและรวดเร็ว ส่งผลให้ซอฟต์แวร์มีคุณภาพและมีความน่าเชื่อถือมากยิ่งขึ้น งานวิจัยนี้จึงได้นำเสนอกรอบงานสำหรับการทำนายข้อบกพร่องของซอฟต์แวร์และจัดลำดับข้อบกพร่องของซอฟต์แวร์ตามระดับความรุนแรง โดยภาพรวมแนวคิดของงานวิจัยแสดงได้ดังรูปที่ 7 ซึ่งแสดงให้เห็นถึงกรอบงานทั้งหมด ตั้งแต่ส่วนการเตรียมข้อมูล การสร้างแบบจำลองเพื่อใช้ในการตรวจจับข้อบกพร่อง ตลอดจนการจัดลำดับของข้อบกพร่องที่พบตามระดับผลกระทบที่มีต่อซอฟต์แวร์ เพื่อเป็นคำแนะนำหรือแนวทางสำหรับนักพัฒนาในการแก้ไขข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์



รูปที่ 7 ภาพรวมแนวคิดของงานวิจัย



รูปที่ 8 ภาพรวมของวิธีการดำเนินงาน

จากรูปที่ 8 แผนภาพกิจกรรมแสดงภาพรวมของวิธีการดำเนินงานซึ่งประกอบด้วย 5 ขั้นตอน โดยสามารถอธิบายการทำงานในแต่ละขั้นตอนได้ดังต่อไปนี้

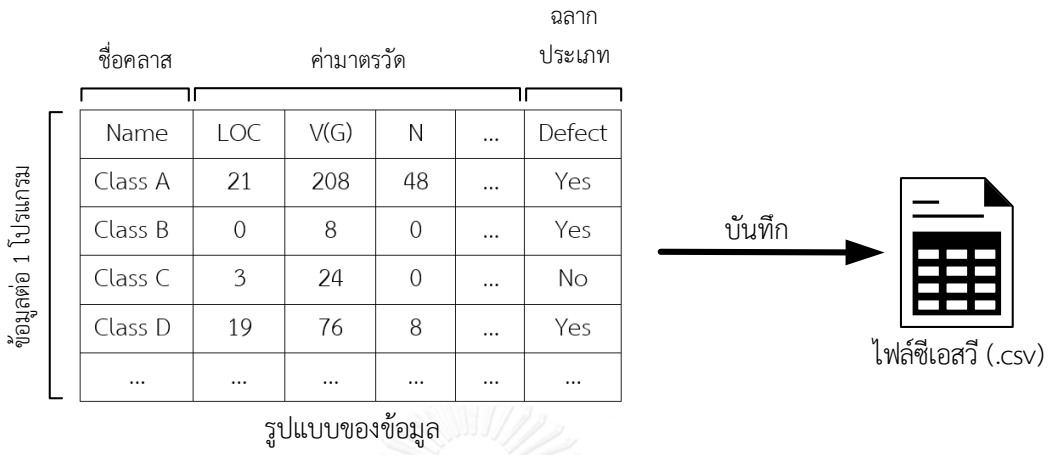
3.1 การเตรียมชุดข้อมูลจากโอเพนซอร์ส

ในงานวิจัยนี้จะทำการรวบรวมชุดโปรแกรมจากเว็บโอเพนซอร์สซอฟต์แวร์ [32] ซึ่งประกอบด้วยโปรแกรมประยุกต์สำหรับใช้งานเฉพาะทางในด้านต่างๆ โดยเลือกเฉพาะชุดโปรแกรมที่มีการพัฒนาด้วยภาษาจาวา ซึ่งผลลัพธ์ของขั้นตอนนี้จะได้รหัสต้นฉบับของชุดโปรแกรมสำหรับใช้ในขั้นตอนการการสกัดคุณลักษณะต่อไป

3.2 การสกัดคุณลักษณะจากรหัสต้นฉบับ

ในงานวิจัยนี้จะทำการสกัดค่าคุณลักษณะโดยนำรหัสต้นฉบับ (source code) ของชุดโปรแกรมที่ได้จากขั้นตอน 3.1 มาสกัดค่าคุณลักษณะด้วยโปรแกรม CKJM [21] โดยมาตรวัดที่ใช้ในงานวิจัยนี้ ประกอบด้วย 17 มาตรวัด ที่ระบุไว้ในบทที่ 2 หัวข้อ 2.1.1 ตารางที่ 1 ค่าที่ได้ในขั้นตอนนี้จะต้องจัดการกับข้อมูลโดยมีรูปแบบไฟล์ซีเอสวี (.csv) และแยกเป็นแต่ละรายโปรแกรม เพื่อใช้เป็นข้อมูลนำเข้าในขั้นตอนถัดไป ซึ่งประกอบด้วย ชื่อคลาส ค่ามาตรวัดที่ได้จากรหัสต้นฉบับ และฉลากประเภท ดังแสดงในรูปที่ 9 โดยฉลากประเภทของชุดโปรแกรมทั้งหมดในขั้นตอนนี้จะนำมาจาก

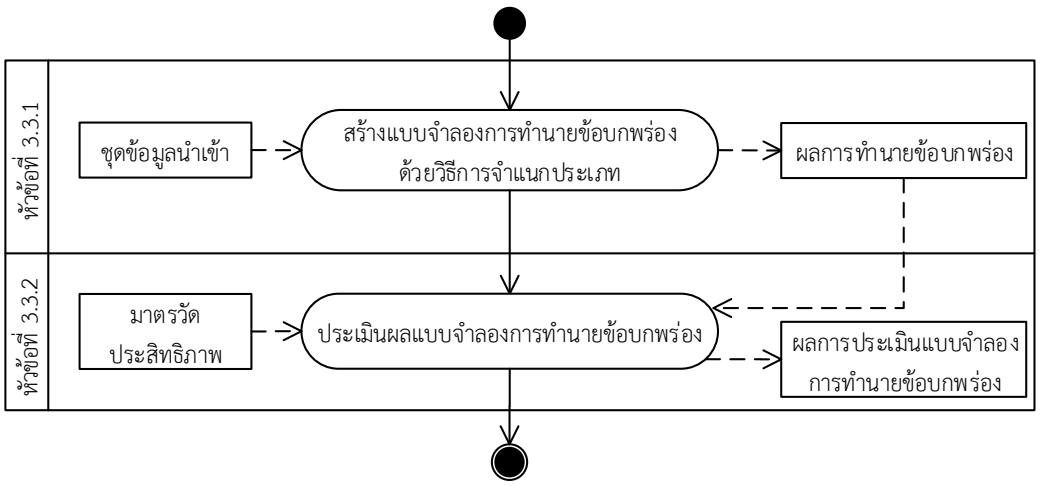
แหล่งข้อมูล PROMISE [33] ซึ่งประกอบด้วย 2 ฉลากประเภท คือ พบข้อบกพร่อง (Yes) และไม่พบข้อบกพร่อง (No)



รูปที่ 9 รูปแบบของข้อมูลสำหรับการทำนายข้อบกพร่อง

3.3 ขั้นตอนการสร้างและประเมินผลแบบจำลองการทำนายข้อบกพร่อง

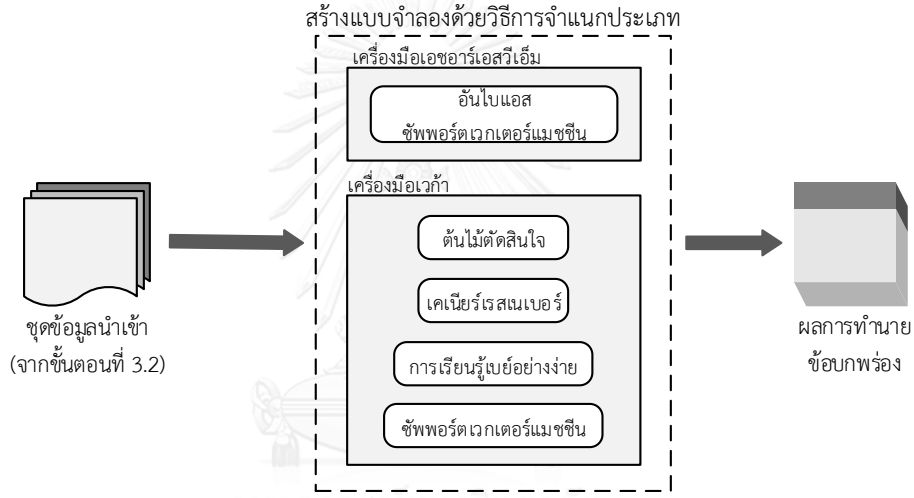
ขั้นตอนนี้มีวัตถุประสงค์เพื่อสร้างแบบจำลองการทำนายข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์ โดยสร้างแบบจำลองรายโปรแกรมด้วยวิธีการจำแนกประเภทที่มีรูปแบบเป็นการเรียนรู้แบบมีผู้สอน จากนั้นจึงทำการประเมินผลแบบจำลองที่สร้างขึ้นด้วยมาตรวัดประสิทธิภาพตามที่ได้กำหนดไว้ ขั้นตอนนี้ประกอบด้วย 2 ขั้นตอนย่อย คือ 1) การสร้างแบบจำลองการทำนายข้อบกพร่อง และ 2) การประเมินผลแบบจำลองการทำนายข้อบกพร่อง ซึ่งสามารถแสดงรายละเอียดได้ด้วยแผนภาพกิจกรรมดังรูปที่ 10



รูปที่ 10 ภาพรวมของการสร้างและประเมินผลแบบจำลองการทำนายข้อบกพร่อง

3.3.1 การสร้างแบบจำลองการทำนายข้อบกพร่อง

ในงานวิจัยนี้ได้ประยุกต์ใช้วิธีการจำแนกประเภทที่ชื่อว่า “อันไบแอสซัพพอร์ตเวกเตอร์แมชชีน หรือ อาร์เอสวีเอ็ม” ดังที่ได้อธิบายรายละเอียดในบทที่ 2 หัวข้อ 2.1.2 เพื่อสร้างแบบจำลองการทำนายข้อบกพร่องของซอฟต์แวร์ด้วยเครื่องมือ HR-SVM [34] นอกจากนี้ผู้วิจัยได้สร้างแบบจำลองด้วยวิธีการจำแนกประเภทแบบดั้งเดิมอีกจำนวน 4 แบบจำลองด้วยเครื่องมือ Weka [35] ซึ่งประกอบด้วย 1) ต้นไม้ตัดสินใจ 2) การเรียนรู้ด้วยเพื่อนบ้านใกล้ที่สุด 3) การเรียนรู้เบย์แบบง่าย และ 4) ซัพพอร์ตเวกเตอร์แมชชีน เพื่อใช้ในการเปรียบเทียบกับวิธีอาร์เอสวีเอ็ม ซึ่งในงานวิจัยนี้ได้ใช้การเลือกสุ่มข้อมูลแบบความเที่ยงตรง 10 กลุ่ม (10-fold cross validation) โดยขั้นตอนการสร้างแบบจำลองสามารถแสดงดังในรูปที่ 11



รูปที่ 11 กระบวนการสร้างแบบจำลองการทำนายข้อบกพร่อง

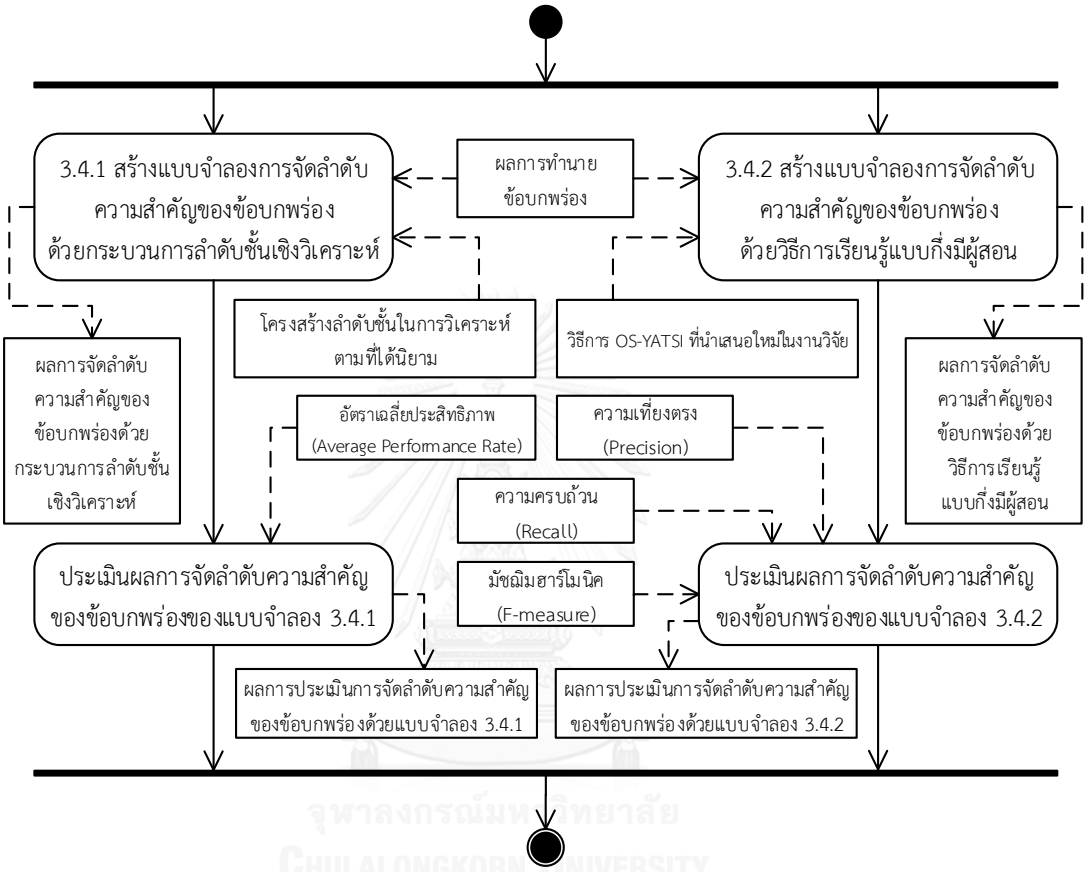
3.3.2 การประเมินผลแบบจำลองการทำนายข้อบกพร่อง

ในขั้นตอนนี้จะใช้สำหรับการประเมินผลแบบจำลอง โดยใช้ชุดข้อมูลที่ได้จากขั้นตอนที่ 3.2 ทำนายข้อบกพร่องตามแบบจำลองที่ได้จากขั้นตอนที่ 3.3.1 จากนั้นทำการประเมินผลโดยใช้มาตรวัดประสิทธิภาพการทำนาย (Prediction Performance Metrics) จำนวน 6 มาตรวัด ดังที่ได้อธิบายรายละเอียดในบทที่ 2 หัวข้อ 2.1.7 ตารางที่ 9

3.4 ขั้นตอนการสร้างและประเมินผลแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่อง

ในขั้นตอนนี้จะนำผลลัพธ์ที่ได้จากขั้นตอนที่ 3.3.1 มาใช้ในการวิเคราะห์เพื่อทำการสร้างแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์ โดยจะสร้างทั้งหมด 2 แบบจำลอง จากนั้นจึงทำการประเมินผลแบบจำลองที่สร้างขึ้นด้วยมาตรวัดประสิทธิภาพตามที่ได้

กำหนดไว้ ซึ่งในขั้นตอนนี้ประกอบด้วย 2 ขั้นตอนย่อย คือ 1) การสร้างและประเมินผลแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องด้วยกระบวนการลำดับชั้นเชิงวิเคราะห์ และ 2) การสร้างและประเมินผลแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน ซึ่งสามารถแสดงรายละเอียดได้ด้วยแผนภาพกิจกรรมดังรูปที่ 12



รูปที่ 12 ภาพรวมของการสร้างและประเมินผลแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่อง

3.4.1 การสร้างและประเมินผลแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องด้วยกระบวนการลำดับชั้นเชิงวิเคราะห์

ขั้นตอนนี้ผู้วิจัยได้ประยุกต์ใช้กระบวนการลำดับชั้นเชิงวิเคราะห์ในการสร้างแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่อง เนื่องจากเป็นวิธีการที่นิยมใช้งานอย่างกว้างขวางที่ช่วยในการตัดสินใจเพื่อให้ได้คำตอบที่เหมาะสมที่สุด ซึ่งในขั้นตอนนี้ประกอบด้วยขั้นตอนย่อย 2 ขั้นตอน คือ 1) การสร้างแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องด้วยกระบวนการลำดับชั้นเชิงวิเคราะห์ และ 2) การประเมินผลแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องด้วยกระบวนการลำดับชั้นเชิงวิเคราะห์ โดยสามารถอธิบายรายละเอียดของขั้นตอนย่อยต่างๆ ได้ดังนี้

1) การสร้างแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องด้วยกระบวนการลำดับชั้นเชิงวิเคราะห์

โดยขั้นตอนการสร้างแบบจำลองด้วยกระบวนการลำดับชั้นเชิงวิเคราะห์ประกอบด้วย 3 ขั้นตอนหลัก (ตามที่ได้อธิบายรายละเอียดในบทที่ 2 หัวข้อ 2.1.4) คือ

(1) กำหนดโครงสร้างลำดับชั้นเชิงวิเคราะห์

ในการวิเคราะห์จะต้องมีการกำหนดโครงสร้างเพื่อแบ่งการวิเคราะห์ออกเป็นลำดับชั้น โดยในแต่ละระดับอาจมีหลายเกณฑ์ตัดสินใจ ซึ่งในงานวิจัยนี้จะพิจารณา 2 เกณฑ์ตัดสินใจ คือ

- เกณฑ์ตัดสินใจที่ 1: ค่าความมั่นใจของการเกิดข้อบกพร่อง (Confidence Value) ซึ่งเป็นผลลัพธ์ที่ได้มาจากการทำนายข้อบกพร่อง (ขั้นตอนที่ 3.3) โดยจะมีค่าอยู่ระหว่าง 0-1

- เกณฑ์ตัดสินใจที่ 2: ค่าผลกระทบการขึ้นต่อกันของคลาส (Class Dependency Impact: CDI) ผู้วิจัยได้นำเสนอมาตรวัดนี้โดยประยุกต์ใช้แนวคิดจากมาตรวัดความไม่เสถียร (Instability: I) [36] ซึ่งสามารถอธิบายรายละเอียดมาตรวัดค่าผลกระทบการขึ้นต่อกันของคลาสได้ดังนี้

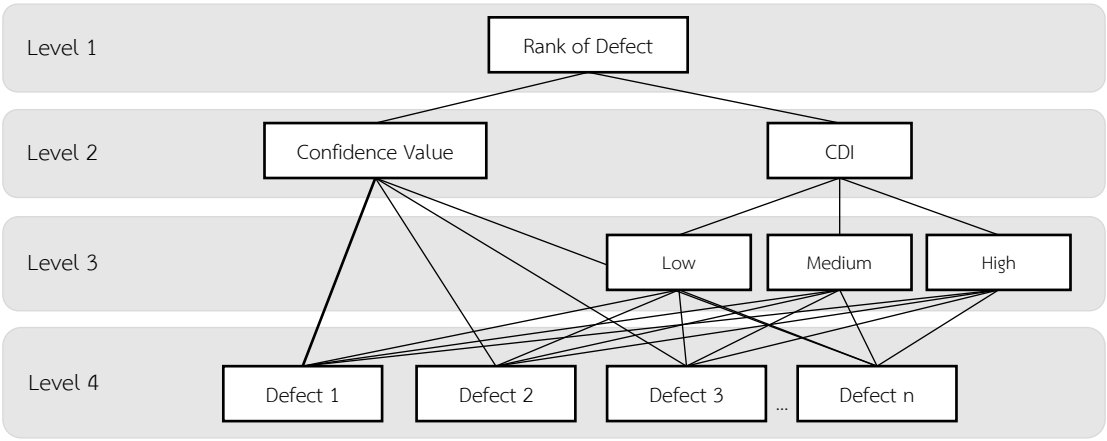
<u>นิยามตามบริบทของงานวิจัย:</u> ผลกระทบการขึ้นต่อกันของคลาส (Class Dependency Impact: CDI) คือ มาตรวัดที่บ่งชี้ถึงผลกระทบของคลาสใดๆที่กำลังพิจารณาที่มีผลต่อคลาสอื่นๆ ภายในโปรแกรม สามารถคำนวณได้ดังสมการ (11)

$$Class\ \ Dependency\ \ Impact\,(CDI) = \frac{Ca}{Ca + Ce} \qquad (11)$$

โดยที่ $Afferent\ Coupling\ (Ca)$ คือ จำนวนคลาสใดๆที่มีการเรียกใช้งานคลาสที่เรากำลังพิจารณา

$Efferent\ Coupling\ (Ce)$ คือ จำนวนคลาสใดๆที่ถูกเรียกใช้งานโดยคลาสที่เรากำลังพิจารณา

โดยค่า CDI จะมีค่าอยู่ระหว่าง 0-1 หากมีค่า CDI สูง หมายความว่า คลาสนั้นๆ มีความเกี่ยวข้องกันสูงมาก ดังนั้น หากมีการเปลี่ยนแปลงหรือเกิดข้อผิดพลาดใดๆขึ้นกับคลาสนั้นๆ จะส่งผลกระทบกับโปรแกรมมาก ซึ่งในเกณฑ์การตัดสินใจที่ 2 นี้ จะประกอบด้วย 3 เกณฑ์การตัดสินใจย่อย คือ 1) ผลกระทบต่ำ (Low) 2) ผลกระทบปานกลาง (Medium) และ 3) ผลกระทบสูง (High) ดังแสดงในรูปที่ 13



รูปที่ 13 แผนภูมิโครงสร้างลำดับชั้นเชิงวิเคราะห์ตามบริบทของงานวิจัย

ซึ่งแต่ละเกณฑ์การตัดสินใจย่อยจะถูกกำหนดเงื่อนไขไว้ดังนี้

- จะเป็นเกณฑ์การตัดสินใจย่อยระดับต่ำ เมื่อ CDI มีค่าอยู่ในช่วง 0.00 - 0.33
- จะเป็นเกณฑ์การตัดสินใจย่อยระดับปานกลาง เมื่อ CDI มีค่าอยู่ในช่วง 0.34 – 0.66
- จะเป็นเกณฑ์การตัดสินใจย่อยระดับสูง เมื่อ CDI มีค่าอยู่ในช่วง 0.67 – 1.00

(2) คำนวณหาลำดับความสำคัญ

การคำนวณหาค่าลำดับความสำคัญของเกณฑ์การตัดสินใจแต่ละเกณฑ์ก่อนที่จะประเมินความสำคัญของทางเลือก ในที่นี้จะการคำนวณหาค่าลำดับความสำคัญของเกณฑ์การตัดสินใจย่อยก่อน ซึ่งประกอบด้วยขั้นตอนย่อยดังต่อไปนี้

- <u>การให้ค่าน้ำหนักความสำคัญของเกณฑ์การตัดสินใจย่อย</u> สามารถทำได้โดยการสร้างตารางเมตริกซ์เปรียบเทียบเกณฑ์การตัดสินใจแบบทีละคู่ ในงานวิจัยนี้จะกำหนดมาตราส่วนในการเปรียบเทียบความสำคัญ คือ หากเกณฑ์การตัดสินใจทั้งแนวสดมภ์และคอลัมน์มีความสำคัญเท่ากันจะมีค่าระดับเป็น 1 หากเกณฑ์การตัดสินใจแนวสดมภ์มีความสำคัญกว่าเกณฑ์การตัดสินใจแนวคอลัมน์ในระดับปานกลางจะมีค่าระดับเป็น 3 และหากเกณฑ์การตัดสินใจแนวสดมภ์มีความสำคัญกว่าเกณฑ์การตัดสินใจแนวคอลัมน์ในระดับมากกว่าจะมีค่าระดับเป็น 7 ดังแสดงในตารางที่ 12 (ตามที่ได้อธิบายตัวอย่างในบทที่ 2 หัวข้อ 2.1.4 ตารางที่ 4)

- <u>คำนวณหาค่าน้ำหนักความสำคัญของเกณฑ์การตัดสินใจย่อย</u> โดยปรับผลรวมของแต่ละคอลัมน์ให้มีค่าเท่ากับ 1 จากนั้นคำนวณหาค่าน้ำหนัก (Eigenvector) ของเกณฑ์การตัดสินใจย่อยดังแสดงในตารางที่ 13

ตารางที่ 12 เมตริกซ์เปรียบเทียบเกณฑ์การตัดสินใจทีละคู่ตามบริบทของงานวิจัย

Defect		Criteria		
		Low	Medium	High
Criteria	Low	1	1/3	1/7
	Medium	3	1	1/3
	High	7	3	1

ตารางที่ 13 การปรับผลรวมของแต่ละคอลัมน์และค่าน้ำหนักของเกณฑ์การตัดสินใจย่อย

Defect		Criteria			Total	Eigenvector
		Low	Medium	High		
Criteria	Low	0.09	0.08	0.10	0.26	0.09
	Medium	0.27	0.23	0.23	0.73	0.24
	High	0.64	0.69	0.68	2.01	0.67
Total		1	1	1	3	1

- ตรวจสอบความสอดคล้องของข้อมูล โดยนำค่าน้ำหนักที่คำนวณได้คูณกับค่าในตารางที่ 12 ในทุกๆแถวและทุกๆคอลัมน์ แล้วหาค่าผลรวมในแต่ละแถวของค่าลำดับความสำคัญ ดังแสดงในตารางที่ 14 จากนั้นนำผลรวมในแต่ละแถวหารด้วยค่าน้ำหนัก (Eigenvector) เพื่อนำไปใช้ในการคำนวณหาค่าความสอดคล้องของข้อมูล (CR) ดังแสดงในตารางที่ 15

ตารางที่ 14 ผลคูณของลำดับความสำคัญ

Defect		Criteria			Total
		Low	Medium	High	
Criteria	Low	0.09	0.08	0.10	0.26
	Medium	0.26	0.24	0.22	0.73
	High	0.62	0.73	0.67	2.02

ตารางที่ 15 ผลหารระหว่างผลรวมในแต่ละแถวกับค่าน้ำหนัก

CDI	Criteria		
	Low	Medium	High
Total	0.26	0.73	2.02
Eigenvector	0.09	0.24	0.67
ผลหาร	3.00	3.01	3.01

จากตารางที่ 15 สามารถคำนวณหาค่า CR ได้ตามสมการ (9) ในบทที่ 2 หัวข้อ 2.1.4 ดังนั้น จะได้ค่า CR = 0.006 ซึ่งมีค่าน้อยกว่า 0.1 นั่นแสดงว่าค่าเกณฑ์การตัดสินใจย่อยมีความสอดคล้องกัน สามารถนำค่าน้ำหนักไปใช้ได้ ส่วนการคำนวณหาค่าน้ำหนักสำหรับเกณฑ์การตัดสินใจ เนื่องจากในงานวิจัยนี้ได้กำหนดเกณฑ์การตัดสินใจเพียง 2 เกณฑ์เท่านั้น คือ ค่าความมั่นใจของการเกิดข้อบกพร่อง (Confidence Value) และค่าผลกระทบการขึ้นต่อกันของคลาส (Class Dependency Impact, CDI) ดังนั้น จึงให้ค่าน้ำหนักของทั้งสองเกณฑ์มีค่าเท่ากัน คือ 0.5

(3) คำนวณค่าน้ำหนักความสำคัญโดยรวมได้ดังสมการ (12) โดยที่ $Defect_i$ คือ ข้อบกพร่องของคลาสที่ i

$$Defect_i = (Confidence\ Value_i \times Eigenvector\ of\ Confidence\ Value) + (Eigenvector_i\ of\ CDI \times Eigenvector\ of\ CDI) \quad (12)$$

<u>ตัวอย่างการคำนวณค่าน้ำหนักความสำคัญโดยรวม</u>

โปรแกรม Eclipse JDT Core:

Class ID 1 มีค่า CDI = 0.46 ดังนั้น เกณฑ์การตัดสินใจย่อยเป็น Medium

Class ID 4 มีค่า CDI = 0.25 ดังนั้น เกณฑ์การตัดสินใจย่อยเป็น Low

Class ID 6 มีค่า CDI = 0.93 ดังนั้น เกณฑ์การตัดสินใจย่อยเป็น High

สามารถคำนวณค่าน้ำหนักความสำคัญโดยรวมของแต่ละข้อบกพร่องได้ดังนี้

Class ID 1 = (0.712 x 0.50) + (0.24 x 0.50) = 0.476

Class ID 4 = (0.85 x 0.50) + (0.09 x 0.50) = 0.470

Class ID 6 = (1.00 x 0.50) + (0.67 x 0.50) = 0.835

ดังนั้น ผลการจัดลำดับความสำคัญของคลาสที่เกิดข้อบกพร่องทั้ง 3 คลาสของโปรแกรม Eclipse JDT Core สามารถแสดงได้ดังตารางที่ 16

ตารางที่ 16 ตัวอย่างผลการจัดลำดับข้อบกพร่องของโปรแกรม Eclipse JDT Core

ลำดับ	ชื่อคลาส	ค่า AHP
1	Class ID 6	0.835
2	Class ID 1	0.476
3	Class ID 4	0.470

2) การประเมินผลแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องด้วยกระบวนการลำดับชั้นเชิงวิเคราะห์

เนื่องจากการจัดลำดับข้อบกพร่องในงานวิจัยนี้ มีจุดประสงค์เพื่อช่วยแนะนำให้นักพัฒนาสามารถตัดสินใจแก้ไขข้อบกพร่องได้อย่างรวดเร็วและถูกต้อง ซึ่งเปรียบเสมือนระบบแนะนำ (Recommender System) ดังนั้น งานวิจัยนี้จึงประยุกต์ใช้มาตรวัดทางด้านระบบแนะนำ [29] คือ ค่าเฉลี่ยความเที่ยงตรง และนำเสนอมาตรวัดใหม่ในการประเมินผลแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่อง คือ ค่าเฉลี่ยการขึ้นต่อกันของคลาส และอัตราเฉลี่ยประสิทธิภาพ สามารถอธิบายรายละเอียดของแต่ละมาตรวัดได้ดังนี้

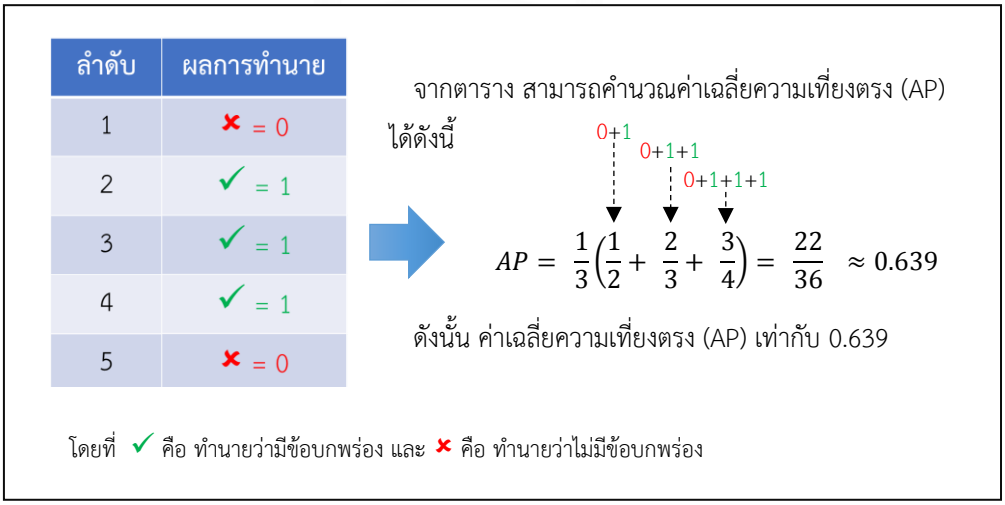
(1) ค่าเฉลี่ยความเที่ยงตรง (Average Precision: AP) เป็นมาตรวัดอันดับความเที่ยงตรง (Ranked Precision Metric) ที่ให้ความสำคัญในอันดับสูงของการทำนายข้อบกพร่องที่ถูกต้อง สามารถคำนวณหาค่าได้ดังสมการ (13) ทั้งนี้ได้ยกตัวอย่างของการคำนวณค่าเฉลี่ยความเที่ยงตรง แสดงดังรูปที่ 14

$$Average\ Precision\ (AP) = \frac{\sum_{j=1}^{n} P(j)}{D_{true}} \qquad (13)$$

โดยที่ $P(j)$ คือ ค่าความเที่ยงตรงของการทำนายว่าเป็นข้อบกพร่อง ณ ลำดับที่ j

n คือ ลำดับของข้อบกพร่องทั้งหมด

D_{true} คือ จำนวนข้อบกพร่องที่ทำนายถูกต้อง



รูปที่ 14 ตัวอย่างการคำนวณหาค่าเฉลี่ยความเที่ยงตรง (AP)

(2) ค่าเฉลี่ยการขึ้นต่อกัน (Average Dependency: AD) เป็นมาตรวัดที่วัดค่าเฉลี่ยผลกระทบการขึ้นต่อกันของคลาสที่ให้ความสำคัญในอันดับสูงของการทำนายข้อบกพร่องที่ถูกต้อง สามารถคำนวณหาค่าได้ดังสมการ (14) ทั้งนี้ได้ยกตัวอย่างของการคำนวณค่าเฉลี่ยการขึ้นต่อกันแสดงดังรูปที่ 15

$$Average\ Dependency\ (AD) = \frac{\sum_{j=1}^{n}\left(\frac{CDI_{true}(k)}{m_k}\right)}{n} \qquad (14)$$

โดยที่ $CDI_{true}(k)$ คือ ค่าผลกระทบการขึ้นต่อกันของคลาส (CDI) ของข้อบกพร่องที่ทำนายถูกต้อง ณ ข้อบกพร่องลำดับที่ k

m_k คือ จำนวนข้อบกพร่องทั้งหมด ณ ข้อบกพร่องลำดับที่ k

n คือ ลำดับของข้อบกพร่องทั้งหมด

ลำดับ	ผลการทำนาย	CDI
1	✗	0.50
2	✓	0.68
3	✓	0.34
4	✓	0.85
5	✗	0.39

ผลการทำนาย
✗ = 0
✓ = 1
✓ = 1
✓ = 1
✗ = 0

CDI
0 * 0.50
1 * 0.68
1 * 0.34
1 * 0.85
0 * 0.39

โดยที่ ✓ คือ ทำนายว่ามีข้อบกพร่อง และ ✗ คือ ทำนายว่าไม่มีข้อบกพร่อง

จากตาราง สามารถคำนวณค่าเฉลี่ยความขึ้นต่อกัน (AD) ได้ดังนี้

$$AD = \frac{1}{5}\left(\frac{0+0.68}{2} + \frac{0+0.68+0.34}{3} + \frac{0+0.68+0.34+0.85}{4} + \frac{0+0.68+0.34+0.85+0}{5}\right) =$$

ดังนั้น ค่าเฉลี่ยความขึ้นต่อกัน (AD) เท่ากับ 0.304

รูปที่ 15 ตัวอย่างการคำนวณหาค่าเฉลี่ยความขึ้นต่อกัน (AD)

(3) อัตราเฉลี่ยประสิทธิภาพ (Average Performance Rate: APR) เป็นมาตรวัดค่าเฉลี่ยที่ให้ความสำคัญกับค่าเฉลี่ยความเที่ยงตรงและค่าเฉลี่ยการขึ้นต่อกันอย่างเท่าๆกัน สามารถคำนวณได้ดังสมการ (15)

$$Average\ Performance\ Rate\ (APR) = \frac{2 \times AP \times AD}{AP + AD} \qquad (15)$$

3.4.2 การสร้างและประเมินผลแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน

ขั้นตอนนี้ผู้วิจัยได้นำเสนอวิธีการเรียนรู้แบบกึ่งมีผู้สอนในการสร้างแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่อง อีกทั้งยังทำการเปรียบเทียบกับวิธีการจำแนกประเภทแบบดั้งเดิมอีกด้วย จากนั้นจึงทำการประเมินผลแบบจำลองที่สร้างขึ้นทั้งหมดด้วยมาตรวัดประสิทธิภาพตามที่ได้กำหนดไว้ ซึ่งในขั้นตอนนี้ประกอบด้วยขั้นตอนย่อย 2 ขั้นตอน คือ 1) การสร้างแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน และ 2) การประเมินผลแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอนโดยสามารถอธิบายรายละเอียดของขั้นตอนย่อยต่างๆ ได้ดังนี้

1) การสร้างแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน

Algorithm	Pseudo code for OS-YATSI algorithm
Input:	A set of label $L = \{l_1, l_2, l_3\}$, classifier C, labeled data D_l, unlabeled data D_u, oversampling ratio R_{os}, oversampling labeled data D_{osl}, number of nearest neighbors K, $N = \|D_{osl}\|$, $M = \|D_u\|$, unlabeled data example d_u

Step1:	Find the majority class l_M with $\|D_M\|$ examples in the labeled data D_l
	Create a set of minority classes L_m that excludes the majority class l_M
	While(L_m is not empty)
	Find the class l_M in L_m with the least number of examples, $\|D_m\|$
	Compute the number of examples $\|D'_m\|$ if oversampling using SMOTE with R_{os}
	If (Diff($\|D'_m\|$, $\|D_M\|$) < Diff($\|D_m\|$, $\|D_M\|$)) Then
	Oversampling the class using SMOTE with R_{os}
	Add the new oversampled example into D_{osl}
	Else
	Remove class l_M from a set of classes L_m

Step2:	Use the classifier C to construct the initial model M_1 by using D_{osl}
	Use the M_1 to "pre-label" all the examples of D_u
	For(i=1 to N)
	Weight = 1.0
	For(j=1 to M)
	Weight = $(N/M) * WeightFactor\ F$
	Combine D_{osl} and D_u to generate D
	For every example in D_u
	Find the K-nearest neighbors to the example from D to produce set D_{kNN}
	For i=1 to K
	If(class of D_{kNN} = 1) sum weight1 of D_{kNN}
	If(class of D_{kNN} = 2) sum weight2 of D_{kNN}
	If(class of D_{kNN} = 3) sum weight 3of D_{kNN}
	Predict the actual class with the largest total weighting score

Step3:	For unlabeled data D_u Find the class with smallest amount of example and produce set C_{small} For another class Select examples equally to C_{small} with their prediction score and produce set $C_{balance}$ Merge C_{small} and $C_{balance}$ to produce balance unlabeled data D'_u

รูปที่ 16 รหัสเทียม (Pseudo Code) ของอัลกอริทึม OS-YATSI

รูปที่ 17 ขั้นตอนการทำงานของวิธีการที่นำเสนอ

ในงานวิจัยนี้ได้ประยุกต์ใช้แนวคิดของวิธีการเรียนรู้แบบกึ่งมีผู้สอน และนำเสนออัลกอริทึมใหม่ที่มีชื่อว่า “OS-YATSI” เพื่อสร้างแบบจำลองการจัดลำดับความสำคัญข้อบกพร่องของซอฟต์แวร์ ทั้งนี้ได้เขียนรหัสเทียม หรือ ซูโดโค้ด (Pseudo Code) เพื่อบอกลำดับขั้นตอนการทำงานของอัลกอริทึมที่นำเสนอ ดังแสดงรายละเอียดในรูปที่ 16 และแสดงขั้นตอนการทำงานของวิธีการที่นำเสนอดังรูปที่ 17 ซึ่งประกอบด้วยขั้นตอนหลัก 3 ขั้นตอน คือ 1) การสุ่มเพิ่มตัวอย่างข้อมูลกลุ่มน้อย (Oversampling) 2) วิธีการเรียนรู้แบบกึ่งมีผู้สอน (Semi-Supervised Learning) และ 3) เกณฑ์การคัดเลือกข้อมูลที่ไม่มีฉลากประเภท (Unlabeled Selection Criteria: USC) โดยสามารถอธิบายรายละเอียดของแต่ละขั้นตอนได้ดังนี้

(1) การสุ่มเพิ่มตัวอย่างข้อมูลกลุ่มน้อย (Oversampling)

ขั้นตอนนี้มีวัตถุประสงค์เพื่อหลีกเลี่ยงความลำเอียงจากกลุ่มของระดับความรุนแรงที่มีเสียงข้างมาก (majority severity level) ในงานวิจัยนี้เลือกใช้วิธีการ SMOTE (Synthetic Minority Over-sampling Technique) [15] ซึ่งเป็นเทคนิคการสุ่มเพิ่มตัวอย่างข้อมูลกลุ่มน้อย โดยการสังเคราะห์ตัวอย่างจากคลาสที่มีเสียงข้างน้อยให้เพิ่มขึ้นจนมีปริมาณใกล้เคียงกับคลาสที่มีเสียงข้างมาก กระบวนการสังเคราะห์ตัวอย่างใหม่จะคำนวณได้จากสมการ (16) สามารถอธิบายรายละเอียดได้ดังนี้

$$x_{new} = x_i + (\hat{x}_i - x)(r) \qquad (16)$$

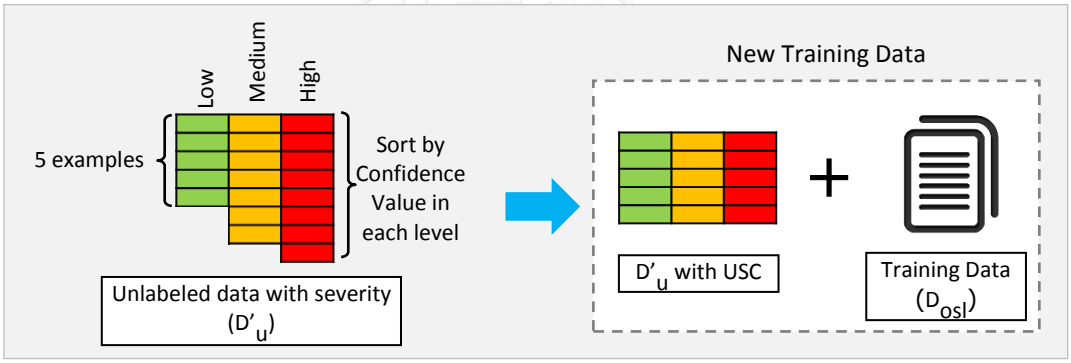
- <u>ขั้นตอนที่ 1</u> สุ่มเลือกคลาสที่มีเสียงข้างน้อย (x_i) มาหนึ่งคลาสเพื่อพิจารณา จากนั้นหากลุ่มของคลาสเพื่อนบ้านที่ใกล้เคียงกับ x_i มากที่สุดโดยใช้วิธีการ k-NN ที่วัดจากระยะห่างยูคลิเดียน
- <u>ขั้นตอนที่ 2</u> สุ่มเลือกคลาสจากกลุ่มเพื่อนบ้านที่ใกล้เคียงที่สุด ($\hat{x}_i$) มาหนึ่งคลาสและสร้างตัวอย่างใหม่ (x_{new}) โดยสุ่มตำแหน่งบนระยะทางระหว่างคลาสทั้งสองตัว (x_i กับ $\hat{x}_i$) โดยที่ r คือ ค่าการสุ่ม มีค่าตั้งแต่ 0-1
- <u>ขั้นตอนที่ 3</u> กระบวนการนี้จะสังเคราะห์ตัวอย่างใหม่ซ้ำไปเรื่อยๆ จนกระทั่งคลาสที่มีเสียงข้างน้อยทั้งหมดถูกใช้เป็นคลาสที่สุ่มมาพิจารณา และมีจำนวนตัวอย่างเท่ากันกับคลาสอื่นๆ

(2) วิธีการเรียนรู้แบบกึ่งมีผู้สอน (Semi-Supervised Learning)

ในคลังเก็บข้อบกพร่อง (Defect Repositories) ข้อบกพร่องที่ถูกพบส่วนใหญ่มักจะไม่มีการระบุระดับความรุนแรง (unlabeled data) มีเพียงแค่บางส่วนเท่านั้นที่มีการระบุถึงระดับความรุนแรง (labeled data) ดังนั้น ขั้นตอนนี้จึงมุ่งเน้นการใช้ประโยชน์จากข้อบกพร่องที่ไม่มีการระบุระดับความรุนแรง โดยใช้อัลกอริทึม YATSI (Yet Another Two Stage Idea) [14] ซึ่งมีกระบวนการทำงาน 2 ขั้นตอน ดังที่ได้กล่าวไปแล้วในบทที่ 2 หัวข้อ 2.1.3

(3) เกณฑ์การคัดเลือกข้อมูลที่ไม่มีฉลากประเภท (Unlabeled Selection Criteria: USC)

หลังจากขั้นตอนที่ (2) เราจะทราบระดับความรุนแรงที่แท้จริงของข้อมูลที่ไม่มีฉลากประเภท เพื่อเป็นการเพิ่มปริมาณข้อมูลสอน เราจะรวมข้อมูลนี้กับข้อมูลที่มีฉลากประเภทที่ผ่านการสุ่มตัวอย่างจากขั้นตอนที่ (1) อย่างไรก็ตาม หากเพิ่มข้อมูลที่ไม่มีฉลากประเภททั้งหมดลงในข้อมูลสอน จะทำให้ข้อมูลสอนเกิดปัญหาความไม่สมดุล ดังนั้น เพื่อหลีกเลี่ยงปัญหาดังกล่าวงานวิจัยนี้จึงนำเสนอเกณฑ์การคัดเลือกข้อมูลที่ไม่มีฉลากประเภท ซึ่งประกอบด้วย 2 ขั้นตอนย่อย ดังแสดงในรูปที่ 18 สามารถอธิบายรายละเอียดได้ดังนี้



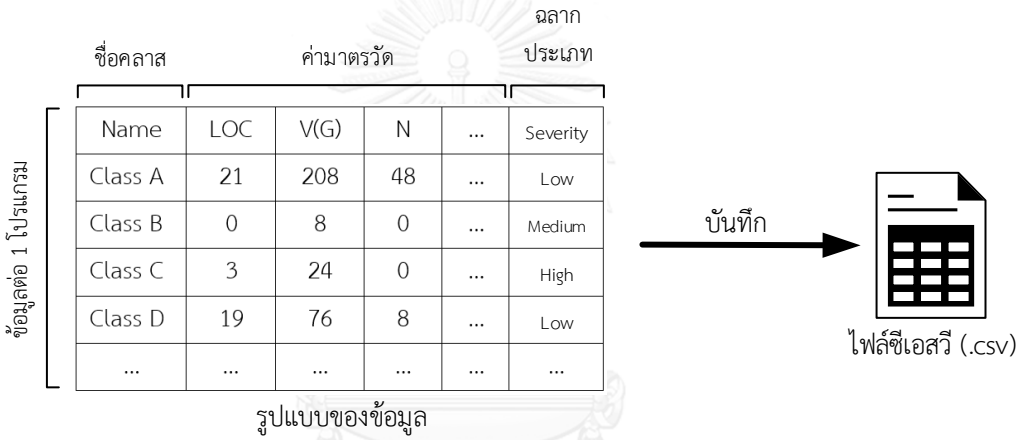
รูปที่ 18 ขั้นตอนของเกณฑ์การคัดเลือกข้อมูลที่ไม่มีฉลากประเภท

- ขั้นตอนที่ 1 พิจารณาระดับความรุนแรงจากข้อมูลที่ไม่มีฉลากประเภทที่ถูกระบุระดับความรุนแรงแล้ว (D'_u) ในขั้นตอนที่สอง (Semi-Supervised Learning) ว่าระดับความรุนแรงใดที่มีจำนวนตัวอย่างน้อยที่สุด จากนั้นให้เพิ่มตัวอย่างทั้งหมดของระดับความรุนแรงนั้นลงในข้อมูลสอน (จากรูปที่ 18 จะเห็นว่าระดับความรุนแรงระดับ Low เป็นระดับที่มีจำนวนตัวอย่างน้อยที่สุด คือ 5 ตัวอย่าง ดังนั้นจึงเพิ่มตัวอย่างทั้งหมดของความรุนแรงระดับ Low ลงในข้อมูลสอนที่ผ่านการสุ่มตัวอย่างแล้ว)

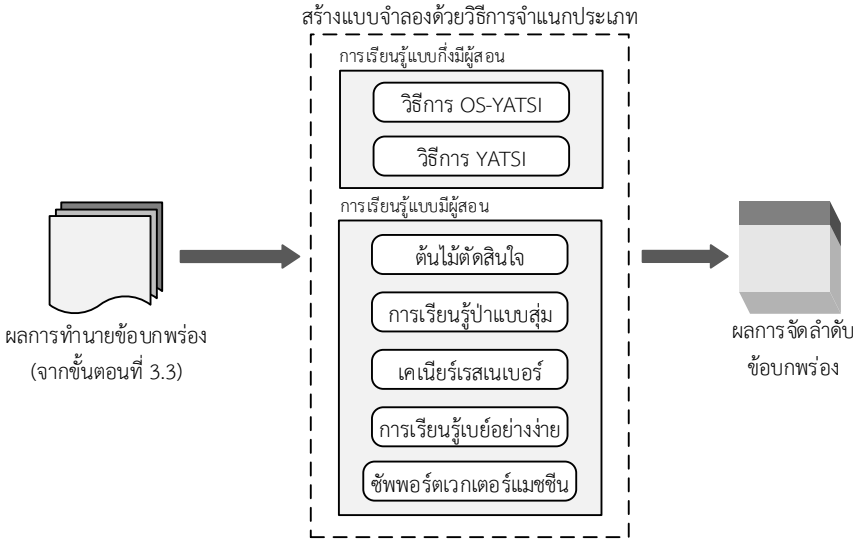
- ขั้นตอนที่ 2 สำหรับระดับความรุนแรงที่เหลืออยู่ ให้เลือกจำนวนตัวอย่างของแต่ละระดับความรุนแรงให้เท่ากับจำนวนตัวอย่างของระดับความรุนแรงที่มีจำนวนน้อยที่สุด (ขั้นตอนที่ 1)

โดยพิจารณาเลือกตัวอย่างตามค่าความมั่นใจของตัวอย่างนั้นๆจากมากไปน้อยตามลำดับ แล้วเพิ่มตัวอย่างลงในข้อมูลสอน (จากรูปที่ 18 ให้เลือกตัวอย่างของความรุนแรงระดับที่ 2 และ 3 ตามค่าความมั่นใจจากมากไปน้อย มาอย่างละ 5 ตัวอย่าง แล้วเพิ่มตัวอย่างที่เลือกมาลงในข้อมูลสอน)

การสร้างแบบจำลองในหัวข้อนี้ ข้อมูลที่ใช้จะมีรูปแบบเหมือนกับหัวข้อ 3.2 คือ อยู่ในรูปแบบไฟล์ซีเอสวี (.csv) และแยกเป็นแต่ละรายโปรแกรม ซึ่งประกอบด้วย ชื่อคลาสที่เกิดข้อบกพร่อง ค่ามาตรวัดที่ได้จากรหัสต้นฉบับ และฉลากประเภทระดับความรุนแรง ดังแสดงในรูปที่ 19 โดยฉลากประเภทระดับความรุนแรงของชุดโปรแกรมทั้งหมดในขั้นตอนนี้จะนำมาจากแหล่งข้อมูลสาธารณะ [37] ซึ่งประกอบด้วยความรุนแรง 3 ระดับ คือ ระดับต่ำ (Low) ระดับปานกลาง (Medium) และระดับสูง (High)



รูปที่ 19 รูปแบบของข้อมูลสำหรับการทำนายข้อบกพร่อง



รูปที่ 20 กระบวนการสร้างแบบจำลองการทำนายข้อบกพร่อง

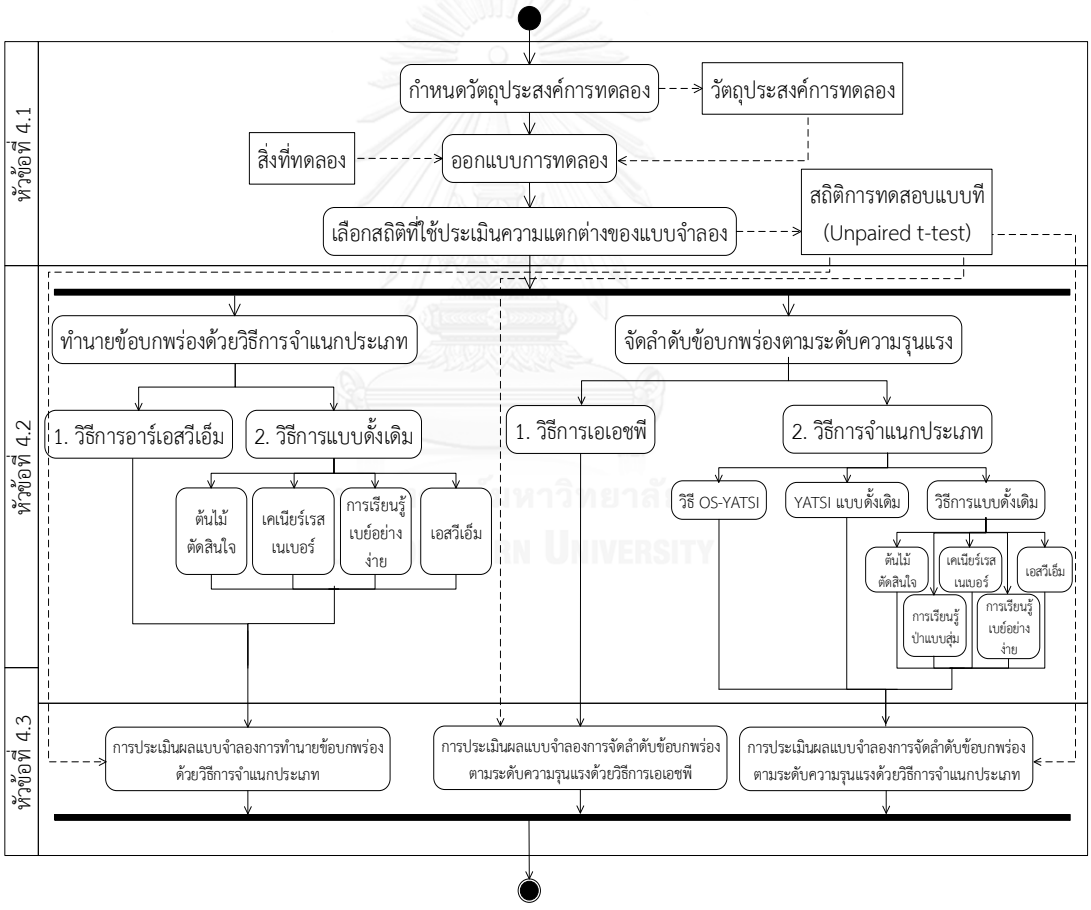
นอกจากนี้ผู้วิจัยได้สร้างแบบจำลองด้วยวิธีการจำแนกประเภทแบบอื่นๆอีกจำนวน 6 แบบจำลอง ดังแสดงในรูปที่ 20 ซึ่งประกอบด้วย 1) YATSI แบบดั้งเดิม 2) ต้นไม้ตัดสินใจ 3) การเรียนรู้ป่าแบบสุ่ม 4) การเรียนรู้ด้วยเพื่อนบ้านใกล้ที่สุด 5) การเรียนรู้เบย์แบบง่าย และ 6) ซัพพอร์ตเวกเตอร์แมชชีน เพื่อเปรียบเทียบประสิทธิภาพกับวิธีการที่ได้นำเสนอ

2) การประเมินผลแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน

ขั้นตอนนี้ใช้สำหรับการประเมิลผลแบบจำลองที่สร้างขึ้นในขั้นตอนที่ 3.4.2 จากนั้นทำการประเมินผลโดยใช้มาตรวัดประสิทธิภาพการทำนาย ซึ่งมาตรวัดที่ใช้ในขั้นตอนนี้จะคล้ายคลึงกับมาตรวัดประสิทธิภาพการทำนายของแบบจำลองการทำนายข้อบกพร่องในหัวข้อ 3.3.2 ซึ่งประกอบไปด้วย ความเที่ยงตรง (Precision) ความครบถ้วน (Recall) และมัชฌิมฮาร์โมนิค (F-measure) อย่างไรก็ตามมาตรวัดเหล่านี้เป็นการประเมินประสิทธิภาพสำหรับการทำนายข้อมูลที่มี 2 กลุ่มเท่านั้น ซึ่งอาจไม่เหมาะสมหากใช้ในการประเมินประสิทธิภาพของแบบจำลองในขั้นตอนนี้ที่มีข้อมูลระดับความรุนแรง 3 กลุ่ม คือ ระดับต่ำ (Low) ระดับปานกลาง (Medium) และระดับสูง (High) ดังนั้น งานวิจัยนี้จึงได้ประเมินประสิทธิภาพของแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอนด้วย ค่าเฉลี่ยมหภาค (Macro-averaging) และ ค่าเฉลี่ยจุลภาค (Micro-averaging) ดังที่ได้อธิบายรายละเอียดในบทที่ 2 หัวข้อ 2.1.7 ตารางที่ 10

บทที่ 4
การทดลองและการประเมินผล

ในบทนี้จะกล่าวถึงรายละเอียดในการทดลอง เพื่อประเมินประสิทธิภาพของแบบจำลองการทำนายข้อบกพร่องของซอฟต์แวร์และแบบจำลองการจัดลำดับข้อบกพร่องของซอฟต์แวร์ตามระดับความรุนแรงที่ได้นำเสนอในงานวิจัย โดยใช้มาตรวัดประสิทธิภาพ นอกจากนี้ยังทำการเปรียบเทียบผลลัพธ์ของวิธีการที่นำเสนอกับแบบจำลองอื่นๆ โดยเริ่มจากการกล่าวถึงรายละเอียดของการวางแผนการทดลอง การดำเนินการทดลอง ผลการทดลอง และการประเมินผลแบบจำลอง ซึ่งสามารถแสดงรายละเอียดภาพรวมของการทดลองด้วยแผนภาพกิจกรรมดังรูปที่ 21



รูปที่ 21 ภาพรวมของการทดลอง

4.1 การวางแผนการทดลอง

4.1.1 วัตถุประสงค์ของการทดลอง

การทดลองนี้มีวัตถุประสงค์เพื่อประเมินประสิทธิภาพของแบบจำลองที่นำเสนอไปใช้ในการทำนายและจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงในบริบทของงานวิจัย โดยทำการเปรียบเทียบผลลัพธ์และประเมินผลแบบจำลองที่นำเสนอในงานวิจัยกับแบบจำลองอื่นๆ ด้วยวิธีการทางสถิติ

4.1.2 สิ่งที่ทดลอง

สิ่งที่ทดลองในการทดลองนี้ คือ ชุดโปรแกรมจากแหล่งข้อมูล PROMISE [33] จำนวนทั้งหมด 15 โปรแกรม ซึ่งประกอบด้วยโปรแกรมประยุกต์ใช้งานในด้านต่างๆ ชุดโปรแกรมทั้งหมดมีการพัฒนาด้วยภาษาจาวา สถิติของแต่ละชุดโปรแกรมสามารถแสดงรายละเอียดดังตารางที่ 17

ตารางที่ 17 สถิติข้อบกพร่องของชุดโปรแกรมที่ใช้ในการทดลอง

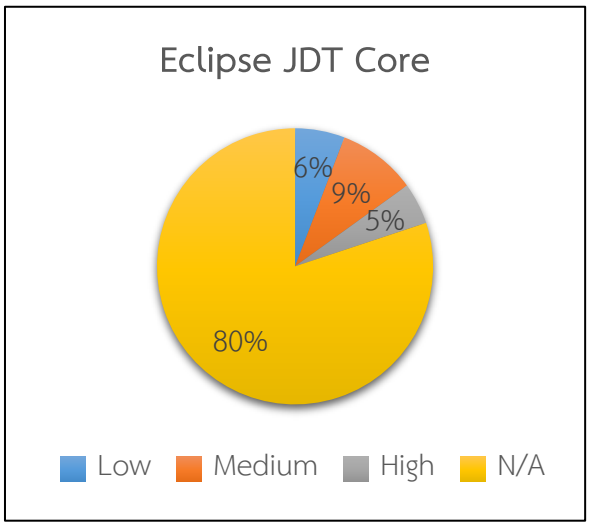
ลำดับ	ชื่อโปรแกรม	จำนวนคลาส	จำนวนคลาสที่มีข้อบกพร่อง	จำนวนคลาสที่ไม่มีข้อบกพร่อง	อัตราส่วนคลาสที่เกิดข้อบกพร่อง
1	Ant	125	20	105	16.00%
2	Camel	608	216	392	35.53%
3	Ivy	352	40	312	11.36%
4	Jedit	272	90	182	33.09%
5	Log4j	135	34	101	25.19%
6	Lucene	195	91	104	46.67%
7	Pbeans	26	20	6	76.92%
8	Poi	237	141	96	59.49%
9	Synapse	157	16	141	10.19%
10	Velocity	229	78	151	34.06%
11	Xalan	723	110	613	15.21%
12	Xerces	440	71	369	16.14%
13	Eclipse JDT Core	997	206	791	20.66%
14	Eclipse PDE UI	1,497	209	1,288	13.96%
15	Mylyn	1,862	245	1,617	13.16%
ค่าเฉลี่ย		7,855	1,587	6,268	28.51%

จากสถิติของชุดโปรแกรมในตารางที่ 17 จะพบว่าชุดโปรแกรมเหล่านี้ประสบกับปัญหาความไม่สมดุลของคลาส โดยมีค่าเฉลี่ยอัตราส่วนคลาสที่เกิดข้อบกพร่องทั้งหมดเท่ากับ 28.51% และมีค่าอัตราส่วนคลาสที่เกิดข้อบกพร่องน้อยที่สุดเท่ากับ 10.19% ในชุดโปรแกรม Synapse

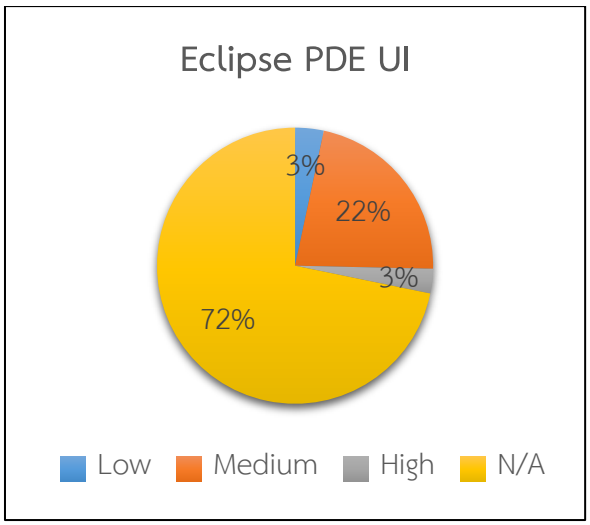
นอกจากนี้ ผู้วิจัยยังได้เลือกชุดโปรแกรมจำนวน 3 โปรแกรมจากตารางที่ 17 ซึ่งประกอบด้วยโปรแกรมที่ 13, 14 และ 15 มาใช้เป็นข้อมูลสอนในการทดลองแบบจำลองการจัดลำดับข้อบกพร่องตามระดับความรุนแรงด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน เนื่องจากโปรแกรมดังกล่าวมีข้อมูลฉลากประเภท [37] ที่บ่งบอกถึงระดับความรุนแรงของคลาสที่เกิดข้อบกพร่อง ซึ่งประกอบด้วยความรุนแรงจำนวน 3 ระดับ คือ ระดับต่ำ ระดับกลาง และระดับสูง ดังแสดงรายละเอียดสถิติของชุดโปรแกรมในตารางที่ 18 และแสดงสัดส่วนข้อมูลของโปรแกรมทั้งสามได้ด้วยแผนภูมิวงกลมดังรูปที่ 22 - 24

ตารางที่ 18 สถิติระดับความรุนแรงของข้อบกพร่องของชุดโปรแกรมที่ใช้ในการทดลอง

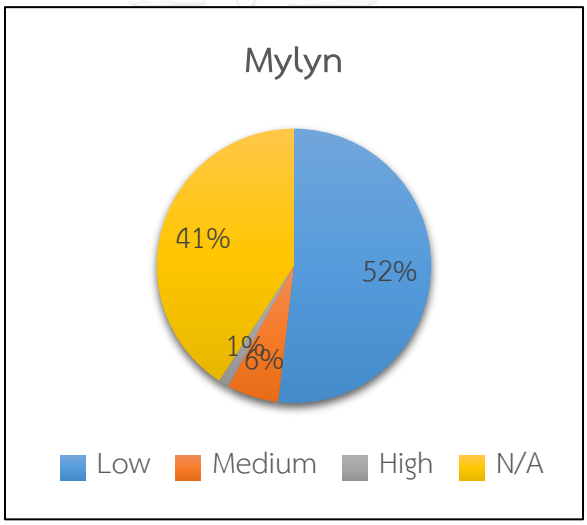
ลำดับ	ชื่อโปรแกรม	จำนวนข้อบกพร่อง	จำนวนข้อบกพร่องตามระดับความรุนแรง				อัตราส่วนระดับความรุนแรง
			ระดับต่ำ	ระดับกลาง	ระดับสูง	ไม่ระบุ	
1	Eclipse JDT Core	206	12	19	10	165	19.90%
2	Eclipse PDE UI	209	7	46	6	150	28.23%
3	Mylyn	245	127	15	3	100	59.18%
Average		220	48.67	26.67	6.33	138.33	35.77%



รูปที่ 22 แผนภูมิวงกลมแสดงสัดส่วนข้อมูลของโปรแกรม Eclipse JDT Core



รูปที่ 23 แผนภูมิวงกลมแสดงสัดส่วนข้อมูลของโปรแกรม Eclipse PDE UI



รูปที่ 24 แผนภูมิวงกลมแสดงสัดส่วนข้อมูลของโปรแกรม Mylyn

4.1.3 สถิติที่ใช้สำหรับการประเมินความแตกต่างของแบบจำลอง

สถิติที่ใช้ในงานวิจัยนี้นำมาใช้ในการประเมินผลแบบจำลองการทำนายข้อบกพร่องและแบบจำลองการจัดลำดับข้อบกพร่องตามระดับความรุนแรง คือ สถิติการทดสอบแบบที โดยใช้ Unpaired t-test [38] ในการทดสอบความแตกต่างค่าเฉลี่ยของกลุ่มตัวอย่างสองกลุ่มที่เป็นอิสระต่อกัน ซึ่งผู้วิจัยได้ตั้งสมมุติฐานทางการวิจัยไว้ดังนี้

$$H_0 : \mu_1 = \mu_2$$

$$H_1 : \mu_1 > \mu_2$$

โดยที่ μ_1 คือ ค่าเฉลี่ยประสิทธิภาพของแบบจำลองของวิธีการที่นำเสนอ

μ_2 คือ ค่าเฉลี่ยประสิทธิภาพของแบบจำลองของวิธีการแบบดั้งเดิม

จากสมมุติฐานทางการวิจัยดังกล่าวจะพบว่ามีตัวแปรที่เกี่ยวข้อง 2 ตัว คือ 1) วิธีการที่ใช้สร้างแบบจำลอง (วิธีการที่นำเสนอ และวิธีการแบบดั้งเดิม) 2) ประสิทธิภาพของแบบจำลอง (วัดออกมาเป็นตัวเลข) ทั้งนี้ผู้วิจัยได้ทำการเปรียบเทียบความแตกต่างโดยทดสอบที่ระดับความเชื่อมั่น 95% และคาดหวังว่าวิธีการที่นำเสนอจะมีค่าแตกต่างอย่างมีนัยสำคัญ

4.2 การดำเนินการทดลอง

4.2.1 การทำนายข้อบกพร่องด้วยวิธีการจำแนกประเภท

ในขั้นตอนนี้จะทำการทดลองการทำนายข้อบกพร่องด้วยแบบจำลองวิธีการจำแนกประเภท จำนวน 5 แบบจำลอง คือ 1) อาร์เอสวีเอ็ม 2) ต้นไม้ตัดสินใจ 3) เคเนียร์เรสเนเบอร์ 4) การเรียนรู้เบย์แบบง่าย และ 5) เอสวีเอ็ม เริ่มต้นจากการเตรียมข้อมูลด้วยการแปลงข้อมูล (scaling) [39] ทุกๆ คุณลักษณะ (attribute) ให้มีช่วงเท่ากันอยู่ในระหว่าง 0 ถึง 1 จากนั้นจึงเปรียบเทียบประสิทธิภาพในการทำนายของทุกๆแบบจำลอง สามารถอธิบายรายละเอียดได้ตามขั้นตอนดังนี้

1) หาวิธีการจำแนกประเภทแบบดั้งเดิมที่ให้ประสิทธิภาพดีที่สุด โดยการเปรียบเทียบประสิทธิภาพของแบบจำลองจำนวน 4 แบบจำลอง คือ ต้นไม้ตัดสินใจ เคเนียร์เรสเนเบอร์ การเรียนรู้เบย์แบบง่าย และเอสวีเอ็ม

2) เปรียบเทียบวิธีการอาร์เอสวีเอ็มและวิธีการจำแนกประเภทแบบดั้งเดิม (ข้อ 1) และประเมินผลด้วยการทดสอบทีที่ระดับความเชื่อมั่น 95%

4.2.2 การจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรง

4.2.2.1 จัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์

ในขั้นตอนนี้จะทำการทดลองจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยวิธีการเอเอชพี โดยมีการเปรียบเทียบประสิทธิภาพของแบบจำลองที่มีเกณฑ์การตัดสินใจแตกต่างกัน 3 รูปแบบ คือ

1) มี 1 เกณฑ์การตัดสินใจ คือ ค่าความมั่นใจของการเกิดข้อบกพร่อง (Confidence Value)

2) มี 1 เกณฑ์การตัดสินใจ คือ ค่าผลกระทบความขึ้นต่อกันของคลาส (Class Dependency Impact: CDI)

3) มี 2 เกณฑ์การตัดสินใจ คือ ค่าความมั่นใจของการเกิดข้อบกพร่อง (Confidence Value) และ ค่าผลกระทบความขึ้นต่อกันของคลาส (Class Dependency Impact: CDI)

การประเมินผลแบบจำลองนั้น จะประเมินผลแบบจำลองตามเกณฑ์การตัดสินใจทั้ง 3 รูปแบบ ด้วยมาตรวัดประสิทธิภาพ คือ อัตราเฉลี่ยประสิทธิภาพ (APR) ดังที่ได้อธิบายรายละเอียดในหัวข้อ 3.4.1

4.2.2.2 จัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน

ในขั้นตอนนี้จะทำการทดลองการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยวิธีการจำแนกประเภทจำนวน 7 แบบจำลอง คือ 1) OS-YATSI 2) YATSI แบบดั้งเดิม 3) ต้นไม้ตัดสินใจ 4) การเรียนรู้ป่าแบบสุ่ม 5) เคเนียร์เรสเนเบอร์ 6) การเรียนรู้เบย์แบบง่าย และ 7) เอสวีเอ็ม เริ่มต้นจากการเตรียมข้อมูลด้วยการแปลงข้อมูล (scaling) เช่นเดียวกับหัวข้อ 4.2.1 จากนั้นจึงเปรียบเทียบประสิทธิภาพในการทำนายของทุกๆแบบจำลอง สามารถอธิบายรายละเอียดได้ตามขั้นตอนดังนี้

1) หาวิธีการจำแนกประเภทแบบดั้งเดิมที่ให้ประสิทธิภาพดีที่สุด โดยการเปรียบเทียบประสิทธิภาพของแบบจำลองจำนวน 5 แบบจำลอง คือ ต้นไม้ตัดสินใจ การเรียนรู้ป่าแบบสุ่ม เคเนียร์เรสเนเบอร์ การเรียนรู้เบย์แบบง่าย และเอสวีเอ็ม

2) วิเคราะห์เกณฑ์การคัดเลือกข้อมูลที่ไม่มีฉลากประเภท (Unlabeled Selection Criteria: USC) ว่ามีผลต่อประสิทธิภาพของ OS-YATSI หรือไม่

3) เปรียบเทียบวิธีการ OS-YATSI (ข้อ 2) กับวิธีการจำแนกประเภทแบบดั้งเดิม (ข้อ 1) นอกจากนี้ยังเปรียบเทียบกับวิธีการ YATSI แบบดั้งเดิมอีกด้วย แล้วจึงประเมินผลแบบจำลองด้วยการทดสอบทีที่ระดับความเชื่อมั่น 95%

4.3 ผลการทดลองและการประเมินผล

ในส่วนนี้จะแสดงผลลัพธ์จากการทดลองและการประเมินผลของแบบจำลองที่สร้างขึ้นทั้งหมดในงานวิจัยนี้ โดยแบ่งผลการทดลองออกเป็น 3 ส่วน คือ 1) ผลการทดลองและการประเมินผลแบบจำลองการทำนายข้อบกพร่อง 2) ผลการทดลองและการประเมินผลแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์ และ 3) ผลการทดลองและการประเมินผลแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน

4.3.1 ผลการทดลองและการประเมินผลแบบจำลองการทำนายข้อบกพร่อง

4.3.1.1 ผลการทดลองแบบจำลองการทำนายข้อบกพร่อง

ในส่วนนี้จะแสดงผลลัพธ์จากการทำนายข้อบกพร่อง ซึ่งประกอบด้วย ชื่อของคลาสที่เกิดข้อบกพร่อง และค่าความมั่นใจของการเกิดข้อบกพร่อง (P-Score) ที่ได้จากแบบจำลอง เนื่องจากข้อมูลที่ใช้มีจำนวนมากจึงยกตัวอย่างมาเพียง 1 โปรแกรม เพื่อให้ทราบรายละเอียดของผลลัพธ์จากแบบจำลองการทำนายข้อบกพร่อง โดยได้ยกตัวอย่างโปรแกรม Eclipse JDT Core ซึ่งแสดงรายละเอียดผลลัพธ์จากการทำนายดังตารางที่ 19

ตารางที่ 19 ตัวอย่างผลลัพธ์จากการทำนายข้อบกพร่อง

ลำดับ	ชื่อคลาส	ค่าความมั่นใจ P-Score
1	org::eclipse::jdt::internal::compiler::ast::ForeachStatement	0.905
2	org::eclipse::jdt::internal::core::DeltaProcessor	1.000
3	org::eclipse::jdt::internal::compiler::ast::MessageSend	0.905
4	org::eclipse::jdt::core::JavaCore	1.000
5	org::eclipse::jdt::internal::core::util::Util	1.000
6	org::eclipse::jdt::internal::eval::CodeSnippetQualifiedNameReference	1.000
7	org::eclipse::jdt::internal::compiler::flow::ExceptionHandlingFlowContext	0.750
8	org::eclipse::jdt::internal::core::hierarchy::TypeHierarchy	1.000
9	org::eclipse::jdt::internal::codeassist::CompletionEngine	1.000
10	org::eclipse::jdt::internal::core::search::matching::ClassFileMatchLocator	1.000
11	org::eclipse::jdt::internal::compiler::lookup::TypeVariableBinding	1.000
12	org::eclipse::jdt::internal::core::util::PublicScanner	1.000
13	org::eclipse::jdt::internal::core::search::matching::VariablePattern	0.842
14	org::eclipse::jdt::internal::core::search::matching::TypeReferenceLocator	0.933
15	org::eclipse::jdt::internal::core::dom::rewrite::ASTRewriteAnalyzer	1.000
16	org::eclipse::jdt::internal::core::search::matching::ConstructorPattern	1.000
17	org::eclipse::jdt::internal::compiler::ast::ASTNode	1.000
18	org::eclipse::jdt::internal::core::SourceMapper	0.933
19	org::eclipse::jdt::internal::core::JavaProject	1.000
20	org::eclipse::jdt::internal::compiler::parser::JavadocParser	1.000
21	org::eclipse::jdt::internal::compiler::ast::QualifiedNameReference	1.000

ตารางที่ 19 ตัวอย่างผลลัพธ์จากการทำนายข้อบกพร่อง (ต่อ)

ลำดับ	ชื่อคลาส	ค่าความมั่นใจ P-Score
22	org::eclipse::jdt::internal::formatter::CodeFormatterVisitor	1.000
23	org::eclipse::jdt::internal::core::Openable	1.000
24	org::eclipse::jdt::internal::compiler::util::Messages	1.000
25	org::eclipse::jdt::internal::core::util::HandleFactory	0.889
...	...	...
141	org::eclipse::jdt::internal::core::hierarchy::HierarchyResolver	0.909

4.3.1.2 การประเมินผลแบบจำลองการทำนายข้อบกพร่อง

ในส่วนนี้จะแสดงผลลัพธ์จากการประเมินผลแบบจำลองการทำนายข้อบกพร่องโดยใช้ข้อมูลเกี่ยวกับการจำแนกข้อบกพร่องจากเมตริกซ์ความสับสน แล้วคำนวณหามาตรวัดประสิทธิภาพเพื่อประเมินประสิทธิภาพของแบบจำลอง ซึ่งประกอบด้วยมาตรวัด คือ ความครบถ้วน (Probability of Detection: PD) ความน่าจะเป็นของการเตือนผิดพลาด (Probability of False Alarm: PF) มัชฌิมฮาร์โมนิค (F-measure: F1) และมัชฌิมเรขาคณิต (G-mean) ดังที่ได้อธิบายรายละเอียดในบทที่ 2 หัวข้อ 2.1.7 ตารางที่ 9 โดยสามารถอธิบายรายละเอียดของการประเมินผลได้ดังนี้

- <u>การประเมินผลแบบจำลองที่สร้างด้วยวิธีการจำแนกประเภทแบบดั้งเดิม</u> ในส่วนนี้จะเปรียบเทียบประสิทธิภาพของแบบจำลองจำนวน 4 แบบจำลอง คือ ต้นไม้ตัดสินใจ (DT) เคเนียร์เรสเนเบอร์ (k-NN) การเรียนรู้เบย์แบบง่าย (NB) และเอสวีเอ็ม (SVM)

ตารางที่ 20 – 23 แสดงค่ามาตรวัดประสิทธิภาพ PD, PF, F1 และ G-mean ตามลำดับ ที่คำนวณโดยใช้ข้อมูลเกี่ยวกับการจำแนกข้อบกพร่องจากเมตริกซ์ความสับสน จากผลการทดลองจะพบว่า DT และ k-NN แสดงให้เห็นถึงประสิทธิภาพการทำนายที่ดีที่สุดเป็นส่วนใหญ่ของชุดโปรแกรมทั้งหมด ในขณะที่ NB และ SVM มีประสิทธิภาพในการทำนายข้อบกพร่องอยู่ในระดับปานกลางและค่อนข้างต่ำในบางชุดโปรแกรม โดยเฉพาะในชุดโปรแกรม Velocity วิธีการ NB มีประสิทธิภาพการตรวจจับข้อบกพร่องเพียง 25.20% เท่านั้น

ตารางที่ 20 ค่า PD ของแบบจำลองการทำนายข้อบกพร่อง

Project	Prediction model			
	DT	*k-NN*	*NB*	*SVM*
Ant	**0.857**	0.747	0.774	0.765
Camel	0.684	0.646	**0.749**	**0.749**
Ivy	**0.891**	0.878	0.492	0.621
Jedit	0.803	0.853	**0.863**	0.814
Log4j	0.751	0.773	**0.922**	0.843
Lucene	0.601	0.786	**0.875**	0.731
Pbeans	**0.900**	**0.900**	0.700	0.850
Poi	0.702	0.680	0.283	**0.794**
Synapse	**0.809**	0.780	0.687	0.808
Velocity	0.735	**0.740**	0.252	0.714
Xalan	0.825	0.829	**0.876**	0.744
Xerces	**0.824**	0.718	0.743	0.582
Eclipse JDT Core	0.836	0.855	**0.946**	0.881
Eclipse PDE UI	0.866	0.768	**0.925**	0.823
Mylyn	0.866	0.799	**0.911**	0.832
Avg.	0.797	0.783	0.733	0.770
SD	0.085	0.072	0.224	0.084

ตารางที่ 21 ค่า PF ของแบบจำลองการทำนายข้อบกพร่อง

Project	Prediction model			
	DT	*k-NN*	*NB*	*SVM*
Ant	0.161	**0.066**	0.181	0.134
Camel	**0.321**	0.365	0.755	0.646
Ivy	0.154	0.195	0.150	**0.138**

(หมายเหตุ: ในแต่ละแถวของตารางค่าที่มีลักษณะเป็นตัวหนา คือ วิธีการที่ให้ประสิทธิภาพสูงสุด)

ตารางที่ 21 ค่า PF ของแบบจำลองการทำนายข้อบกพร่อง (ต่อ)

Project	Prediction model			
	DT	*k-NN*	*NB*	*SVM*
Jedit	**0.210**	0.243	0.448	0.232
Log4j	0.228	**0.138**	0.406	0.286
Lucene	**0.301**	0.344	0.541	0.313
Pbeans	0.050	0.300	**0.000**	0.200
Poi	0.270	0.164	**0.086**	0.363
Synapse	0.114	**0.100**	0.200	0.157
Velocity	0.238	0.272	**0.112**	0.318
Xalan	**0.152**	0.180	0.618	0.277
Xerces	**0.100**	0.144	0.575	0.348
Eclipse JDT Core	0.169	**0.154**	0.643	0.402
Eclipse PDE UI	0.143	**0.120**	0.725	0.492
Mylyn	0.135	**0.129**	0.676	0.449
Avg.	0.183	0.194	0.408	0.317
SD	0.077	0.091	0.262	0.142

ตารางที่ 22 ค่า F1 ของแบบจำลองการทำนายข้อบกพร่อง

Project	Prediction model			
	DT	*k-NN*	*NB*	*SVM*
Ant	0.790	**0.816**	0.784	0.799
Camel	0.608	**0.642**	0.629	0.619
Ivy	**0.870**	0.846	0.596	0.703
Jedit	0.798	**0.814**	0.748	0.795
Log4j	0.756	**0.807**	0.793	0.793
Lucene	0.623	**0.742**	0.725	0.719

(หมายเหตุ: ในแต่ละแถวของตารางค่าที่มีลักษณะเป็นตัวหนา คือ วิธีการที่ให้ประสิทธิภาพสูงสุด)

ตารางที่ 22 ค่า F1 ของแบบจำลองการทำนายข้อบกพร่อง (ต่อ)

Project	Prediction model			
	DT	*k-NN*	*NB*	*SVM*
Pbeans	0.583	**0.837**	0.767	0.827
Poi	0.706	0.704	0.405	**0.710**
Synapse	0.816	**0.826**	0.725	0.820
Velocity	0.682	**0.720**	0.355	0.699
Xalan	**0.734**	0.726	0.702	**0.734**
Xerces	**0.856**	0.771	0.639	0.602
Eclipse JDT Core	0.833	**0.851**	0.731	0.772
Eclipse PDE UI	**0.862**	0.813	0.698	0.711
Mylyn	**0.742**	0.729	0.704	0.730
Avg.	0.751	0.776	0.667	0.736
SD	0.094	0.062	0.129	0.067

ตารางที่ 23 ค่า G-mean ของแบบจำลองการทำนายข้อบกพร่อง

Project	Prediction model			
	DT	*k-NN*	*NB*	*SVM*
Ant	0.814	**0.905**	0.881	0.893
Camel	0.776	**0.783**	0.616	0.714
Ivy	**0.931**	0.914	0.788	0.847
Jedit	0.806	**0.832**	0.817	0.827
Log4j	0.855	**0.857**	0.850	0.827
Lucene	0.580	**0.783**	0.705	0.718
Pbeans	0.785	**0.951**	0.858	0.867
Poi	0.819	0.851	0.749	**0.827**
Synapse	0.897	**0.900**	0.819	0.898

(หมายเหตุ: ในแต่ละแถวของตารางค่าที่มีลักษณะเป็นตัวหนา คือ วิธีการที่ให้ประสิทธิภาพสูงสุด)

ตารางที่ 23 ค่า G-mean ของแบบจำลองการทำนายข้อบกพร่อง (ต่อ)

Project	Prediction model			
	DT	*k-NN*	*NB*	*SVM*
Velocity	0.856	**0.873**	0.867	0.816
Xalan	**0.913**	0.907	0.743	0.851
Xerces	**0.927**	0.882	0.797	0.762
Eclipse JDT Core	0.812	**0.892**	0.733	0.848
Eclipse PDE UI	**0.828**	0.806	0.702	0.801
Mylyn	**0.730**	0.713	0.725	0.720
Avg.	0.822	0.857	0.777	0.814
SD	0.089	0.063	0.075	0.060

(หมายเหตุ: ในแต่ละแถวของตารางค่าที่มีลักษณะเป็นตัวหนา คือ วิธีการที่ให้ประสิทธิภาพสูงสุด)

เนื่องจากมาตรวัด F1 และ G-mean เป็นมาตรวัดประสิทธิภาพที่เหมาะสมสำหรับข้อมูลที่มีปัญหาความไม่สมดุลกันของคลาส ดังนั้น เราจะเลือกแบบจำลองการทำนายข้อบกพร่องที่ให้ประสิทธิภาพดีที่สุดในแต่ละซอฟต์แวร์ โดยพิจารณาจากค่า F1 และ G-mean จากตารางที่ 22 และตารางที่ 23 ซึ่งสามารถสรุปได้ดังตารางที่ 24

ตารางที่ 24 แบบจำลองการทำนายข้อบกพร่องแบบดั้งเดิมที่ให้ประสิทธิภาพดีที่สุด

Project	Winner Method	F1	G-mean
Ant	*k-NN*	0.816	0.905
Camel	*k-NN*	0.642	0.783
Ivy	*DT*	0.870	0.931
Jedit	*k-NN*	0.814	0.832
Log4j	*k-NN*	0.807	0.857
Lucene	*k-NN*	0.742	0.783
Pbeans	*k-NN*	0.837	0.951
Poi	*SVM*	0.710	0.827
Synapse	*k-NN*	0.826	0.900
Velocity	*k-NN*	0.720	0.873

ตารางที่ 24 แบบจำลองการทำนายข้อบกพร่องแบบดั้งเดิมที่ให้ประสิทธิภาพดีที่สุด (ต่อ)

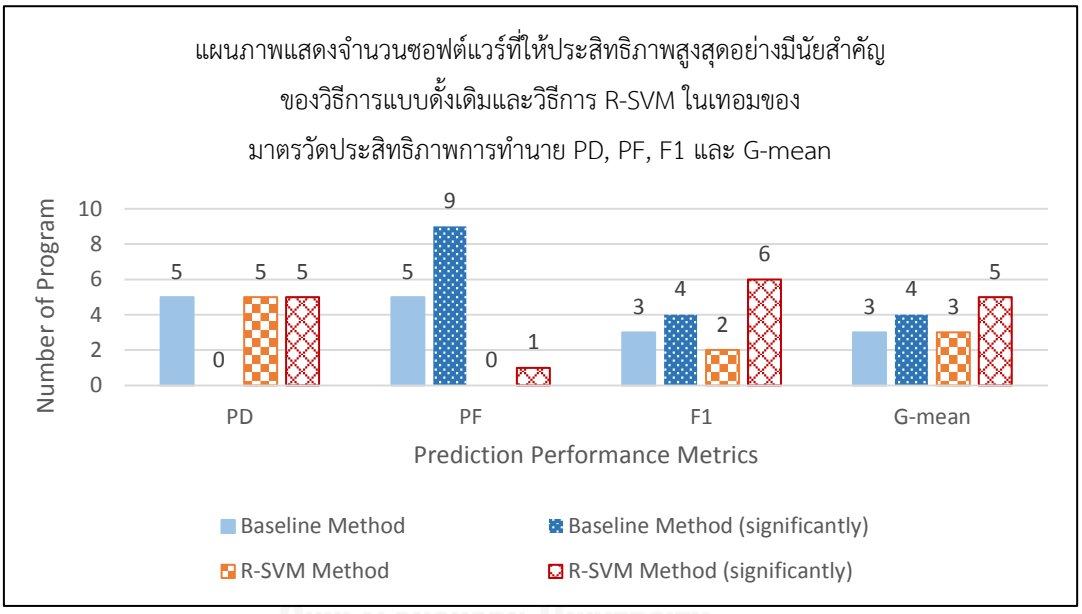
Project	Winner Method	F1	G-mean
Xalan	*DT*	0.734	0.913
Xerces	*DT*	0.856	0.927
Eclipse JDT Core	*k-NN*	0.851	0.892
Eclipse PDE UI	*DT*	0.862	0.828
Mylyn	*DT*	0.742	0.730
Avg.	-	0.789	0.862
SD	-	0.068	0.064

ตารางที่ 25 การเปรียบเทียบประสิทธิภาพการทำนายของแบบจำลองระหว่างวิธี R-SVM กับวิธีแบบดั้งเดิม

Project	PD		PF		F1		G-mean	
	Baseline	R-SVM	Baseline	R-SVM	Baseline	R-SVM	Baseline	R-SVM
Ant	0.857	**0.902**	**0.066****	0.275	0.816	**0.832****	**0.905**	0.886
Camel	0.749	**0.905***	**0.321**	0.471	0.642	**0.650**	0.783	**0.804****
Ivy	**0.891**	0.831	**0.138****	0.313	**0.870****	0.796	**0.931**	0.864
Jedit	**0.863**	0.840	**0.210**	0.224	**0.814**	0.785	0.832	**0.878****
Log4j	0.922	**0.983***	**0.138***	0.297	**0.807**	0.775	0.857	**0.860**
Lucene	**0.875**	0.941	**0.301**	0.334	**0.742**	0.692	0.783	**0.806***
Pbeans	0.900	**1.000**	**0.000**	0.003	0.837	**0.893**	0.951	**0.952**
Poi	0.794	**0.811**	**0.086***	0.279	0.710	**0.714***	0.827	**0.943****
Synapse	0.809	**0.875**	**0.100****	0.211	0.826	**0.836****	0.900	**0.906**
Velocity	0.740	**0.784**	0.112	**0.041***	0.720	**0.731****	**0.873**	0.837
Xalan	**0.876**	0.843	**0.152****	0.385	0.734	**0.757****	**0.913****	0.845
Xerces	0.824	**0.858***	**0.100**	0.121	**0.856****	0.703	**0.927***	0.874
Eclipse JDT Core	0.946	**0.990****	**0.154****	0.372	**0.851****	0.744	**0.892***	0.840
Eclipse PDE UI	**0.925**	0.914	**0.120****	0.627	**0.862****	0.716	**0.828***	0.744
Mylyn	0.911	**0.954****	**0.129****	0.535	0.742	**0.767****	0.730	**0.805****
Avg.	0.859	0.874	0.174	0.318	0.763	0.750	0.852	0.861
SD	0.063	0.066	0.125	0.161	0.061	0.072	0.069	0.054

(* และ ** แสดงถึงความแตกต่างอย่างมีนัยสำคัญที่ระดับความเชื่อมั่น 95% และ 99% ตามลำดับ)

- <u>การประเมินผลแบบจำลองที่สร้างด้วยวิธีการอาร์เอสวีเอ็ม</u> ในส่วนนี้จะประเมินผลประสิทธิภาพของแบบจำลองด้วยวิธีการ R-SVM และเปรียบเทียบกับแบบจำลองการทำนายข้อบกพร่องแบบดั้งเดิมที่ให้ประสิทธิภาพดีที่สุด ดังแสดงในตารางที่ 25 ซึ่งแสดงการเปรียบเทียบประสิทธิภาพของแบบจำลองด้วยมาตรวัดประสิทธิภาพ PD, PF, F1 และ G-mean ตามลำดับ ในแต่ละแถวค่าที่มีลักษณะเป็นตัวหนาคือวิธีการที่ให้ประสิทธิภาพสูงสุด จากผลการทดลองจะพบว่า วิธี R-SVM มีประสิทธิภาพดีกว่าวิธีแบบดั้งเดิมเป็นส่วนใหญ่ของซอฟต์แวร์ทั้งหมด เพื่อให้เห็นผลลัพธ์ได้อย่างชัดเจน จึงสรุปเป็นแผนภาพแสดงจำนวนซอฟต์แวร์ที่มีประสิทธิภาพดีกว่าอย่างมีนัยสำคัญของวิธีแบบดั้งเดิม และวิธี R-SVM ในแต่ละมาตรวัดได้ดังรูปที่ 25



รูปที่ 25 จำนวนซอฟต์แวร์ที่ให้ประสิทธิภาพสูงสุดอย่างมีนัยสำคัญของวิธีแบบดั้งเดิมและวิธี R-SVM ในเทอมของมาตรวัดประสิทธิภาพการทำนาย PD, PF, F1 และ G-mean

จากรูปที่ 25 สามารถสรุปผลตามมาตรวัดประสิทธิภาพการทำนายได้ดังนี้

(1) มาตรวัดความครบถ้วน (Probability of Detection: PD) วิธี R-SVM ให้ประสิทธิภาพดีกว่าวิธีการแบบดั้งเดิมจำนวน 5 :ซอฟต์แวร์ และดีกว่าวิธีการแบบดั้งเดิมอย่างมีนัยสำคัญจำนวน 5 ซอฟต์แวร์

(2) ความน่าจะเป็นของการเตือนผิดพลาด (Probability of False Alarm: PF) วิธีแบบดั้งเดิมให้ประสิทธิภาพดีกว่าวิธี R-SVM อย่างมีนัยสำคัญจำนวน 9 ซอฟต์แวร์

(3) มัชฌิมฮาร์โมนิค (F-measure: F1) วิธี R-SVM ให้ประสิทธิภาพดีกว่าวิธีการแบบดั้งเดิมจำนวน 2 ซอฟต์แวร์ และดีกว่าวิธีการแบบดั้งเดิมอย่างมีนัยสำคัญจำนวน 6 ซอฟต์แวร์

(4) มัชฌิมเรขาคณิต (G-mean) วิธี R-SVM ให้ประสิทธิภาพดีกว่าวิธีการแบบดั้งเดิมจำนวน 3 ซอฟต์แวร์ และดีกว่าวิธีการแบบดั้งเดิมอย่างมีนัยสำคัญจำนวน 5 ซอฟต์แวร์

4.3.2 ผลการทดลองและการประเมินผลแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์

4.3.2.1 ผลการทดลองแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์

ในส่วนนี้จะแสดงผลลัพธ์จากการจัดลำดับความสำคัญของข้อบกพร่องด้วยวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์ ซึ่งประกอบด้วย ลำดับการแก้ไขคลาสที่เกิดข้อบกพร่อง ชื่อของคลาสที่เกิดข้อบกพร่อง ระดับความรุนแรง และค่า AHP ผลลัพธ์ในส่วนนี้จะจัดลำดับข้อบกพร่องตามค่าระดับความรุนแรงและค่า AHP ตามลำดับ โดยระดับความรุนแรงที่แสดงนี้จะนำมาจากค่า CDI ที่ถูกแปลงเป็นระดับความรุนแรงจำนวน 3 ระดับ ส่วนค่า AHP ได้มาจากการคำนวณหาน้ำหนักความสำคัญโดยรวมในแต่ละข้อบกพร่อง ดังที่ได้อธิบายรายละเอียดในบทที่ 3 หัวข้อ 3.4.1

เนื่องจากข้อมูลที่ใช้มีจำนวนมากจึงยกตัวอย่างมาเพียง 1 โปรแกรม เพื่อให้ทราบรายละเอียดของผลลัพธ์จากแบบจำลองการทำนายข้อบกพร่อง โดยได้ยกตัวอย่างโปรแกรม Eclipse JDT Core ซึ่งแสดงรายละเอียดผลลัพธ์จากการทำนายดังตารางที่ 26

ตารางที่ 26 ตัวอย่างผลลัพธ์จากการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์

ลำดับการแก้ไข	ชื่อคลาส	ระดับความรุนแรง	ค่า AHP
1	org::eclipse::jdt::internal::compiler::lookup::TypeVariableBinding	High	0.834
2	org::eclipse::jdt::internal::compiler::ast::ASTNode	High	0.834
3	org::eclipse::jdt::internal::compiler::lookup::ReferenceBinding	High	0.834
4	org::eclipse::jdt::internal::compiler::lookup::LookupEnvironment	High	0.834
5	org::eclipse::jdt::internal::compiler::util::Util	High	0.834
6	org::eclipse::jdt::internal::core::JavaModelOperation	High	0.834
7	org::eclipse::jdt::internal::compiler::impl::IntConstant	High	0.834
8	org::eclipse::jdt::internal::compiler::codegen::BranchLabel	High	0.834
9	org::eclipse::jdt::internal::compiler::problem::ProblemReporter	High	0.834
10	org::eclipse::jdt::core::Flags	High	0.834
11	org::eclipse::jdt::core::dom::ASTNode	High	0.805
12	org::eclipse::jdt::internal::compiler::ast::Expression	High	0.805

ตารางที่ 26 ตัวอย่างผลลัพธ์จากการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์ (ต่อ)

ลำดับการแก้ไข	ชื่อคลาส	ระดับความรุนแรง	ค่า AHP
13	org::eclipse::jdt::internal::compiler::impl::CompilerOptions	High	0.789
14	org::eclipse::jdt::internal::compiler::lookup::Binding	High	0.784
15	org::eclipse::jdt::internal::compiler::parser::RecoveryScannerData	High	0.784
16	org::eclipse::jdt::core::compiler::CategorizedProblem	High	0.784
17	org::eclipse::jdt::internal::compiler::lookup::MethodBinding	High	0.784
18	org::eclipse::jdt::internal::compiler::impl::Constant	High	0.779
19	org::eclipse::jdt::internal::core::JavaElement	High	0.766
20	org::eclipse::jdt::internal::compiler::codegen::CodeStream	High	0.758
21	org::eclipse::jdt::internal::compiler::lookup::Scope	High	0.758
22	org::eclipse::jdt::internal::compiler::lookup::BlockScope	High	0.758
23	org::eclipse::jdt::internal::compiler::lookup::TypeBinding	High	0.720
24	org::eclipse::jdt::internal::compiler::CompilationResult	High	0.720
25	org::eclipse::jdt::internal::compiler::lookup::MethodScope	High	0.681
...	...	...	...
141	org::eclipse::jdt::core::dom::DefaultCommentMapper	Low	0.378

4.3.2.2 การประเมินผลแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์

ในส่วนนี้จะแสดงผลลัพธ์จากการประเมินผลแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยมาตรวัดประสิทธิภาพอัตราเฉลี่ยประสิทธิภาพ (Average Performance Rate: APR) ดังที่ได้อธิบายรายละเอียดในบทที่ 3 หัวข้อ 3.4.1 โดยสามารถอธิบายรายละเอียดของการประเมินผลได้ดังนี้

- <u>การประเมินผลแบบจำลองที่สร้างด้วยวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์</u> ในส่วนนี้จะเปรียบเทียบประสิทธิภาพของแบบจำลองที่มีเกณฑ์การตัดสินใจแตกต่างกันทั้ง 3 รูปแบบ คือ

1) R1 ค่าความมั่นใจของการเกิดข้อบกพร่อง (Confidence Value)

2) R2 ค่าผลกระทบความขึ้นต่อกันของคลาส (Class Dependency Impact: CDI)

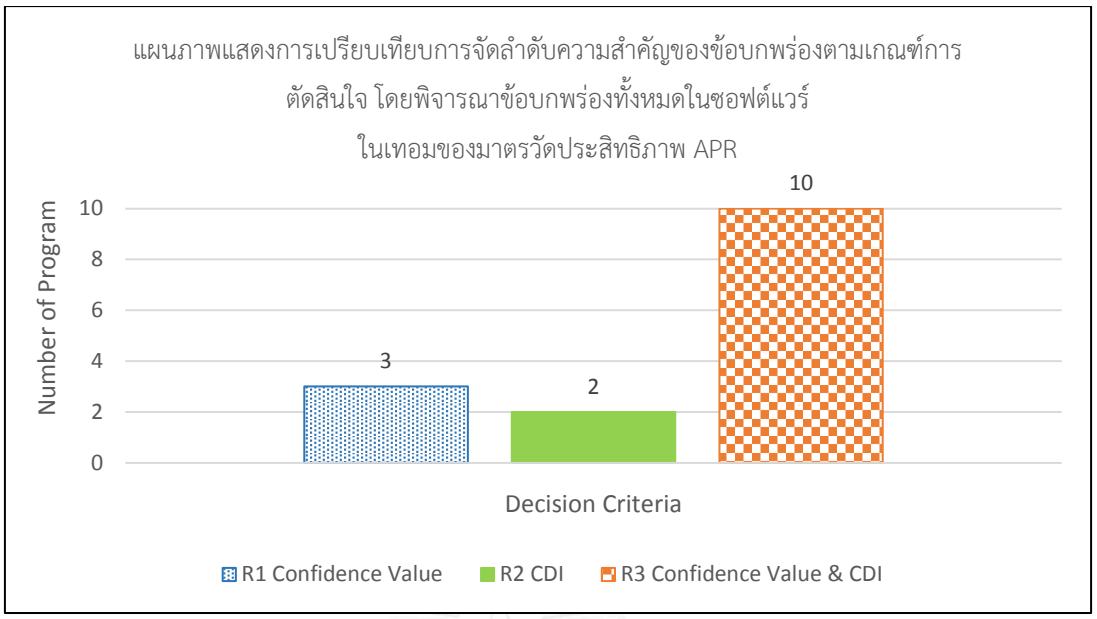
3) R3 ค่าความมั่นใจของการเกิดข้อบกพร่อง (Confidence Value) และ ค่าผลกระทบความขึ้นต่อกันของคลาส (Class Dependency Impact: CDI)

ตารางที่ 27 ค่า APR ของแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์

Project	All		
	R1	R2	R3
Ant	0.426	0.387	**0.483**
Camel	0.467	0.563	**0.593**
Ivy	0.469	**0.504**	0.430
Jedit	**0.446**	0.379	0.419
Log4j	0.476	0.484	**0.537**
Lucene	0.181	0.201	**0.252**
Pbeans	0.574	0.763	**0.778**
Poi	0.350	0.411	**0.491**
Synapse	**0.061**	0.037	0.056
Velocity	0.538	0.730	**0.738**
Xalan	**0.073**	0.014	0.051
Xerces	0.016	0.087	**0.148**
Eclipse JDT Core	0.342	0.273	**0.426**
Eclipse PDE UI	0.477	0.402	**0.571**
Mylyn	0.607	**0.677**	0.619

(หมายเหตุ: ในแต่ละแถวของตารางค่าที่มีลักษณะเป็นตัวหนา คือ วิธีการที่ให้ประสิทธิภาพสูงสุด)

ตารางที่ 27 แสดงการเปรียบเทียบประสิทธิภาพของแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องด้วยมาตรวัดประสิทธิภาพ APR เมื่อวิเคราะห์ผลลัพธ์ในตารางจะพบว่า การจัดลำดับความสำคัญตามเกณฑ์การตัดสินใจ R3 จะให้ประสิทธิภาพดีกว่าเกณฑ์การตัดสินใจ R1 และ R2 ในส่วนใหญ่ของจำนวนซอฟต์แวร์ทั้งหมด เนื่องจากมีการพิจารณาทั้งโอกาสในการเกิดข้อบกพร่อง และผลกระทบของข้อบกพร่องที่เกิดขึ้นที่ส่งผลต่อซอฟต์แวร์ เพื่อให้เห็นผลลัพธ์ได้อย่างชัดเจน จึงสรุปเป็นแผนภาพแสดงจำนวนซอฟต์แวร์ที่ให้ประสิทธิภาพสูงในการจัดลำดับความสำคัญตามเกณฑ์การตัดสินใจทั้ง 3 แบบได้ดังรูปที่ 26



รูปที่ 26 จำนวนซอฟต์แวร์ที่ให้ประสิทธิภาพสูงในการจัดลำดับความสำคัญตามเกณฑ์การตัดสินใจทั้ง 3 แบบ โดยพิจารณาข้อบกพร่องทั้งหมดภายในซอฟต์แวร์ ในเทอมของมาตรวัด APR

จากรูปที่ 26 เมื่อพิจารณาข้อบกพร่องทั้งหมดภายในซอฟต์แวร์ สามารถสรุปผลตามเกณฑ์การตัดสินใจได้ดังนี้

(1) เกณฑ์การตัดสินใจ R1 ให้ประสิทธิภาพดีกว่าเกณฑ์การตัดสินใจ R2 และ R3 จำนวน 3 ซอฟต์แวร์ ในเทอมของมาตรวัดประสิทธิภาพ APR

(2) เกณฑ์การตัดสินใจ R2 ให้ประสิทธิภาพดีกว่าเกณฑ์การตัดสินใจ R1 และ R3 จำนวน 2 ซอฟต์แวร์ ในเทอมของมาตรวัดประสิทธิภาพ APR

(3) เกณฑ์การตัดสินใจ R3 ให้ประสิทธิภาพดีกว่าเกณฑ์การตัดสินใจ R1 และ R2 จำนวน 10 ซอฟต์แวร์ ในเทอมของมาตรวัดประสิทธิภาพ APR ซึ่งเป็นเกณฑ์การตัดสินใจที่ให้ประสิทธิภาพดีที่สุด

4.3.3 ผลการทดลองและการประเมินผลแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน

4.3.3.1 ผลการทดลองแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน

ในส่วนนี้จะแสดงผลลัพธ์จากการจัดลำดับความสำคัญของข้อบกพร่อง ซึ่งประกอบด้วย ลำดับการแก้ไขคลาสที่เกิดข้อบกพร่อง ชื่อของคลาสที่เกิดข้อบกพร่อง ระดับความรุนแรง และค่าความมั่นใจ (S-Score) โดยระดับความรุนแรงและค่าความมั่นใจที่แสดงนี้ได้มาจากผลลัพธ์จากการทำนายของแบบจำลอง เนื่องจากข้อมูลที่ใช้มีจำนวนมากจึงยกตัวอย่างมาเพียง 1 โปรแกรม เพื่อให้

ทราบรายละเอียดของผลลัพธ์จากแบบจำลองการทำนายข้อบกพร่อง โดยได้ยกตัวอย่างโปรแกรม Eclipse JDT Core ซึ่งแสดงรายละเอียดผลลัพธ์จากการทำนายดังตารางที่ 28

ตารางที่ 28 ตัวอย่างผลลัพธ์จากการจัดลำดับข้อบกพร่องตามระดับความรุนแรงด้วยวิธีการจำแนกประเภท

ลำดับการแก้ไข	ชื่อคลาส	ระดับความรุนแรง	ค่าความมั่นใจ S-Score
1	org::eclipse::jdt::core::Flags	High	0.900
2	org::eclipse::jdt::internal::compiler::util::Messages	High	0.800
3	org::eclipse::jdt::internal::compiler::parser::JavadocParser	High	0.700
4	org::eclipse::jdt::internal::compiler::parser::RecoveryScannerData	High	0.700
5	org::eclipse::jdt::internal::formatter::comment::MultiCommentLine	High	0.700
6	org::eclipse::jdt::internal::core::util::LRUCache	High	0.700
7	org::eclipse::jdt::core::dom::DefaultCommentMapper	High	0.700
8	org::eclipse::jdt::internal::core::search::PatternSearchJob	High	0.700
9	org::eclipse::jdt::internal::core::JavaModelCache	High	0.700
10	org::eclipse::jdt::internal::compiler::flow::ExceptionHandlingFlowContext	High	0.600
11	org::eclipse::jdt::internal::compiler::CompilationResult	High	0.600
12	org::eclipse::jdt::core::compiler::CategorizedProblem	High	0.600
13	org::eclipse::jdt::internal::compiler::SourceElementParser	High	0.600
14	org::eclipse::jdt::internal::compiler::lookup::ClassScope	High	0.600
15	org::eclipse::jdt::internal::codeassist::complete::CompletionJavadocParser	High	0.600
16	org::eclipse::jdt::internal::core::search::JavaWorkspaceScope	High	0.600
17	org::eclipse::jdt::internal::core::search::matching::MatchLocatorParser	High	0.600
18	org::eclipse::jdt::internal::core::search::matching::FieldLocator	High	0.600
19	org::eclipse::jdt::internal::compiler::parser::TypeConverter	High	0.600
20	org::eclipse::jdt::internal::compiler::parser::RecoveredInitializer	High	0.600
21	org::eclipse::jdt::internal::compiler::ast::ForeachStatement	High	0.500

ตารางที่ 28 ตัวอย่างผลลัพธ์จากการจัดลำดับข้อบกพร่องตามระดับความรุนแรงด้วยวิธีการจำแนกประเภท (ต่อ)

ลำดับการแก้ไข	ชื่อคลาส	ระดับความรุนแรง	ค่าความมั่นใจ S-Score
22	org::eclipse::jdt::internal::compiler::ast::Javadoc	High	0.500
23	org::eclipse::jdt::internal::compiler::ast::SwitchStatement	High	0.500
24	org::eclipse::jdt::internal::compiler::ast::Assignment	High	0.500
25	org::eclipse::jdt::internal::compiler::ast::CastExpression	High	0.500
...	...	...	...
141	org::eclipse::jdt::core::dom::ASTConverter	Low	0.364

4.3.3.2 การประเมินผลแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน

ในส่วนนี้จะแสดงผลลัพธ์จากการประเมินผลแบบจำลองการทำนายข้อบกพร่องโดยใช้ข้อมูลเกี่ยวกับการจำแนกข้อบกพร่องจากเมตริกซ์ความสับสน แล้วคำนวณหามาตรวัดประสิทธิภาพเพื่อประเมินประสิทธิภาพของแบบจำลอง ซึ่งประกอบด้วยมาตรวัด คือ ความเที่ยงตรง (Pr) ความครบถ้วน (Re) และมัชฌิมฮาร์โมนิค (F1) แบบค่าเฉลี่ยมหภาค (Macro-averaging) และค่าเฉลี่ยจุลภาค (Micro-averaging) ดังที่อธิบายรายละเอียดในบทที่ 2 หัวข้อ 2.1.7 ตารางที่ 10 โดยสามารถอธิบายรายละเอียดของการประเมินผลได้ดังนี้

- <u>การประเมินผลแบบจำลองที่สร้างด้วยวิธีการจำแนกประเภทแบบดั้งเดิม</u> ในส่วนนี้จะเปรียบเทียบประสิทธิภาพของแบบจำลองทั้ง 5 คือ ต้นไม้ตัดสินใจ (DT) การเรียนรู้ป่าแบบสุ่ม (RF) เคเนียร์เรสเนเบอร์ (k-NN) การเรียนรู้เบย์แบบง่าย (NB) และเอสวีเอ็ม (SVM) ด้วยมาตรวัดประสิทธิภาพดังแสดงในตารางที่ 29 - 31 โดยแต่ละแถวของตารางค่าที่มีลักษณะเป็นตัวหนา คือ วิธีการที่ให้ประสิทธิภาพสูงสุด จากผลการทดลองจะพบว่า k-NN มีประสิทธิภาพดีที่สุดทั้งมาตรวัดในแบบ macro-average และ micro-average บนซอฟต์แวร์ Mylyn ในขณะที่ซอฟต์แวร์ส่วนที่เหลือจะได้รับประสิทธิภาพที่ดีจากวิธีการอื่นที่แตกต่างกัน นอกจากนี้จะเลือกวิธีการที่ให้ประสิทธิภาพสูงสุดทั้งมาตรวัดในแบบ macro-average และ micro-average โดยพิจารณาจากค่า F1-measure เพื่อนำไปใช้ในการเปรียบเทียบประสิทธิภาพกับงานวิจัยที่ได้นำเสนอ ซึ่งสามารถสรุปได้ดังตารางที่ 32

ตารางที่ 29 ค่า Precision ของแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน

Project	DT		RF		k-NN		NB		SVM	
	Macro	*Micro*	*Macro*	*Micro*	*Macro*	*Micro*	*Macro*	*Micro*	*Macro*	*Micro*
JDT Core	0.330	0.445	0.293	0.366	0.302	0.317	**0.419**	**0.344**	0.186	0.440
PDE UI	**0.263**	**0.630**	0.257	0.746	0.261	0.679	0.255	0.546	0.258	0.763
Mylyn	0.291	0.855	0.321	0.855	**0.361**	**0.758**	0.318	0.745	0.292	0.876
Avg.	0.295	0.643	0.202	0.656	0.308	0.585	0.331	0.545	0.245	0.693
SD	0.034	0.205	0.153	0.257	0.050	0.235	0.083	0.201	0.054	0.226

ตารางที่ 30 ค่า Recall ของแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน

Project	DT		RF		k-NN		NB		SVM	
	Macro	*Micro*	*Macro*	*Micro*	*Macro*	*Micro*	*Macro*	*Micro*	*Macro*	*Micro*
JDT Core	**0.393**	**0.445**	0.291	0.366	0.272	0.317	0.356	0.344	0.329	0.440
PDE UI	0.318	0.630	0.319	0.746	0.290	0.679	0.233	0.546	**0.326**	**0.763**
Mylyn	0.325	0.855	0.345	0.855	**0.347**	**0.758**	0.323	0.745	0.333	0.876
Avg.	0.345	0.643	0.318	0.656	0.303	0.585	0.304	0.545	0.329	0.693
SD	0.041	0.205	0.027	0.257	0.039	0.235	0.064	0.201	0.004	0.226

ตารางที่ 31 ค่า F1-measure ของแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน

Project	DT		RF		k-NN		NB		SVM	
	Macro	*Micro*	*Macro*	*Micro*	*Macro*	*Micro*	*Macro*	*Micro*	*Macro*	*Micro*
JDT Core	0.349	0.445	0.255	0.366	0.280	0.317	**0.350**	**0.344**	0.230	0.440
PDE UI	0.275	0.630	0.285	0.746	0.275	0.679	0.241	0.546	**0.288**	**0.763**
Mylyn	0.307	0.855	0.332	0.855	**0.351**	**0.758**	0.315	0.745	0.311	0.876
Avg.	0.310	0.643	0.291	0.656	0.302	0.585	0.302	0.545	0.276	0.693
SD	0.037	0.205	0.039	0.257	0.043	0.235	0.056	0.201	0.042	0.226

(หมายเหตุ: ในแต่ละแถวของตารางค่าที่มีลักษณะเป็นตัวหนา คือ วิธีการที่ให้ประสิทธิภาพสูงสุด)

ตารางที่ 32 ค่า F1-measure ของแบบจำลองการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรงด้วยวิธีการแบบดั้งเดิมที่ให้ประสิทธิภาพดีที่สุด

Project	Winner	F1	
		Macro	*Micro*
JDT Core	NB	0.350	0.344
PDE UI	SVM	0.288	0.763
Mylyn	k-NN	0.351	0.758
Avg.	-	0.330	0.622
SD	-	0.036	0.240

ตารางที่ 33 ค่า F1-measure ของการเปรียบเทียบแบบจำลอง OS-YATSI ระหว่างใช้และไม่ใช้ USC

Data	With USC		Without USC	
	Macro	*Micro*	*Macro*	*Micro*
JDT Core	**0.484**	**0.513**	0.459	0.491
PDE UI	**0.430**	**0.746**	0.290	0.629
Mylyn	**0.361**	**0.807**	0.349	0.800
Avg.	0.425	0.689	0.366	0.640
SD	0.062	0.155	0.086	0.155

(หมายเหตุ: ในแต่ละแถวของตารางค่าที่มีลักษณะเป็นตัวหนา คือ วิธีการที่ให้ประสิทธิภาพสูงสุด)

- <u>การประเมินแบบจำลองที่มีการวิเคราะห์ด้วย USC</u> การประเมินนี้มีจุดประสงค์เพื่อวิเคราะห์เกณฑ์การคัดเลือกข้อมูลที่ไม่มีฉลากประเภท (Unlabeled Selection Criteria: USC) ว่าสามารถจัดการกับปัญหาความไม่สมดุลและช่วยปรับปรุงประสิทธิภาพการทำนายของแบบจำลอง OS-YATSI ได้หรือไม่ โดยใช้วิธีแบบดั้งเดิมจากตารางที่ 32 เป็นตัวจำแนกของแบบจำลองของวิธี OS-YATSI ซึ่งผลลัพธ์จากตารางที่ 33 แสดงให้เห็นว่าแบบจำลอง OS-YATSI ที่ใช้ USC ให้ประสิทธิภาพดีกว่าแบบจำลองที่ไม่ใช้ USC ในทุกๆ ซอฟต์แวร์อย่างชัดเจน นอกจากนี้ผลลัพธ์ยังสะท้อนให้เห็นว่าข้อมูลที่ไม่มีฉลากประเภททุกๆ ตัวไม่สามารถปรับปรุงประสิทธิภาพการทำนายให้ดีขึ้นได้เสมอไป ยิ่งไปกว่านั้นอาจทำให้ประสิทธิภาพการทำนายลดลงอีกด้วย

ตารางที่ 34 การเปรียบเทียบค่า Precision ของวิธีการ OS-YATSI, YATSI และวิธีการแบบดั้งเดิม

Data	Baseline		YATSI		OS-YATSI	
	Macro	*Micro*	*Macro*	*Micro*	*Macro*	*Micro*
JDT Core	0.419	0.344	0.406	0.410	**0.514***	**0.513****
PDE UI	0.263	0.630	0.297	0.730	**0.436****	**0.746**
Mylyn	0.361	0.758	0.329	0.759	**0.426***	**0.807***
Avg.	0.348	0.577	0.344	0.633	0.459	0.689
SD	0.079	0.212	0.056	0.194	0.048	0.155

ตารางที่ 35 การเปรียบเทียบค่า Recall ของวิธีการ OS-YATSI, YATSI และวิธีการแบบดั้งเดิม

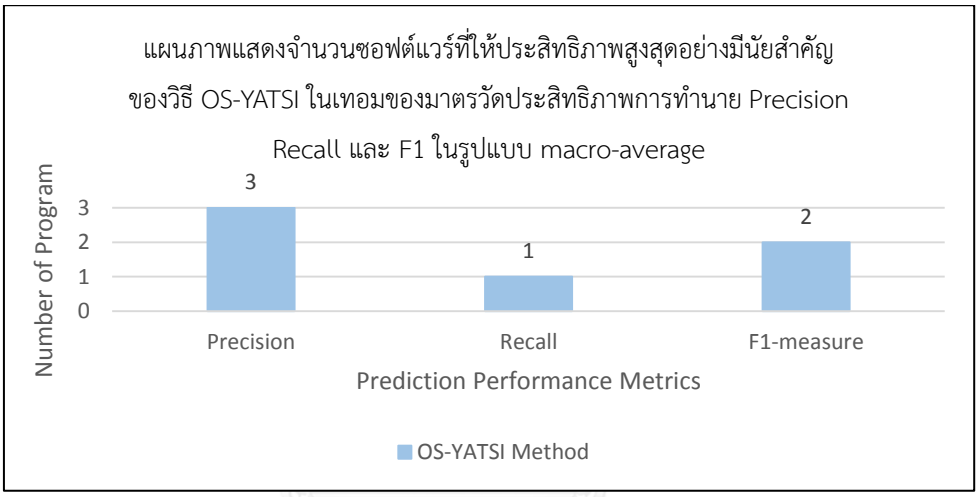
Data	Baseline		YATSI		OS-YATSI	
	Macro	*Micro*	*Macro*	*Micro*	*Macro*	*Micro*
JDT Core	0.393	0.445	0.390	0.410	**0.499**	**0.513***
PDE UI	0.326	**0.763**	0.360	0.730	**0.464***	0.746
Mylyn	0.347	0.758	0.348	0.759	**0.366**	**0.807***
Avg.	0.355	0.655	0.366	0.633	0.443	0.689
SD	0.034	0.182	0.022	0.194	0.069	0.155

ตารางที่ 36 การเปรียบเทียบค่า F1 ของวิธีการ OS-YATSI, YATSI และวิธีการแบบดั้งเดิม

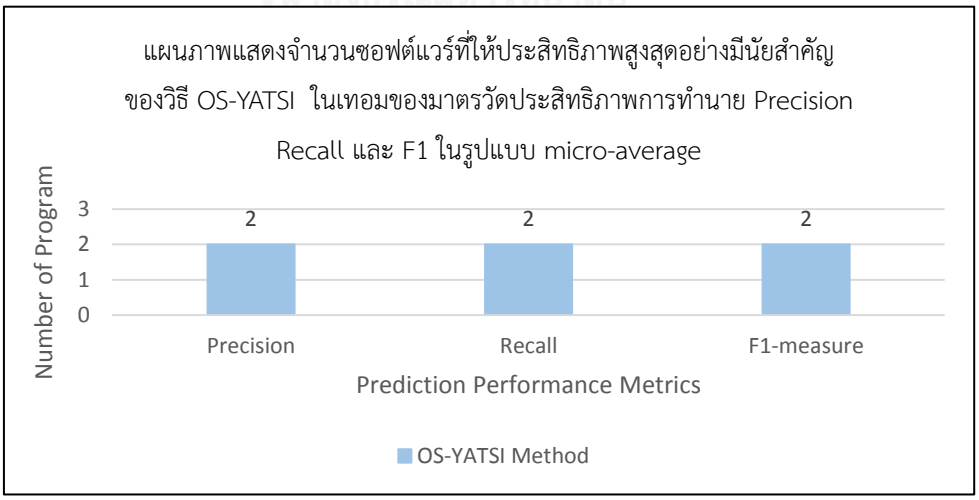
Data	Baseline		YATSI		OS-YATSI	
	Macro	*Micro*	*Macro*	*Micro*	*Macro*	*Micro*
JDT Core	0.350	0.344	0.381	0.410	**0.484***	**0.513****
PDE UI	0.288	**0.763**	0.324	0.730	**0.430***	0.746
Mylyn	0.351	0.758	0.330	0.759	**0.361**	**0.807***
Avg.	0.330	0.622	0.345	0.633	0.425	0.689
SD	0.036	0.240	0.031	0.194	0.062	0.155

(* และ ** แสดงถึงความแตกต่างอย่างมีนัยสำคัญที่ระดับความเชื่อมั่น 95% และ 99% ตามลำดับ)

- <u>การประเมินแบบจำลองที่สร้างด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน</u> ในขั้นตอนนี้จะเปรียบเทียบแบบจำลอง OS-YATSI กับแบบจำลองด้วยวิธีการแบบดั้งเดิมที่แสดงในตารางที่ 32 นอกจากนี้ยังได้เปรียบเทียบแบบจำลอง OS-YATSI กับแบบจำลอง YATSI แบบดั้งเดิมอีกด้วย ตารางที่ 34 – 36 แสดงการเปรียบเทียบมาตรวัดประสิทธิภาพ Pr, Re, และ F1 ทั้งแบบ micro-average และ macro-average ซึ่งพบว่าแบบจำลอง OS-YATSI มีประสิทธิภาพดีกว่าแบบจำลองด้วยวิธีแบบดั้งเดิมและแบบจำลอง YATSI แบบดั้งเดิมในทุกมาตรวัดประสิทธิภาพ เพื่อให้เห็นผลลัพธ์ได้อย่างชัดเจน จึงสรุปเป็นแผนภาพแสดงจำนวนซอฟต์แวร์ที่มีประสิทธิภาพดีกว่าอย่างมีนัยสำคัญของวิธีการ OS-YATSI ในแต่ละมาตรวัดได้ดังรูปที่ 27



(ก) มาตรวัดประสิทธิภาพการทำนายในรูปแบบ macro-average



(ข) มาตรวัดประสิทธิภาพการทำนายในรูปแบบ micro-average

รูปที่ 27 แผนภาพแสดงจำนวนซอฟต์แวร์ที่ให้ประสิทธิภาพสูงสุดอย่างมีนัยสำคัญของวิธี OS-YATSI ในทุกมาตรวัดประสิทธิภาพการทำนายในรูปแบบ (ก) macro-average และ (ข) micro-average

จากรูปที่ 27 สามารถวิเคราะห์และสรุปผลได้ดังนี้

(1) เมื่อพิจารณามาตรวัดแบบ macro-average จะพบว่า แบบจำลอง OS-YATSI ให้ประสิทธิภาพดีกว่าอย่างมีนัยสำคัญ โดยมีจำนวนซอฟต์แวร์ที่มีประสิทธิภาพดีกว่าอย่างมีนัยสำคัญในแต่ละมาตรวัด คือ มาตรวัด Precision จำนวน 3 ซอฟต์แวร์ มาตรวัด Recall จำนวน 1 ซอฟต์แวร์ และมาตรวัด F1-measure จำนวน 2 ซอฟต์แวร์ นอกจากนี้แบบจำลอง OS-YATSI สามารถปรับปรุงประสิทธิภาพการทำนายจากวิธีการแบบดั้งเดิมเพิ่มขึ้นเฉลี่ย 28.79% โดยเฉพาะอย่างยิ่งในซอฟต์แวร์ PDE UI สามารถปรับปรุงประสิทธิภาพการทำนายเพิ่มขึ้นถึง 49.31%

(2) เมื่อพิจารณามาตรวัดแบบ micro-average จะพบว่า แบบจำลอง OS-YATSI ให้ประสิทธิภาพดีกว่าอย่างมีนัยสำคัญ โดยมีจำนวนซอฟต์แวร์ที่มีประสิทธิภาพดีกว่าอย่างมีนัยสำคัญในแต่ละมาตรวัด คือ มาตรวัด Precision จำนวน 2 ซอฟต์แวร์ มาตรวัด Recall จำนวน 2 ซอฟต์แวร์ และมาตรวัด F1-measure จำนวน 2 ซอฟต์แวร์

ดังนั้น การทดลองนี้แสดงให้เห็นว่าวิธีการ OS-YATSI มีประสิทธิภาพและเหมาะสมที่จะนำไปประยุกต์ใช้ในกระบวนการทำนายระดับความรุนแรงของข้อบกพร่องในซอฟต์แวร์ได้

บทที่ 5
การออกแบบและพัฒนาเครื่องมือต้นแบบ
สำหรับการตรวจจับและจัดลำดับข้อบกพร่องของซอฟต์แวร์ในชุดข้อมูลไม่สมดุล

ในบทนี้จะอธิบายถึงวิธีการพัฒนาเครื่องมือต้นแบบ ประกอบด้วย การอธิบายความต้องการที่เป็นหน้าที่หลัก การออกแบบการทำงาน ส่วนต่อประสานผู้ใช้ ตลอดจนการทดสอบเครื่องมือต้นแบบที่พัฒนา โดยจะออกแบบด้วยภาษายูเอ็มแอล เพื่อใช้ในการอธิบายแบบจำลองต่างๆ ในการนำเสนอภาพรวมของระบบ ซึ่งประกอบด้วย แผนภาพยูสเคส แผนภาพคลาส และแผนภาพกิจกรรม ซึ่งมีเนื้อหาดังนี้

5.1 ความต้องการเชิงหน้าที่

การพัฒนาเครื่องมือต้นแบบที่สนับสนุนกรอบงานสำหรับตรวจจับและจัดลำดับข้อบกพร่องของซอฟต์แวร์ในชุดข้อมูลไม่สมดุล ระบบจะต้องมีความสามารถในการทำงานที่เป็นหน้าที่หลัก ซึ่งประกอบด้วยการทำงาน 4 ส่วน ดังนี้

5.1.1 การนำเข้าไฟล์ข้อมูล

เครื่องมือต้นแบบที่พัฒนาในงานวิจัยนี้ จะมีการนำเข้าไฟล์ข้อมูลซึ่งประกอบด้วยไฟล์ 2 ประเภท คือ ไฟล์โครงการซอฟต์แวร์ที่ใช้สำหรับการทำนายข้อบกพร่อง และไฟล์ข้อมูลป้อนกลับจากผู้ใช้ที่ใช้สำหรับการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรง โดยชนิดของไฟล์ทั้งสองจะอยู่ในรูปแบบของตาราง (.csv)

5.1.2 การทำนายข้อบกพร่อง

เครื่องมือต้นแบบจะทำการอ่านไฟล์โครงการซอฟต์แวร์และเก็บข้อมูลของทุกๆคลาสจากไฟล์โครงการซอฟต์แวร์ โดยข้อมูลดังกล่าวก็คือ ค่ามาตรวัดซอฟต์แวร์แต่ละตัว เมื่อทำการเก็บค่าต่างๆ แล้วก็จะทำการสร้างแบบจำลองด้วยวิธีการที่ให้ประสิทธิภาพดีที่สุด เพื่อใช้ในการทำนายคลาสที่เกิดข้อบกพร่องในซอฟต์แวร์ จากนั้นเครื่องมือจะตรวจจับคลาสที่เกิดข้อบกพร่องโดยวิเคราะห์จากมาตรวัดคุณลักษณะภายในของซอฟต์แวร์ แล้วเก็บผลลัพธ์ไว้แสดงผลในขั้นตอนต่อไป

5.1.3 การจัดลำดับข้อบกพร่องตามระดับความรุนแรง

การจัดลำดับข้อบกพร่องตามระดับความรุนแรงของเครื่องมือนี้จะแบ่งออกเป็น 2 แบบจำลอง คือ แบบจำลองกระบวนการลำดับชั้นเชิงวิเคราะห์ และแบบจำลองวิธีการเรียนรู้แบบกึ่งมีผู้สอน โดยเครื่องมือต้นแบบจะนำผลลัพธ์จากการทำนายคลาสที่เกิดข้อบกพร่องมาคำนวณหาระดับ

ความรุนแรงของข้อบกพร่องในคลาสนั้นๆ ตามข้อกำหนดของแต่ละแบบจำลอง จากนั้นจึงจัดลำดับของคลาสตามระดับความรุนแรงและโอกาสที่จะเกิดข้อบกพร่อง แล้วเก็บผลลัพธ์ไว้เพื่อการแสดงผลในขั้นตอนต่อไป

5.1.4 การแสดงผล

ระบบสามารถแสดงรายงานผลลัพธ์ที่ได้จากการทำนายและจัดลำดับข้อบกพร่องตามระดับความรุนแรงได้ โดยการแสดงผลของเครื่องมือต้นแบบนี้จะแสดงผลในรูปแบบของตาราง ซึ่งประกอบด้วยคอลัมน์ต่างๆ ที่แสดงข้อมูลของคลาสที่เกิดข้อบกพร่อง คือ ลำดับการแก้ไขคลาสที่เกิดข้อบกพร่อง ชื่อคลาส ระดับความรุนแรง ค่าความมั่นใจ และมาตรวัดซอฟต์แวร์ นอกจากนี้ผู้ใช้ยังสามารถบันทึกไฟล์เพื่อนำรายงานผลลัพธ์ที่ได้ไปใช้งานในภายหลังได้

5.2 การออกแบบเครื่องมือ

เครื่องมือต้นแบบนี้พัฒนาขึ้นเพื่อช่วยอำนวยความสะดวกในการตรวจจับและจัดลำดับของข้อบกพร่องตามระดับความรุนแรง เพื่อเป็นแนวทางให้กับนักพัฒนาสำหรับนำไปใช้ตัดสินใจในการแก้ไขปรับปรุงซอฟต์แวร์ให้มีคุณภาพ ซึ่งสามารถอธิบายรายละเอียดส่วนต่างๆรวมถึงกระบวนการทำงานของเครื่องมือต้นแบบได้ดังนี้

5.2.1 แผนภาพยูสเคส

จากการวิเคราะห์การเกิดปฏิสัมพันธ์ระหว่างผู้ใช้งานกับระบบ สามารถเขียนแผนภาพยูสเคสอธิบายลำดับเหตุการณ์ที่เกิดขึ้น เพื่อแสดงให้เห็นถึงบทบาทการใช้งานระบบของผู้ใช้ ดังแสดงในรูปที่ 28 ซึ่งมีรายละเอียดดังนี้

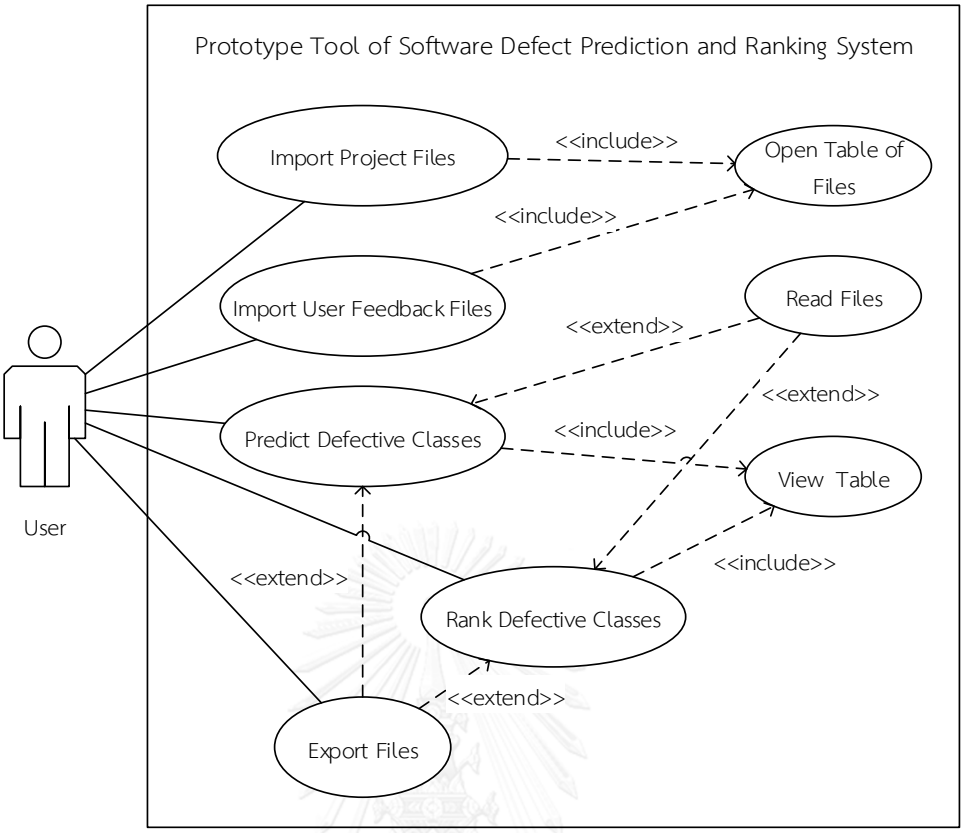
1) ผู้ใช้งานสามารถกำหนดไฟล์โครงการซอฟต์แวร์ที่ต้องการนำมาทำนายข้อบกพร่องของซอฟต์แวร์ได้

2) ผู้ใช้งานสามารถกำหนดไฟล์ข้อมูลป้อนกลับจากผู้ใช้ที่ต้องการนำมาจัดลำดับของข้อบกพร่องตามระดับความรุนแรงได้

3) ผู้ใช้งานสามารถทำนายข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์ได้

4) ผู้ใช้งานสามารถเลือกวิธีการจัดลำดับข้อบกพร่องตามระดับความรุนแรงได้ โดยมีให้เลือกทั้งหมด 2 วิธี คือ วิธีกระบวนการลำดับชั้นเชิงวิเคราะห์ และวิธีการเรียนรู้แบบกึ่งมีผู้สอน

5) ผู้ใช้งานสามารถบันทึกไฟล์รายงานผลลัพธ์จากการทำนายข้อบกพร่องและการจัดลำดับข้อบกพร่องตามระดับความรุนแรงเพื่อนำไปใช้งานภายหลังได้



รูปที่ 28 แผนภาพยูสเคสของเครื่องมือต้นแบบสำหรับตรวจจับและจัดลำดับข้อบกพร่อง

จากรูปที่ 28 สามารถเขียนอธิบายความสัมพันธ์ของยูสเคสที่เกิดขึ้นภายในระบบได้ดังนี้

1) ยูสเคส Import Project Files

จากตารางที่ 37 เป็นยูสเคสที่แสดงลำดับเหตุการณ์เมื่อผู้ใช้นำเข้าไฟล์โครงการซอฟต์แวร์เข้าสู่ระบบ เพื่อนำไปใช้ในการทำนายข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์

ตารางที่ 37 ขั้นตอนการทำงานของยูสเคส Import Project Files

Use case name	Import Project Files
Entry condition	1. มีไฟล์โครงการซอฟต์แวร์ เพื่อใช้ในการทำนายข้อบกพร่องของซอฟต์แวร์ 2. มีค่ามาตรวัดซอฟต์แวร์ที่มีรูปแบบถูกต้องตามข้อกำหนด
Flow of events	3. ผู้ใช้งานเลือกนำเข้าไฟล์โครงการซอฟต์แวร์ในรูปแบบตาราง (.csv) 4. ระบบแสดงรายละเอียดข้อมูลไฟล์โครงการซอฟต์แวร์ในรูปแบบตาราง
Exit condition	5. ผู้ใช้งานเลือกทำขั้นตอนต่อไป

2) ยูสเคส Import User Feedback Files

จากตารางที่ 38 เป็นยูสเคสที่แสดงลำดับเหตุการณ์เมื่อผู้ใช้นำเข้าไฟล์ข้อมูลป้อนกลับจากผู้ใช้เข้าสู่ระบบ เพื่อนำไปใช้ในการจัดลำดับข้อบกพร่องที่เกิดขึ้นตามระดับความรุนแรง

ตารางที่ 38 ขั้นตอนการทำงานของยูสเคส Import User Feedback Files

Use case name	Import User Feedback Files
Entry condition	1. มีไฟล์ข้อมูลป้อนกลับจากผู้ใช้ (User Feedback) เพื่อใช้ในการจัดลำดับข้อบกพร่องตามระดับความรุนแรง 2. มีค่ามาตรวัดซอฟต์แวร์และระดับความรุนแรงที่มีรูปแบบถูกต้องตามข้อกำหนด
Flow of events	3. ผู้ใช้งานเลือกนำเข้าไฟล์ข้อมูลป้อนกลับจากผู้ใช้ในรูปแบบตาราง (.csv) 4. ระบบแสดงรายละเอียดข้อมูลไฟล์ข้อมูลป้อนกลับจากผู้ใช้ในรูปแบบตาราง
Exit condition	5. ผู้ใช้งานเลือกทำขั้นตอนต่อไป

3) ยูสเคส Predict Defective Classes

จากตารางที่ 39 เป็นยูสเคสที่แสดงลำดับเหตุการณ์การทำนายข้อบกพร่องของระบบ เมื่อผู้ใช้นำเข้าไฟล์โครงการซอฟต์แวร์ ระบบจะทำการอ่านไฟล์และค่าของมาตรวัดต่างๆ แล้วทำนายหาคลาสที่เกิดข้อบกพร่อง จากนั้นจึงแสดงรายงานผลลัพธ์การทำนายข้อบกพร่องในรูปแบบตาราง

ตารางที่ 39 ขั้นตอนการทำงานของยูสเคส Predict Defective Classes

Use case name	Predict Defective Classes
Entry condition	1. มีชุดข้อมูลที่มีค่ามาตรวัดซอฟต์แวร์ที่ได้นำเข้าอย่างถูกต้อง และระบบสามารถเปิดอ่านไฟล์ได้
Flow of events	2. ผู้ใช้งานเลือกทำขั้นตอนต่อไป 3. ระบบจะแสดงสถานะการทำนายข้อบกพร่อง 4. ระบบแสดงรายงานผลลัพธ์จากการทำนายข้อบกพร่องในรูปแบบของตาราง
Exit condition	5. ผู้ใช้งานเลือกทำขั้นตอนต่อไป

4) ยูสเคส Rank Defective Classes

จากตารางที่ 40 เป็นยูสเคสที่แสดงลำดับเหตุการณ์ในการเลือกแบบจำลองการจัดลำดับข้อบกพร่องของซอฟต์แวร์ตามระดับความรุนแรง ระบบจะทำการอ่านไฟล์และค่าของมาตรวัดต่างๆ แล้วจัดลำดับข้อบกพร่อง จากนั้นจึงแสดงรายงานผลลัพธ์การจัดลำดับข้อบกพร่องตามระดับความรุนแรงในรูปแบบตาราง

ตารางที่ 40 ขั้นตอนการทำงานของยูสเคส Rank Defective Classes

Use case name	Rank Defective Classes
Entry condition	1. มีรายงานผลลัพธ์จากการทำนายข้อบกพร่อง 2. มีไฟล์ป้อนกลับจากผู้ใช้ที่มีค่ามาตรวัดและระดับความรุนแรงที่ได้นำเข้าอย่างถูกต้อง และระบบสามารถเปิดอ่านไฟล์ได้
Flow of events	3. ผู้ใช้งานเลือกแบบจำลองที่ใช้ในการจัดลำดับข้อบกพร่องของซอฟต์แวร์ตามระดับความรุนแรง 4. ระบบจะแสดงสถานะการจัดลำดับข้อบกพร่องตามระดับความรุนแรง 5. ระบบแสดงรายงานผลลัพธ์จากการจัดลำดับข้อบกพร่องของซอฟต์แวร์ตามระดับความรุนแรงในรูปแบบของตาราง
Exit condition	6. ผู้ใช้งานเลือกใช้งานเมนูอื่น

5) ยูสเคส Export Files

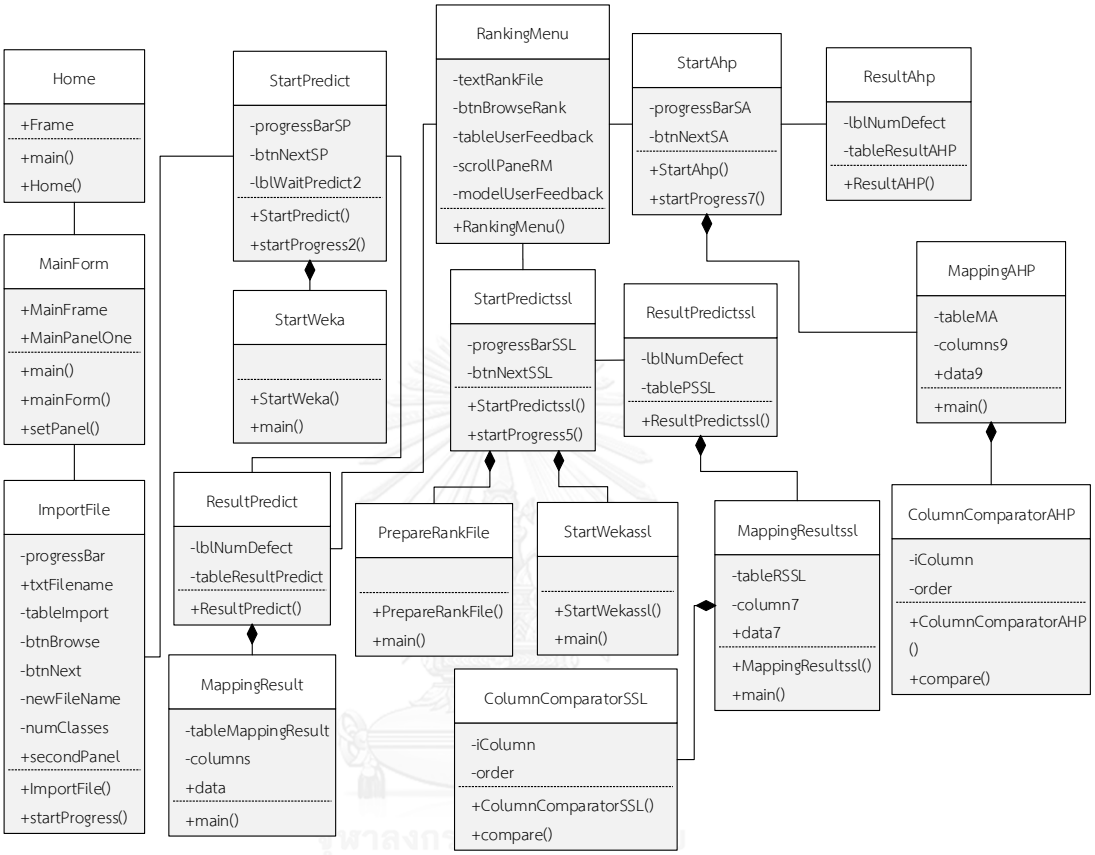
จากตารางที่ 41 เป็นยูสเคสแสดงลำดับเหตุการณ์ที่เกิดขึ้น เมื่อผู้ใช้งานเลือกเมนู Export จากระบบ เพื่อทำการบันทึกไฟล์งานของการทำนายข้อบกพร่อง หรือ วิธีการจัดลำดับข้อบกพร่องตามระดับความรุนแรง ในรูปแบบของตาราง

ตารางที่ 41 ขั้นตอนการทำงานของยูสเคส Export Files

Use case name	Export Files
Entry condition	1. มีรายงานผลลัพธ์ของการทำนายข้อบกพร่อง หรือ รายงานผลลัพธ์การจัดลำดับข้อบกพร่องตามระดับความรุนแรง 2. ผู้ใช้งานเลือกเมนู Export
Flow of events	3. ผู้ใช้งานเลือกแหล่งที่เก็บของไฟล์รายงานผลลัพธ์ 4. ผู้ใช้งานตั้งชื่อไฟล์งาน หรือใช้ชื่อไฟล์งานเดิมตามที่ปรากฏ
Exit condition	5. ผู้ใช้งานเลือกบันทึกไฟล์ หรือ ยกเลิกการบันทึกไฟล์

5.2.2 แผนภาพคลาส

แผนภาพคลาสแสดงความสัมพันธ์ขององค์ประกอบของระบบภายในเครื่องมือดังรูปที่ 29 ซึ่งสามารถอธิบายได้ดังนี้



รูปที่ 29 แผนภาพคลาสแสดงองค์ประกอบของเครื่องมือ

1) แผนภาพคลาส Home

คลาส Home เป็นคลาสแสดงหน้าจอต้อนรับของเครื่องมือ SDPRS ซึ่งย่อมาจาก Software Defect Prediction and Ranking System ประกอบด้วยเมนู Start และรายละเอียดเกี่ยวกับผู้พัฒนาเครื่องมือต้นแบบ โดยมีการเรียกใช้งานคลาส MainForm เพื่อเข้าสู่หน้าจอหลักในการทำงาน

2) แผนภาพคลาส MainForm

คลาส MainForm เป็นคลาสแสดงหน้าจอหลักในการทำงานของเครื่องมือ โดยทำหน้าที่เรียกหน้าจอของคลาสอื่นๆ ขึ้นมา เพื่อนำมาแสดงผล โดยจะมีการเรียกใช้งานคลาส ImportFile เพื่อนำเข้าข้อมูลสำหรับการทำนายข้อบกพร่อง

3) แผนภาพคลาส ImportFile

คลาส ImportFile เป็นคลาสที่แสดงหน้าจอสำหรับนำข้อมูลเข้า และแสดงรายละเอียดของข้อมูลในรูปแบบของตาราง โดยจะรับข้อมูลในรูปแบบของตาราง (.csv) โดยจะมีการเรียกใช้งานคลาส StartPredict เพื่อใช้ในการทำนายข้อบกพร่อง

4) แผนภาพคลาส StartPredict

คลาส StartPredict เป็นคลาสที่แสดงหน้าจอสำหรับประมวลผลการทำนายข้อบกพร่อง ซึ่งจะแสดงแถบสถานะการทำนายข้อบกพร่อง โดยจะมีการเรียกใช้งานคลาส StartWeka ซึ่งทำหน้าที่ในการเรียกใช้ API ของ Weka ในการทำนายข้อบกพร่อง และเรียกใช้งานคลาส ResultPredict เพื่อแสดงผลการทำนายข้อบกพร่อง

5) แผนภาพคลาส StartWeka

คลาส StartWeka เป็นส่วนหนึ่งของคลาส StartPredict ซึ่งทำหน้าที่ในการเรียกใช้งาน API ของ Weka โดยจะสร้างแบบจำลองการทำนายข้อบกพร่องและสามารถปรับค่าพารามิเตอร์ต่างๆ ของแบบจำลองได้

6) แผนภาพคลาส ResultPredict

คลาส ResultPredict เป็นคลาสที่แสดงหน้าจอสำหรับแสดงผลการทำนายข้อบกพร่องในรูปแบบของตาราง โดยจะแสดงเฉพาะคลาสที่เกิดข้อบกพร่องเท่านั้น ซึ่งจะมีการเรียกใช้งานคลาส RankingMenu และคลาส MappingResult ซึ่งเป็นส่วนหนึ่งของคลาสนี้ด้วย และสามารถ export รายงานผลลัพธ์เพื่อนำไปใช้งานภายหลังได้

7) แผนภาพคลาส MappingResult

คลาส MappingResult เป็นส่วนหนึ่งของคลาส ResultPredict ซึ่งทำหน้าที่ในการเชื่อมโยงชื่อคลาสกับข้อบกพร่องที่ทำนายได้

8) แผนภาพคลาส RankingMenu

คลาส RankingMenu เป็นคลาสที่แสดงหน้าจอสำหรับเลือกเมนูการจัดลำดับความสำคัญของข้อบกพร่องตามระดับความรุนแรง โดยจะมีการเรียกใช้งานคลาส StartPredictssl และคลาส StartAhp

9) แผนภาพคลาส StartPredictssl

คลาส StartPredictssl เป็นคลาสที่แสดงหน้าจอสำหรับประมวลผลการจัดลำดับข้อบกพร่องตามระดับความรุนแรง ซึ่งจะแสดงแถบสถานะจัดลำดับข้อบกพร่อง โดยจะมีการเรียกใช้งานคลาส StartWekassl และเรียกใช้งานคลาส PrepareRankFile ซึ่งเป็นส่วนหนึ่งของคลาสนี้ด้วย นอกจากนี้ยังมีการเรียกใช้คลาส ResultPredictssl เพื่อแสดงผลการจัดลำดับข้อบกพร่องตามระดับความรุนแรง

10) แผนภาพคลาส PrepareRankFile

คลาส PrepareRankFile เป็นคลาสที่ทำหน้าที่ในการเชื่อมโยงรายงานผลลัพธ์จากการทำนายข้อบกพร่อง โดยทำการจัดรูปแบบของรายงานผลลัพธ์ให้อยู่ในรูปแบบตาราง (.csv) เพื่อนำไปใช้ในการจัดลำดับข้อบกพร่องตามระดับความรุนแรงด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน

11) แผนภาพคลาส StartWekassl

คลาส StartWekassl เป็นคลาสที่ทำหน้าที่ในการเรียกใช้ API ของ Weka ในการจัดลำดับข้อบกพร่องด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน โดยจะสร้างแบบจำลองการจัดลำดับข้อบกพร่องและสามารถปรับค่าพารามิเตอร์ต่างๆ ของแบบจำลองได้

12) แผนภาพคลาส ResultPredictssl

คลาส ResultPredictssl เป็นคลาสที่แสดงหน้าจอสำหรับแสดงผลการจัดลำดับข้อบกพร่องในรูปแบบของตาราง โดยจะแสดงลำดับการแก้ไขของคลาสที่เกิดข้อบกพร่องตามระดับความรุนแรง ซึ่งจะมีการเรียกใช้งานคลาส MappingResultssl ซึ่งเป็นส่วนหนึ่งของคลาสนี้ด้วย และสามารถ export รายงานผลลัพธ์เพื่อนำไปใช้งานภายหลังได้

13) แผนภาพคลาส MappingResultssl

คลาส MappingResultssl เป็นส่วนหนึ่งของคลาส ResultPredictssl ซึ่งทำหน้าที่ในการเชื่อมโยงลำดับการแก้ไข ชื่อคลาส และระดับความรุนแรงของข้อบกพร่องที่ทำนายได้ โดยจะมีการเรียกใช้งานคลาส ColumnComparatorSSL เพื่อเรียงลำดับของข้อบกพร่องตามระดับความรุนแรง

14) แผนภาพคลาส ColumnComparatorSSL

คลาส ColumnComparatorSSL เป็นส่วนหนึ่งของคลาส คลาส MappingResultssl ซึ่งทำหน้าที่ในการเปรียบเทียบระดับความรุนแรงของข้อบกพร่อง โดยจะเรียงลำดับข้อบกพร่องจากความรุนแรงมากไปน้อยตามลำดับ

15) แผนภาพคลาส StartAhp

คลาส StartAhp เป็นคลาสที่ทำหน้าที่ในการคำนวนหาค่า AHP ตามเกณฑ์การตัดสินใจที่งานวิจัยนี้ได้เสนอ เพื่อใช้ในการจัดลำดับข้อบกพร่องตามระดับความรุนแรง โดยจะสร้างแบบจำลองการจัดลำดับข้อบกพร่องและสามารถปรับค่าพารามิเตอร์ต่างๆ ของแบบจำลองได้ และมีการเรียกใช้งานคลาส ResultAhp และคลาส MappingAHP ด้วย

16) แผนภาพคลาส ResultAhp

คลาส ResultAhp เป็นคลาสที่แสดงหน้าจอสำหรับแสดงผลการจัดลำดับข้อบกพร่องในรูปแบบของตาราง โดยจะแสดงลำดับการแก้ไขของคลาสที่เกิดข้อบกพร่องตามระดับความรุนแรง และสามารถ export รายงานผลลัพธ์เพื่อนำไปใช้งานภายหลังได้

17) แผนภาพคลาส MappingAHP

คลาส MappingAHP เป็นส่วนหนึ่งของคลาส StartAhp ซึ่งทำหน้าที่ในการเชื่อมโยงลำดับการแก้ไข ชื่อคลาส และระดับความรุนแรงของข้อบกพร่องที่ทำนายได้ โดยจะมีการเรียกใช้งานคลาส ColumnComparatorAHP เพื่อเรียงลำดับของข้อบกพร่องตามระดับความรุนแรง

18) แผนภาพคลาส ColumnComparatorAHP

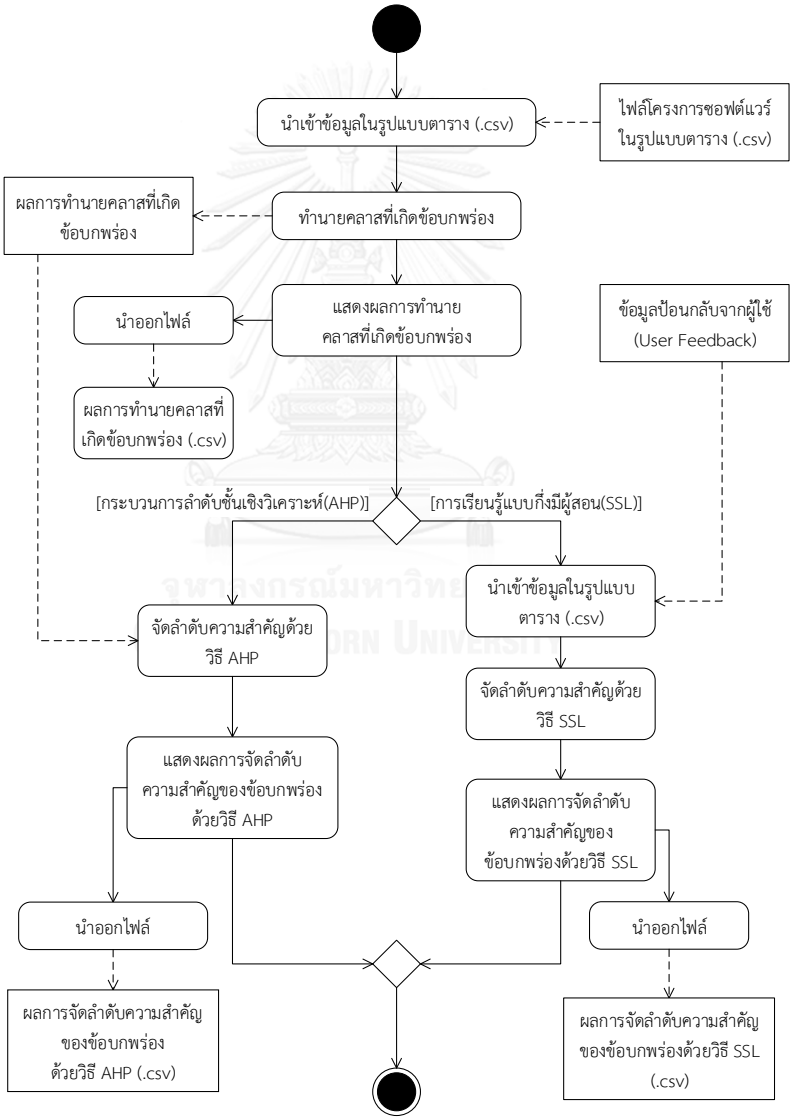
คลาส ColumnComparatorAHP เป็นส่วนหนึ่งของคลาส คลาส MappingAHP ซึ่งทำหน้าที่ในการเปรียบเทียบระดับความรุนแรงของข้อบกพร่อง โดยจะเรียงลำดับข้อบกพร่องจากความรุนแรงมากไปน้อยตามลำดับ

5.2.3 แผนภาพกิจกรรม

แผนภาพกิจกรรมแสดงลำดับการเกิดกิจกรรมการทำงานต่างๆ ภายในระบบของเครื่องมือต้นแบบดังรูปที่ 30 เครื่องมือต้นแบบนี้ใช้สำหรับการตรวจจับข้อคลาสที่เกิดข้อบกพร่องและจัดลำดับข้อบกพร่องตามระดับความรุนแรง

กิจกรรมเริ่มต้นจากผู้ใช้นำข้อมูลเข้าไฟล์โครงการซอฟต์แวร์ โดยไฟล์ที่นำเข้าต้องอยู่ในรูปแบบตาราง (.csv) เมื่อนำเข้าข้อมูลแล้วระบบจะทำการตรวจสอบความถูกต้อง และความครบถ้วนของข้อมูลนำเข้า จากนั้นระบบจะทำนายคลาสที่เกิดข้อบกพร่องด้วยแบบจำลองวิธีการจำแนกประเภท ผลลัพธ์จากขั้นตอนนี้ จะได้รายงานคลาสที่เกิดข้อบกพร่อง ซึ่งผู้ใช้งานสามารถนำออกไฟล์หรือ บันทึกไฟล์รายงานคลาสที่เกิดข้อบกพร่องนี้ได้ เพื่อนำไปใช้งานในภายหลัง นอกจากนี้รายงานคลาสที่เกิดข้อบกพร่องนี้ยังนำไปใช้ในการจัดลำดับข้อบกพร่องตามระดับความรุนแรง โดยระบบสามารถจัดลำดับข้อบกพร่องตามระดับความรุนแรงโดยใช้วิธีกระบวนการลำดับชั้นเชิงวิเคราะห์ และ

วิธีการเรียนรู้แบบกึ่งมีผู้สอน ซึ่งวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์จะจัดลำดับข้อบกพร่องโดยพิจารณาจากเกณฑ์การตัดสินใจค่าความมั่นใจและค่าผลกระทบการขึ้นต่อกันของคลาส ส่วนวิธีการเรียนรู้แบบกึ่งมีผู้สอน ผู้ใช้งานจะต้องนำเข้าข้อมูลป้อนกลับจากผู้ใช้ซึ่งเป็นข้อมูลของคลาสที่เกิดข้อบกพร่องบางส่วนของโครงการซอฟต์แวร์ที่มีการระบุถึงระดับความรุนแรง มาใช้ในการเรียนรู้เพื่อให้ทราบระดับความรุนแรงของข้อมูลที่เหลือและจัดลำดับคลาสที่เกิดข้อบกพร่องเหล่านี้ตามระดับความรุนแรง จากนั้นระบบจะแสดงรายงานผลการจัดลำดับข้อบกพร่องตามระดับความรุนแรง ซึ่งผู้ใช้สามารถเลือกนำออกไฟล์ หรือ บันทึกไฟล์ ได้เช่นเดียวกับรายงานข้อบกพร่อง แต่หากผู้ใช้เลือกปิดหน้าจอการทำงาน ระบบก็จะปิดและหยุดการทำงานทั้งหมด



รูปที่ 30 แผนภาพกิจกรรมของเครื่องมือต้นแบบสำหรับตรวจจับและจัดลำดับข้อบกพร่องของซอฟต์แวร์ในชุดข้อมูลไม่สมดุล

5.2.4 เครื่องมือสนับสนุนในการพัฒนา

ผู้วิจัยได้ใช้ฮาร์ดแวร์ (Hardware) และซอฟต์แวร์ (Software) ที่สนับสนุนการพัฒนาเครื่องมือต้นแบบของงานวิจัย ซึ่งมีคุณสมบัติดังนี้

5.2.4.1 ด้านฮาร์ดแวร์

1) เครื่องคอมพิวเตอร์ส่วนบุคคล หน่วยประมวลผล Intel Core i5 ความเร็ว 1.80 กิกะเฮิรตซ์ (GHz)
2) จานบันทึกแบบแข็ง (Hard Disk) 239 กิกะไบต์ (GB)
3) หน่วยความจำ 4 กิกะไบต์ (GB)

5.2.4.2 ด้านซอฟต์แวร์

1) ระบบปฏิบัติการไมโครซอฟท์วินโดวส์รุ่นที่ 10 (Microsoft Windows 10)
2) อิคลิปส์รุ่นที่ 4.4 (Eclipse 4.4 Luna) เป็นเครื่องมือสำหรับการพัฒนาโปรแกรมภาษาจาวา (Java)
3) เครื่องมือคำนวณหาค่ามาตรวัดจากรหัสต้นฉบับ Ckjm [21]
4) เครื่องมือ Weka เวอร์ชัน 3.7 [35] เป็นเครื่องมือสำหรับงานด้านการทำเหมืองข้อมูล (Data Mining)
5) เครื่องมือ HR-SVM เวอร์ชัน 1.1 [34] เป็นเครื่องมือสำหรับงานด้านการทำเหมืองข้อมูลที่สามารถแก้ไขปัญหาความไม่สมดุล (Imbalanced Issue) ของคลาสได้

บทที่ 6
สรุปผลการวิจัย

6.1 สรุปผลการวิจัย

การตรวจจับและจัดลำดับข้อบกพร่องในซอฟต์แวร์ตามระดับความรุนแรง ถือเป็นกระบวนการที่สำคัญในงานทางด้านวิศวกรรมซอฟต์แวร์ ที่สามารถวิเคราะห์โอกาสที่จะเกิดข้อบกพร่องขึ้นภายในซอฟต์แวร์ และประเมินระดับความรุนแรงของข้อบกพร่องที่เกิดขึ้นแต่ละตัว เพื่อนำไปใช้ในการจัดลำดับความสำคัญในการแก้ไขข้อบกพร่องได้อย่างถูกต้องและมีประสิทธิภาพ แต่กระบวนการนี้ก็มักจะเกิดข้อผิดพลาด เนื่องจากประสบกับปัญหาความไม่สมดุลของข้อมูล ที่มีอัตราส่วนของข้อมูลกลุ่มใดกลุ่มหนึ่งมากกว่าข้อมูลกลุ่มอื่นๆ ซึ่งเป็นสาเหตุให้ประสิทธิภาพการตรวจจับลดลง นอกจากนี้การจัดลำดับของคลาสที่เกิดข้อบกพร่องยังมีการพิจารณาจากระดับความรุนแรงเพียงอย่างเดียวเท่านั้น ซึ่งความเป็นจริงแล้วคลาสที่มีความรุนแรงสูงอาจมีโอกาสเกิดข้อบกพร่องน้อยมากก็เป็นได้ ซึ่งหากมีการพิจารณาค่าความมั่นใจ หรือโอกาสในการเกิดข้อบกพร่องร่วมกันกับระดับความรุนแรง จะช่วยให้การจัดลำดับในการแก้ไขข้อบกพร่องมีความแม่นยำมากยิ่งขึ้น

จากปัญหาที่ได้กล่าวไปข้างต้น งานวิจัยนี้จึงได้นำเสนอกรอบงานสำหรับตรวจจับและจัดลำดับข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์ โดยแบ่งการทำงานออกเป็น 2 ส่วน คือ การตรวจจับข้อบกพร่อง และการจัดลำดับข้อบกพร่องตามระดับความรุนแรง

ในส่วนแรก กรอบงานจะนำเสนอการตรวจจับข้อบกพร่องโดยการประยุกต์ใช้วิธีอันไบแอสซัพพอร์ตเวกเตอร์แมชชีน ซึ่งสามารถแก้ไขปัญหาความไม่สมดุลของคลาสโดยการใช้เทคนิคการปรับแก้ค่าขีดแบ่งเพื่อลดไบแอสของจำนวนคลาสที่มีเสียงข้างมาก ทำให้ประสิทธิภาพการทำนายมีความถูกต้องและแม่นยำเพิ่มมากขึ้น โดยจะทำการทดลองกับซอฟต์แวร์จำนวน 15 โปรแกรม และเปรียบเทียบประสิทธิภาพกับวิธีการจำแนกประเภทแบบดั้งเดิม จำนวน 4 วิธี ประกอบด้วย ต้นไม้ตัดสินใจ เคเนียร์เรสเนเบอร์ การเรียนรู้เบย์แบบง่าย และซัพพอร์ตเวกเตอร์แมชชีน จากผลการทดลองสามารถสรุปได้ว่า ในเทอมของมาตรวัด PD วิธีการอันไบแอสซัพพอร์ตเวกเตอร์แมชชีนให้ประสิทธิภาพดีกว่าวิธีการแบบดั้งเดิมจำนวน 10 โปรแกรมจากจำนวนทั้งหมด 15 โปรแกรม (ประสิทธิภาพดีกว่าวิธีการแบบดั้งเดิมจำนวน 5 โปรแกรม และดีกว่าวิธีการแบบดั้งเดิมอย่างมีนัยสำคัญจำนวน 5 โปรแกรม) ในเทอมของมาตรวัด F1 วิธีการอันไบแอสซัพพอร์ตเวกเตอร์แมชชีนให้ประสิทธิภาพดีกว่าวิธีการแบบดั้งเดิมจำนวน 8 โปรแกรมจากจำนวนทั้งหมด 15 โปรแกรม (ประสิทธิภาพดีกว่าวิธีการแบบดั้งเดิมจำนวน 2 โปรแกรม และดีกว่าอย่างมีนัยสำคัญจำนวน 6

โปรแกรม) และในเทอมของมาตรวัด G-mean วิธีการอันไบแอสซัพพอร์ตเวกเตอร์แมชชีนให้ประสิทธิภาพดีกว่าวิธีการแบบดั้งเดิมจำนวน 8 โปรแกรมจากจำนวนทั้งหมด 15 โปรแกรม (ประสิทธิภาพดีกว่าจำนวน 3 โปรแกรม และดีกว่าอย่างมีนัยสำคัญจำนวน 5 โปรแกรม)

ในส่วนที่สอง กรอบงานจะนำเสนอการจัดลำดับข้อบกพร่องตามระดับความรุนแรง โดยจะนำผลลัพธ์จากกรอบงานในส่วนแรกมาจัดลำดับตามระดับความรุนแรง เพื่อช่วยนักพัฒนาในการตัดสินใจแก้ไขข้อบกพร่องได้อย่างถูกต้องและมีประสิทธิภาพ ซึ่งประกอบด้วย 2 วิธี คือ กระบวนการลำดับชั้นเชิงวิเคราะห์ และวิธีการ OS-YATSI

ในกระบวนการลำดับชั้นเชิงวิเคราะห์จะจัดลำดับข้อบกพร่องที่เกิดขึ้น โดยพิจารณาเกณฑ์การตัดสินใจร่วมกันจำนวน 2 เกณฑ์ ระหว่างค่าความมั่นใจที่จะเกิดข้อบกพร่อง และค่าการขึ้นต่อกันของคลาส สาเหตุที่พิจารณาทั้งสองเกณฑ์การตัดสินใจเนื่องจาก หากพิจารณาเฉพาะค่าความมั่นใจที่จะเกิดข้อบกพร่อง พบว่าการจัดลำดับข้อบกพร่องที่เกิดขึ้นนั้นจะเรียงตามโอกาสที่คลาสนั้นๆจะเกิดข้อบกพร่องขึ้นเท่านั้น ไม่ได้บอกถึงผลกระทบของคลาส ซึ่งในความเป็นจริงคลาสที่มีโอกาสเกิดข้อบกพร่องสูง อาจส่งผลกระทบเพียงเล็กน้อยก็เป็นได้ ในทางตรงกันข้ามหากพิจารณาเฉพาะค่าการขึ้นต่อกันของคลาส พบว่าการจัดลำดับของข้อบกพร่องที่เกิดขึ้นนั้นจะเรียงตามผลกระทบหรือความรุนแรงของคลาสใดๆที่มีต่อโปรแกรม โดยคลาสที่มีค่า Dependency สูงๆอาจมีโอกาสเกิดข้อบกพร่องน้อยมากๆหรือไม่เกิดข้อบกพร่องเลย ซึ่งหากไปทำการแก้ไขหรือเปลี่ยนแปลงอาจเป็นการเพิ่มข้อบกพร่องให้กับโปรแกรม และส่งผลกระทบกับโปรแกรมมากยิ่งขึ้น จากเหตุผลดังกล่าวงานวิจัยนี้จึงเลือกพิจารณาเกณฑ์ตัดสินใจทั้งสองร่วมกัน เพราะสามารถบ่งชี้ถึงคลาสที่มีระดับความรุนแรงและโอกาสที่จะเกิดข้อบกพร่องขึ้นในคลาสนั้นได้ในเวลาเดียวกัน โดยทำการทดลองกับซอฟต์แวร์จำนวน 15 โปรแกรม และเปรียบเทียบประสิทธิภาพของแบบจำลองที่มีเกณฑ์การตัดสินใจแตกต่างกัน คือ R1:พิจารณาค่าความมั่นใจที่จะเกิดข้อบกพร่องเพียงอย่างเดียว R2:พิจารณาค่าการขึ้นต่อกันของคลาสเพียงอย่างเดียว และ R3:พิจารณาทั้งค่าความมั่นใจที่จะเกิดข้อบกพร่องและค่าการขึ้นต่อกันของคลาสร่วมกัน ซึ่งจากผลการทดลองสามารถสรุปได้ว่า วิธีกระบวนการลำดับชั้นเชิงวิเคราะห์แบบที่มีเกณฑ์การตัดสินใจแบบ R3 คือ พิจารณาทั้งค่าความมั่นใจที่จะเกิดข้อบกพร่องและค่าการขึ้นต่อกันของคลาสร่วมกัน ให้ประสิทธิภาพดีที่สุดจำนวน 10 โปรแกรมจากจำนวนทั้งหมด 15 โปรแกรม ในขณะที่วิธีกระบวนการลำดับชั้นเชิงวิเคราะห์แบบที่มีเกณฑ์การตัดสินใจแบบ R1 และ R2 ให้ประสิทธิภาพดีที่สุดจำนวน 3 และ 2 โปรแกรมตามลำดับ

ส่วนการจัดลำดับข้อบกพร่องด้วยวิธีการ OS-YATSI จะมีการประยุกต์ใช้วิธีการเรียนรู้แบบกึ่งมีผู้สอนร่วมกับการสุ่มเพิ่มตัวอย่างกลุ่มน้อย โดยอนุญาตให้นักพัฒนาหรือผู้ใช้กรอกระดับความรุนแรงของข้อบกพร่องเพียงบางส่วน และนำข้อบกพร่องเหล่านี้ไปสร้างแบบจำลองเพื่อช่วยระบุระดับความ

รุนแรงของข้อบกพร่องส่วนที่เหลือ ซึ่งกระบวนการทำงานจะประกอบด้วย 3 ขั้นตอน คือ การสุ่มตัวอย่างกลุ่มน้อย การเรียนรู้แบบกึ่งมีผู้สอน และเกณฑ์การคัดเลือกข้อมูลที่ไม่มีฉลากประเภท ในขั้นแรก จะทำให้ข้อมูลมีความสมดุลโดยการสุ่มเพิ่มตัวอย่างของระดับความรุนแรงให้แต่ละระดับมีจำนวนตัวอย่างเท่ากัน เพื่อใช้ในการสร้างแบบจำลองโดยที่ไม่ไบแอสกับระดับความรุนแรงที่มีเสียงข้างมาก จากนั้นข้อบกพร่องส่วนที่เหลือจะถูกระบุระดับความรุนแรงโดยใช้อัลกอริทึม YATSI สุดท้ายข้อบกพร่องที่ถูกระบุระดับความรุนแรงเหล่านี้จะถูกเลือกตามค่ามั่นใจ แล้วนำไปรวมกับข้อมูลในขั้นแรกที่ผ่านการสุ่มตัวอย่างแล้ว เพื่อนำไปใช้สร้างแบบจำลองสุดท้ายในการหาระดับความรุนแรงที่แท้จริงของแต่ละข้อบกพร่อง โดยจะทำการทดลองกับซอฟต์แวร์จำนวน 3 โปรแกรม ที่มีการนำเข้าผลป้อนกลับจากผู้ใช้ (ผลเฉลยของข้อบกพร่องบางส่วน) และเปรียบเทียบประสิทธิภาพกับวิธีจำแนกประเภทแบบดั้งเดิม ซึ่งจากผลการทดลองสามารถสรุปได้ว่า

แบบ macro-average ในเทอมของมาตรวัด Pr วิธีการ OS-YATSI ให้ประสิทธิภาพดีกว่าวิธีการแบบดั้งเดิมอย่างมีนัยสำคัญในทุกโปรแกรม(3 โปรแกรม) ในเทอมของมาตรวัด Re วิธีการ OS-YATSI ให้ประสิทธิภาพดีกว่าวิธีการแบบดั้งเดิมจำนวน 2 โปรแกรมจากจำนวนทั้งหมด 3 โปรแกรม และดีกว่าวิธีการแบบดั้งเดิมอย่างมีนัยสำคัญจำนวน 1 โปรแกรมจากจำนวนทั้งหมด 3 โปรแกรม และในเทอมของมาตรวัด Re วิธีการ OS-YATSI ให้ประสิทธิภาพดีกว่าวิธีการแบบดั้งเดิมจำนวน 1 โปรแกรมจากจำนวนทั้งหมด 3 โปรแกรม และดีกว่าวิธีการแบบดั้งเดิมอย่างมีนัยสำคัญจำนวน 2 โปรแกรมจากจำนวนทั้งหมด 3 โปรแกรม

แบบ micro-average ในเทอมของมาตรวัด Pr วิธีการ OS-YATSI ให้ประสิทธิภาพดีกว่าวิธีการแบบดั้งเดิมจำนวน 1 โปรแกรมจากจำนวนทั้งหมด 3 โปรแกรม และดีกว่าวิธีการแบบดั้งเดิมอย่างมีนัยสำคัญจำนวน 2 โปรแกรมจากจำนวนทั้งหมด 3 โปรแกรม ในเทอมของมาตรวัด Re วิธีการ OS-YATSI ให้ประสิทธิภาพดีกว่าวิธีการแบบดั้งเดิมอย่างมีนัยสำคัญจำนวน 2 โปรแกรมจากจำนวนทั้งหมด 3 โปรแกรม และในเทอมของมาตรวัด Re วิธีการ OS-YATSI ให้ประสิทธิภาพดีกว่าวิธีการแบบดั้งเดิมอย่างมีนัยสำคัญจำนวน 2 โปรแกรมจากจำนวนทั้งหมด 3 โปรแกรม

จากผลการทดลองทั้งหมดแสดงให้เห็นว่า กรอบงานสำหรับตรวจจับและจัดลำดับข้อบกพร่องของซอฟต์แวร์ในชุดข้อมูลไม่สมดุล สามารถแก้ไขปัญหาดังกล่าวได้ และยังให้ประสิทธิภาพดีกว่าวิธีการแบบดั้งเดิมที่นำมาเปรียบเทียบอย่างมีนัยสำคัญ ซึ่งเป็นข้อบ่งชี้ว่ากรอบงานที่ได้นำเสนอนี้จะสามารถช่วยนักพัฒนาในการตัดสินใจแก้ไขข้อบกพร่องตามระดับความรุนแรงที่ส่งผลกระทบต่อซอฟต์แวร์ได้

นอกจากนี้ผู้วิจัยยังได้มีการออกแบบและพัฒนาเครื่องมือต้นแบบที่สนับสนุนกรอบงานสำหรับตรวจจับและจัดลำดับข้อบกพร่องของซอฟต์แวร์ในชุดข้อมูลไม่สมดุล ซึ่งสามารถตรวจจับและจัดลำดับข้อบกพร่องตามระดับความรุนแรงได้อย่างอัตโนมัติ ทั้งยังอำนวยความสะดวกให้แก่นักพัฒนาในการแก้ไขข้อบกพร่องที่เกิดขึ้นในซอฟต์แวร์ได้อย่างถูกต้อง รวดเร็วและอยู่ภายในระยะเวลาที่กำหนดอีกด้วย

6.2 ข้อจำกัดของงานวิจัย

ข้อจำกัดของงานวิจัยนี้ประกอบด้วย

1) ค่าผลกระทบการขึ้นต่อกันของคลาส (CDI) อาจไม่ได้สะท้อนให้เห็นผลกระทบหรือความรุนแรงต่อซอฟต์แวร์ที่เกิดขึ้นจริงได้ในกรณีที่ค่า CDI < 1 เนื่องจากเมื่อพิจารณานิยามตามบริบทของงานวิจัยจะพบว่า

- CDI = 0 ก็ต่อเมื่อ Ca = 0 นั่นหมายความว่า ไม่มีคลาสใดในโปรแกรมที่มาเรียกใช้งานคลาสที่เรากำลังพิจารณา ซึ่งหากเกิดข้อบกพร่องขึ้นในคลาสที่เรากำลังพิจารณาก็จะไม่ส่งผลกระทบกับคลาสอื่นๆในโปรแกรม

- CDI = 1 ก็ต่อเมื่อ Ce = 0 นั่นหมายความว่า คลาสที่เรากำลังพิจารณาไม่ได้มีการเรียกใช้งานคลาสใดในโปรแกรม มีแต่คลาสอื่นๆในโปรแกรมที่เรียกใช้งานคลาสที่เรากำลังพิจารณา ดังนั้น หากเกิดข้อบกพร่องขึ้นในคลาสที่เรากำลังพิจารณาก็จะส่งผลกระทบจำนวนมากภายในโปรแกรม

- อย่างไรก็ตาม หาก CDI มีค่าน้อยกว่า 1 ไม่ได้หมายความว่า ผลกระทบที่เกิดขึ้นในโปรแกรมจะน้อยกว่ากรณีที่ CDI = 1 เพราะ CDI < 1 ก็ต่อเมื่อ Ca และ Ce ไม่เท่ากับศูนย์ หรือกล่าวได้ว่า Ca มีค่าเท่าเดิม เพียงแต่มีค่า Ce เพิ่มขึ้นเท่านั้น ซึ่งผลกระทบที่เกิดขึ้นก็น่าจะเท่ากับกรณีที่ CDI = 1 หรือความจริงแล้วการที่ Ce $\neq$ 0 อาจทำให้ผลกระทบที่เกิดขึ้นมากกว่าการที่ Ca $\neq$ 0 อย่างเดียวก็ได้ เพราะถ้าหากเกิดข้อบกพร่องขึ้นในคลาสที่เรากำลังพิจารณา แล้วคลาสนี้มีการเรียกใช้งานคลาสอื่นๆภายในโปรแกรมด้วย ผลการทำงานที่ผิดพลาดของคลาสที่เรากำลังพิจารณานี้อาจส่งผลให้การเรียกใช้งานคลาสอื่นๆภายในโปรแกรมผิดพลาดด้วยเช่นกัน

2) การจัดลำดับข้อบกพร่องตามระดับความรุนแรงสามารถระบุถึงระดับความรุนแรงได้เพียง 3 ระดับเท่านั้น

3) เครื่องมือต้นแบบที่พัฒนารองรับไฟล์ข้อมูลที่มีนามสกุล “.csv” และมีการกำหนดรูปแบบไฟล์นำเข้าตามบริบทของงานวิจัยเท่านั้น

6.3 งานวิจัยในอนาคต

สามารถแบ่งการทำงานวิจัยในอนาคตได้ดังนี้

1) ปรับปรุงแบบจำลองโดยนำเสนอหรือประยุกต์ใช้วิธีการอื่นๆ ที่มีสามารถแก้ไขปัญหาความไม่สมดุลของคลาสได้ เพื่อให้แบบจำลองสามารถตรวจจับข้อบกพร่องได้แม่นยำและมีประสิทธิภาพมากขึ้น

2) นำเสนอหรือประยุกต์ใช้มาตรวัดซอฟต์แวร์อื่นๆ ที่มีความสัมพันธ์กับการทำนายและการจัดลำดับข้อบกพร่องของซอฟต์แวร์ตามระดับความรุนแรง

6.4 ผลงานตีพิมพ์จากวิทยานิพนธ์

1) หัวเรื่องงานวิจัยที่ตีพิมพ์ชื่อ “Software Defect Prediction in Imbalanced Data Sets Using Unbiased Support Vector Machine” ในการประชุมวิชาการระดับนานานชาติ “The 6th International Conference on Information Science and Applications (ICISA 2015)” ซึ่งจัดขึ้น ณ จังหวัดชลบุรี ประเทศไทย ระหว่างวันที่ 24 – 26 กุมภาพันธ์ พ.ศ. 2558

2) หัวเรื่องงานวิจัยที่ตีพิมพ์ชื่อ “Improve Accuracy of Defect Severity Categorization Using Semi-Supervised Approach on Imbalanced Data Sets” ในการประชุมวิชาการระดับนานานชาติ “The 24th International MultiConference of Engineers and Computer Scientists (IMECS 2016)” ซึ่งจัดขึ้น ณ เมืองเกาลูน เขตบริหารพิเศษฮ่องกงแห่งสาธารณรัฐประชาชนจีน ระหว่างวันที่ 16 – 18 มีนาคม พ.ศ. 2559

รายการอ้างอิง

1. Elish, K.O. and M.O. Elish, *Predicting defect-prone software modules using support vector machines.* J. Syst. Softw., 2008. 81(5): p. 649-660.
2. Seliya, N., T.M. Khoshgoftaar, and J. Van Hulse. *Predicting Faults in High Assurance Software*. in *High-Assurance Systems Engineering (HASE), 2010 IEEE 12th International Symposium on*. 2010.
3. Gray, D., et al. *Software defect prediction using static code metrics underestimates defect-proneness*. in *Neural Networks (IJCNN), The 2010 International Joint Conference on*. 2010.
4. Shuo, W. and Y. Xin, *Using Class Imbalance Learning for Software Defect Prediction.* Reliability, IEEE Transactions on, 2013. 62(2): p. 434-443.
5. *NASA IV & V Facility. Metric Data Program*. Available from: http://MDP.ivv.nasa.org/.
6. Menzies, T. and A. Marcus. *Automated severity assessment of software defect reports*. in *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*. 2008.
7. Lamkanfi, A., et al. *Predicting the severity of a reported bug*. in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. 2010.
8. Lamkanfi, A., et al. *Comparing Mining Algorithms for Predicting the Severity of a Reported Bug*. in *Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on*. 2011.
9. Chaturvedi, K.K. and V.B. Singh. *Determining Bug severity using machine learning techniques*. in *Software Engineering (CONSEG), 2012 CSI Sixth International Conference on*. 2012.
10. Cheng-Zen, Y., et al. *An Empirical Study on Improving Severity Prediction of Defect Reports Using Feature Selection*. in *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*. 2012.

11. Singha Roy, N.K. and B. Rossi. *Towards an Improvement of Bug Severity Classification*. in *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*. 2014.
12. Vateekul, P., S. Dendamrongvit, and M. Kubat, *Improving SVM Performance in Multi-Label Domains: Threshold Adjustment.* International Journal on Artificial Intelligence Tools, 2013.
13. Saaty, T.L., *The Analytic Hierarchy Process*. 1980, New York: McGraw-Hill.
14. Driessens, K., et al., *Using Weighted Nearest Neighbor to Benefit from Unlabeled Data*, in *Advances in Knowledge Discovery and Data Mining*, W.-K. Ng, et al., Editors. 2006, Springer Berlin Heidelberg. p. 60-69.
15. Chawla, N.V., et al., *SMOTE: synthetic minority over-sampling technique.* J. Artif. Int. Res., 2002. 16(1): p. 321-357.
16. Chidamber, S.R. and C.F. Kemerer, *A metrics suit for object oriented design*, in *IEEE Transactions on Software Engineering*. 1994. p. 476-493.
17. Martin, R., *OO Design Quality Metrics - An Analysis of Dependencies*, in *Proc. of Workshop Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, OOPSLA'94*. 1994.
18. Bansiya, J. and C.G. Davis, *A hierarchical model for object-oriented design quality assessment.* IEEE Transactions on Software Engineering, 2002. 28(1): p. 4-17.
19. Mei-Huei, T., K. Ming-Hung, and C. Mei-Hwa. *An empirical study on object-oriented metrics*. in *Software Metrics Symposium, 1999. Proceedings. Sixth International*. 1999.
20. McCabe, T.J., *A complexity measure*, in *IEEE Transactions on Software Engineering*. 1976. p. 308-320.
21. Jureczko, M. and D. Spinellis. *Using Object-Oriented Design Metrics to Predict Software Defects*. in *Models and Methodology of System Dependability. Proceedings of 2010: Fifth International Conference on Dependability of Computer Systems*. 2010. Poland: Oficyna Wydawnicza Politechniki Wroc awskiej.
22. Mitchell, T., *Machine Learning*. 1997, s.l.: McGraw-Hill.

23. Han, J. and M. Kamber, *Data Mining: Concepts and Techniques*. 2 ed. 2006, s.l.: Morgan Kaufmann.
24. Breiman, L., *Random Forests.* Mach. Learn., 2001. 45(1): p. 5-32.
25. Zhu, X., *SemiSupervised classification learning survey*. 2005: Computer Sciences TR 1530 , University of Wisconsin-Madison.
26. Huizingh, E.K.R.E. and H.C.J. Vrolijk, *Decision support for information systems management: applying analytic hierarchy process*. 1995, s.n.
27. *IEEE Standard Glossary of Software Engineering Terminology.* IEEE Std 610.12-1990, 1990: p. 1-84.
28. *IEEE Standard Classification for Software Anomalies.* IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993), 2010: p. 1-23.
29. Rijsbergen, C.J.V., *Information Retrieval*. 2 ed. 1979, London: Butterworths.
30. Kubat, M. and S. Matwin, *Addressing the curse of imbalanced training sets: one-sided selection*, in *Proc. 14th International Conference on Machine Learning*. 1997, Morgan Kaufmann. p. 179-186.
31. Yang, Y., *An Evaluation of Statistical Approaches to Text Categorization.* Inf. Retr., 1999. 1(1-2): p. 69-90.
32. *SourceForge*. Available from: http://www.sourceforge.net.
33. Menzies, T., Krishna, R., Pryor, D., *The Promise Repository of Empirical Software Engineering Data*. 2015.
34. Vateekul, P., M. Kubat, and K. Sarinnapakorn, *Top-down optimized SVMs for hierarchical multi-label classification: A case study in gene function prediction*. Intelligent Data Analysis.
35. *WEKA Software*. Available from: http://www.cs.waikato.ac.th.nz/ml/weka.
36. Martin, R.C., *Agile Software Development: Principles, Patterns, and Practices*. 2003: Prentice Hall PTR. 710.
37. D'Ambros, M., M. Lanza, and R. Robbes. *An extensive comparison of bug prediction approaches*. in *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. 2010.
38. Rice, J.A., *Mathematical Statistics and Data Analysis*. 1995: Duxbury Press.

39. Chih-Wei, H., C. Chih-Chung, and L. Chih-Jen, *A Practical Guide to Support Vector Classification*. 2003, National Taiwan University: Department of Computer Science.

ภาคผนวก

ภาคผนวก ก
การใช้งานเครื่องมือต้นแบบ

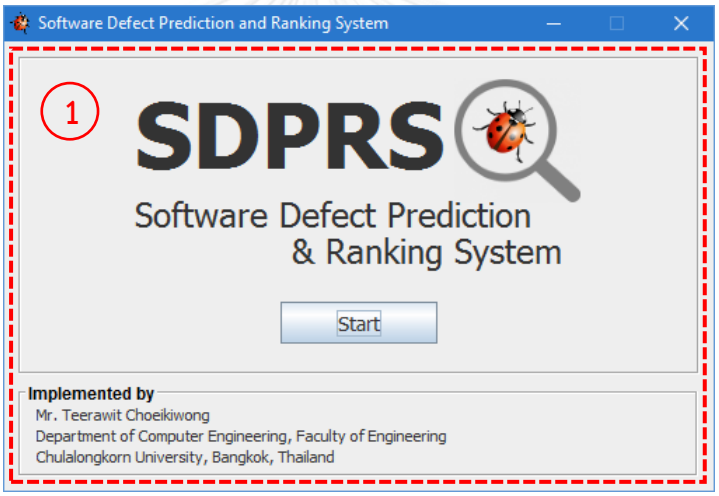
เครื่องมือต้นแบบนี้เป็นเครื่องมือที่พัฒนาจากงานวิจัยที่ได้นำเสนอกรอบงานสำหรับตรวจจับและจัดลำดับข้อบกพร่องของซอฟต์แวร์ในชุดข้อมูลไม่สมดุล โดยเครื่องมือนี้จะมีการนำเข้าไฟล์ในรูปแบบของตาราง (.csv) ซึ่งจะแบ่งการทำงานของโปรแกรมออกเป็น 4 ขั้นตอน คือ

1) ขั้นตอนการนำเข้าไฟล์

เมื่อเปิดโปรแกรมจะแสดงหน้าต่างดังรูปที่ 31 ซึ่งเป็นหน้าจอแสดงการต้อนรับของเครื่องมือ (หมายเลข 1) โดยจะประกอบด้วย 2 ส่วน คือ

ส่วนที่ 1 แสดงชื่อของเครื่องมือ และเมนู Start

ส่วนที่ 2 แสดงรายละเอียดข้อมูลเกี่ยวกับผู้พัฒนา



รูปที่ 31 หน้าต่างแสดงหน้าจอต้อนรับ

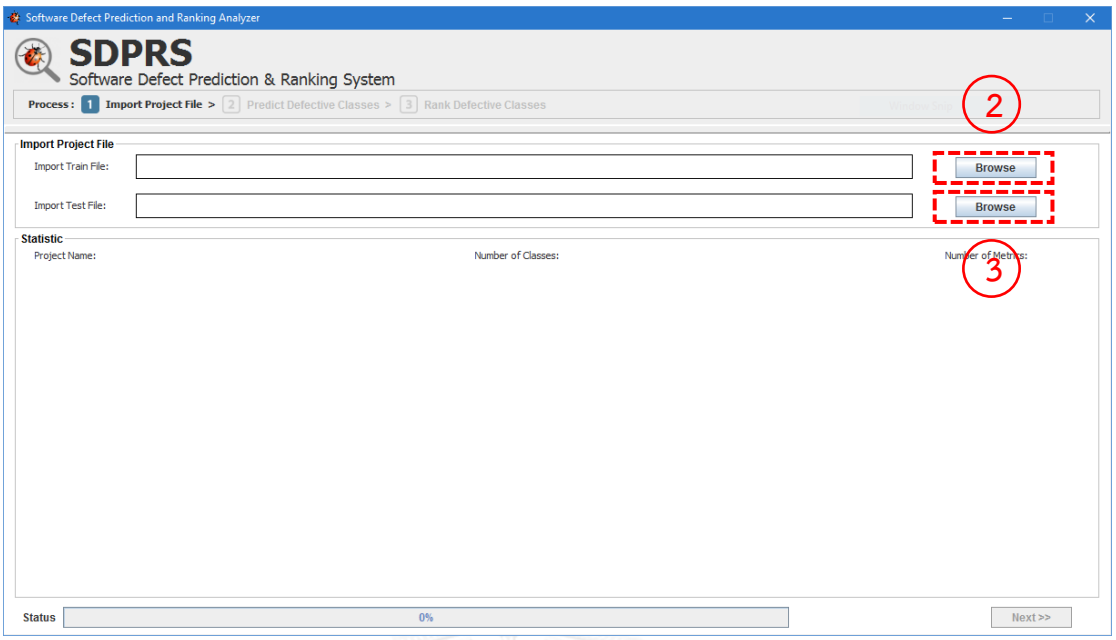
เริ่มต้นการใช้งานเครื่องมือ เมื่อผู้ใช้คลิกที่ปุ่ม 

เครื่องมือจะแสดงหน้าต่างใหม่เพื่อเริ่มการทำงาน โดยจะแสดงหน้าต่างรอการนำเข้าไฟล์เพื่อทำนายข้อบกพร่องดังรูปที่ 32

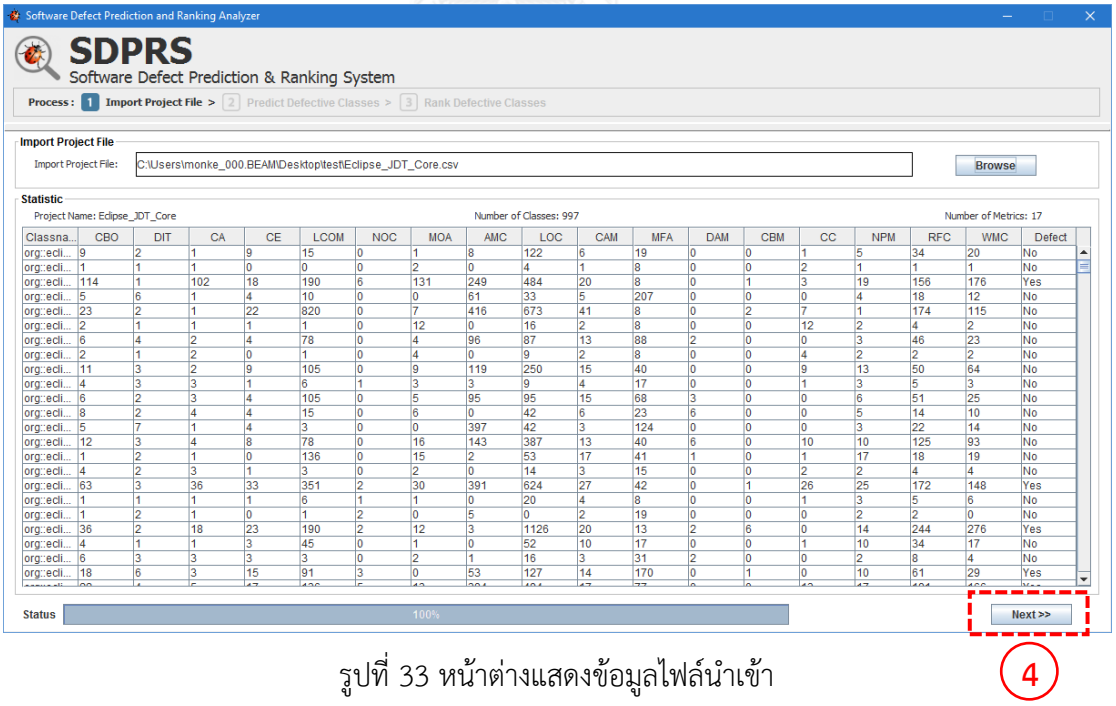
จากนั้นผู้ใช้สามารถเลือกไฟล์โครงการซอฟต์แวร์ที่ต้องการทำนายข้อบกพร่องโดยการนำเข้าไฟล์ข้อมูลสอนให้คลิกที่ปุ่ม 

(หมายเลข 2) และนำเข้าไฟล์ข้อมูลทดสอบให้คลิกที่ปุ่ม 

(หมายเลข 3) เครื่องมือจะแสดงรายละเอียดข้อมูลของไฟล์นำเข้าในรูปแบบตารางดังรูปที่ 33 ซึ่งประกอบด้วย ชื่อโครงการซอฟต์แวร์ ชื่อคลาส จำนวนคลาสทั้งหมด จำนวนมาตรวัดทั้งหมด และค่ามาตรวัดแต่ละตัว ถ้าหากผู้ใช้ไม่ได้ขำเข้าไฟล์โครงการซอฟต์แวร์ เครื่องมือก็จะไม่อนุญาตให้ทำขั้นตอนถัดไปได้



รูปที่ 32 หน้าต่างหลักรอการนำเข้าไฟล์



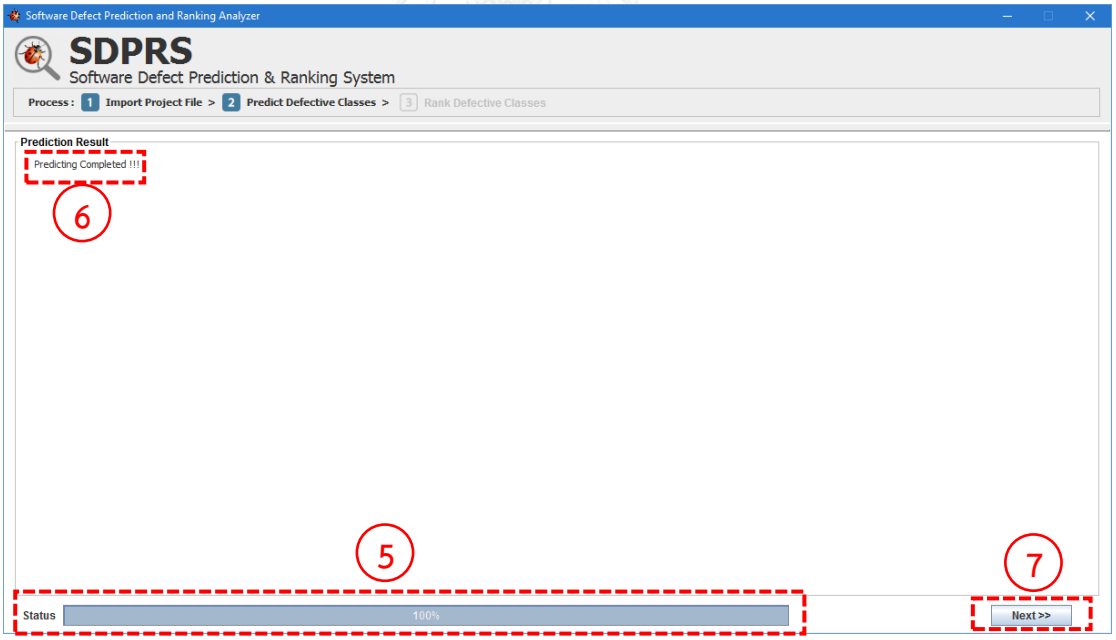
รูปที่ 33 หน้าต่างแสดงข้อมูลไฟล์นำเข้า

2) ขั้นตอนการทำนายข้อบกพร่อง

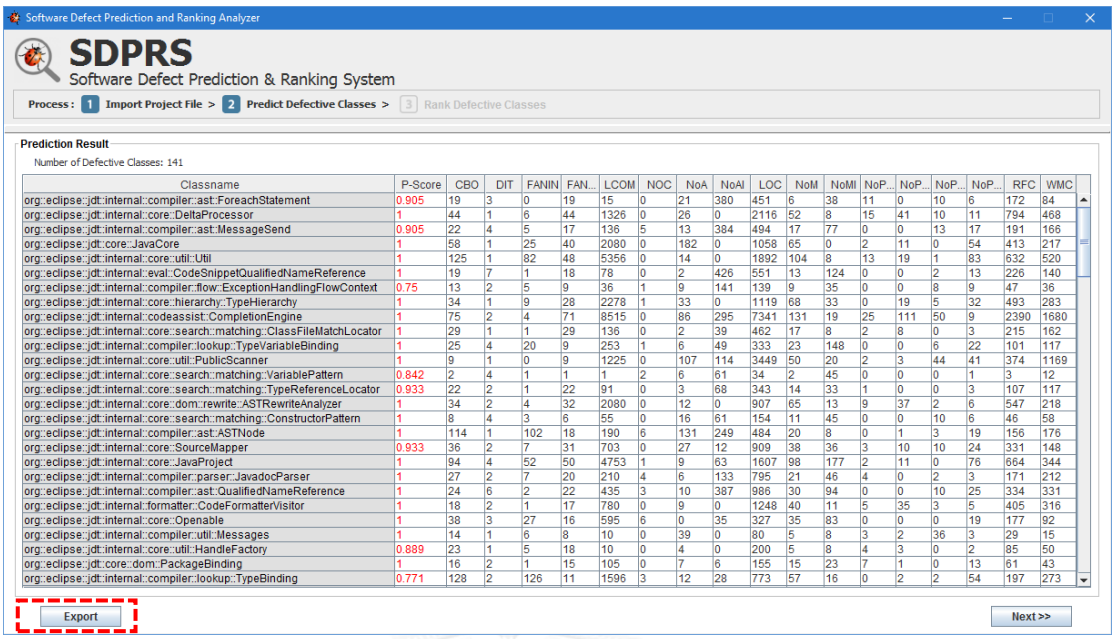
ในขั้นตอนนี้จะเป็นขั้นตอนการทำนายข้อบกพร่อง โดยหลังจากผู้ใช้นำเข้าไฟล์โครงการซอฟต์แวร์ในขั้นตอนที่ 1) แล้ว สามารถเริ่มต้นการทำนายได้โดยการคลิกที่ปุ่ม Next >> (จากรูปที่ 33 หมายเลข 4)

จากนั้นเครื่องมือจะเข้าสู่กระบวนการทำนายข้อบกพร่องดังรูปที่ 34 โดยจะแสดงแถบสถานะการทำนายข้อบกพร่อง (หมายเลข 5) ซึ่งผู้ใช้จะต้องรอจนกว่าแถบสถานะการทำนายจะครบ 100% เมื่อแถวสถานะครบ 100% แล้วจะแสดงข้อความ “Predicting Completed !!!” (หมายเลข 6) ซึ่งหมายความว่าเครื่องมือทำนายข้อบกพร่องเสร็จแล้ว ผู้ใช้สามารถเลือกดูผลลัพธ์การทำนายข้อบกพร่องได้โดย คลิกที่ปุ่ม Next >> (หมายเลข 7) เครื่องมือก็จะแสดงผลลัพธ์การทำนายข้อบกพร่องดังรูปที่ 35

หลังจากที่ผู้ใช้เรียกดูผลลัพธ์แล้ว สามารถนำออกไฟล์ เพื่อบันทึกผลลัพธ์ไปใช้งานได้ในภายหลัง โดยคลิกที่ปุ่ม Export (หมายเลข 8)

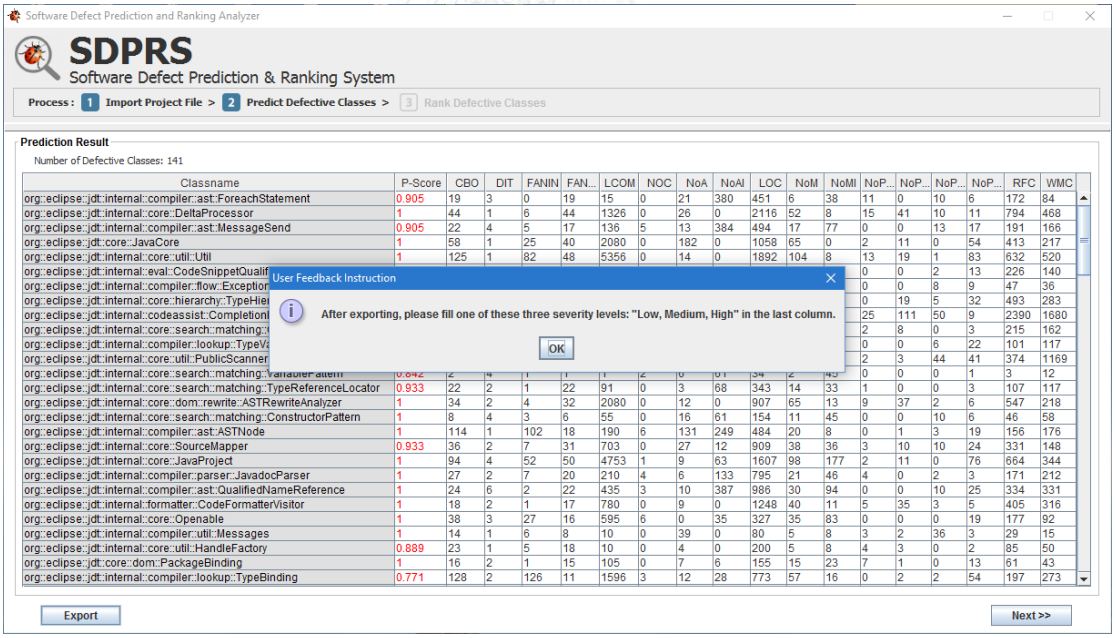


รูปที่ 34 หน้าต่างแสดงสถานะการทำนายข้อบกพร่อง



8

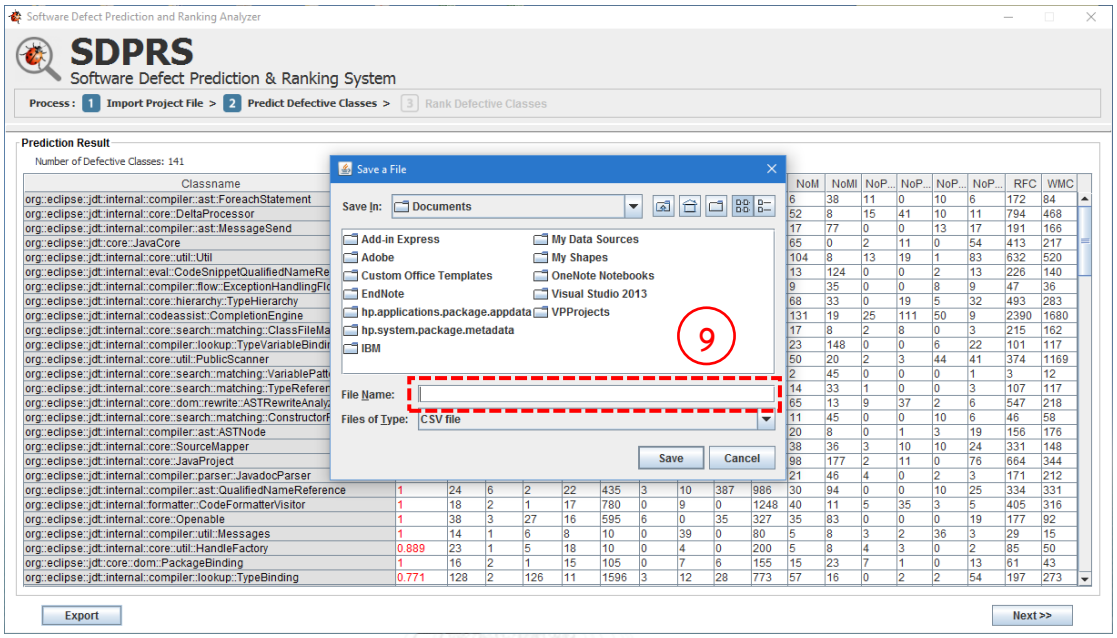
รูปที่ 35 หน้าต่างแสดงผลลัพธ์จากการทำนายข้อบกพร่อง



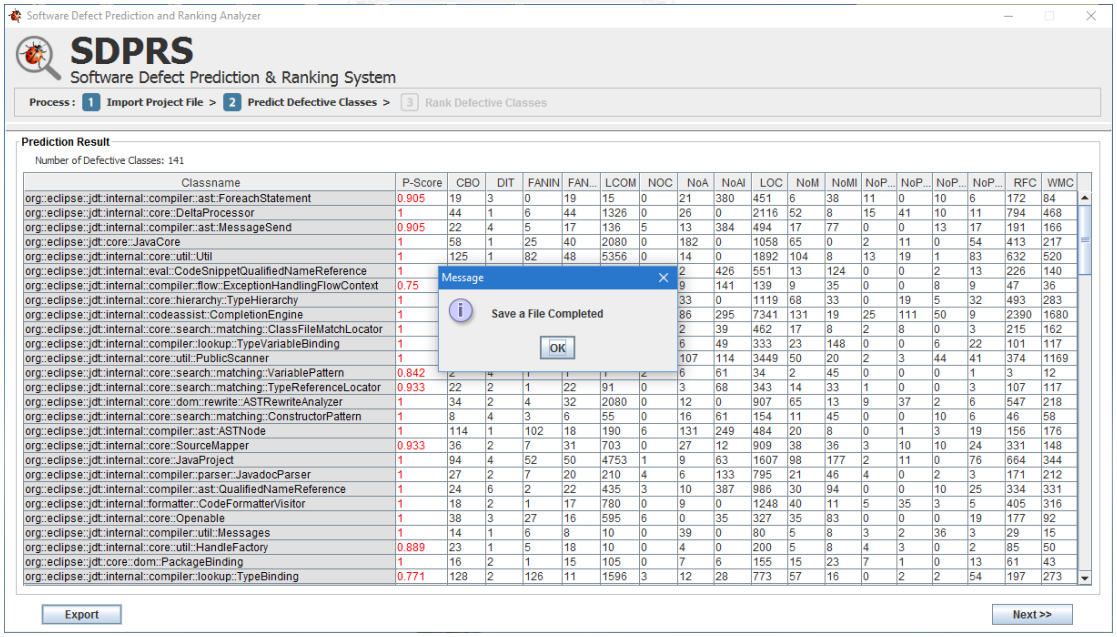
รูปที่ 36 หน้าต่างแสดงคำแนะนำเมื่อนำออกไฟล์ผลลัพธ์การทำนายข้อบกพร่อง

หลังจากผู้ใช้กดปุ่ม Export (หมายเลข 8) แล้ว เครื่องมือจะแสดงหน้าต่างข้อความแจ้งเตือนคำแนะนำดังรูปที่ 36 โดยมีรายละเอียดว่า “หากผู้ใช้ต้องการนำออกไฟล์ไฟล์ เพื่อใช้เป็นข้อมูลนำเข้าในการจัดลำดับ ผู้ใช้จะสามารถกรอกระดับความรุนแรงได้เพียง 3 ระดับ ตามที่กำหนดเท่านั้น” เมื่อผู้ใช้คลิกปุ่ม OK เครื่องมือจึงจะแสดงหน้าต่างสำหรับบันทึกไฟล์ดังรูปที่ 37 ให้ผู้ใช้

กรอกที่อยู่สำหรับบันทึกไฟล์ลงในช่องหมายเลข 9 หลังจากผู้ใช้บันทึกไฟล์เสร็จ เครื่องมือจะแสดงหน้าต่างแสดงข้อความแจ้งเตือนเมื่อบันทึกไฟล์ผลลัพธ์การทำนายข้อบกพร่องสำเร็จดังรูปที่ 38



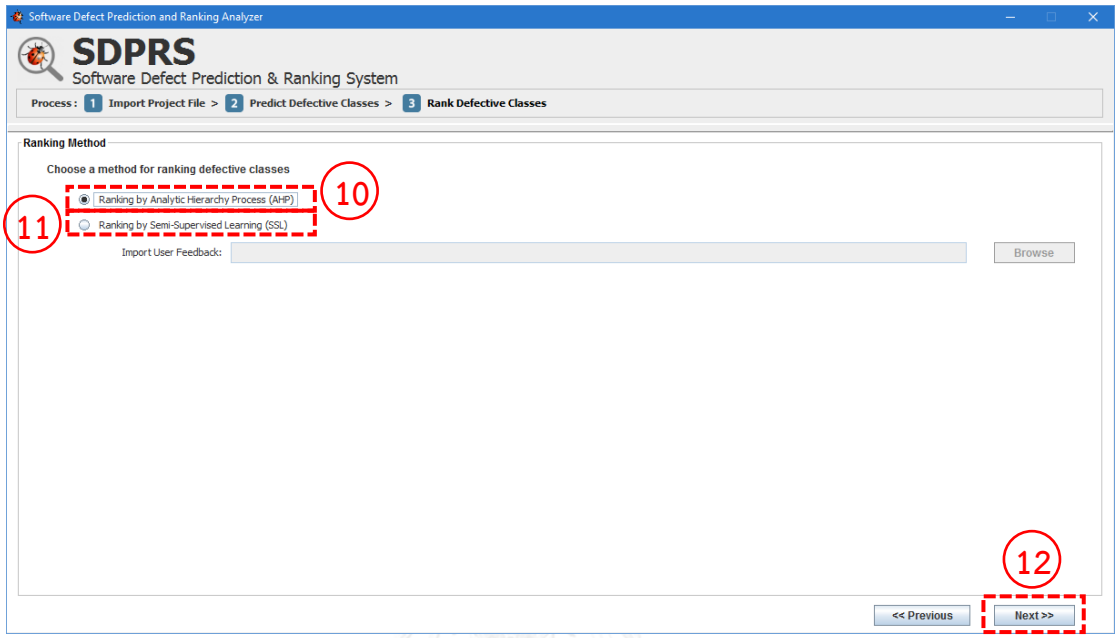
รูปที่ 37 หน้าต่างแสดงการบันทึกไฟล์ผลลัพธ์การทำนายข้อบกพร่อง



รูปที่ 38 หน้าต่างแสดงข้อความแจ้งเตือนเมื่อบันทึกไฟล์ผลลัพธ์การทำนายข้อบกพร่องสำเร็จ

3) ขั้นตอนการจัดลำดับข้อบกพร่องด้วยวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์

ในขั้นตอนนี้จะเป็นการจัดลำดับข้อบกพร่องโดยเครื่องมือจะแสดงเมนูการจัดลำดับข้อบกพร่อง ซึ่งประกอบด้วย 2 วิธี คือ AHP (หมายเลข 10) และ SSL (หมายเลข 11)

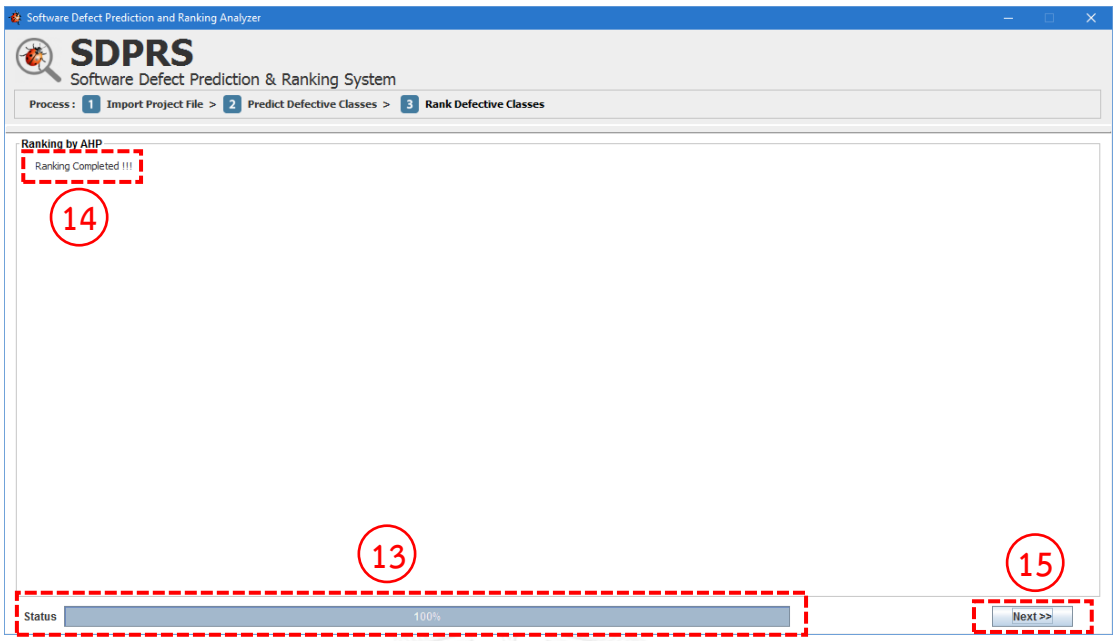


รูปที่ 39 หน้าต่างแสดงเมนูการจัดลำดับข้อบกพร่อง

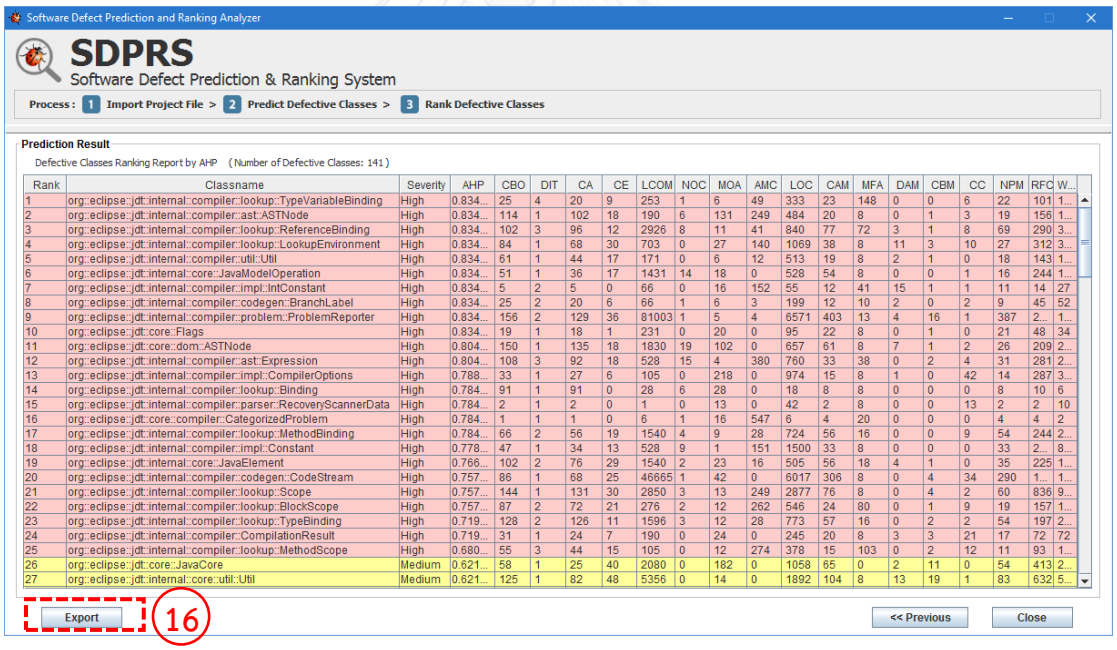
เมื่อผู้ใช้เลือกการจัดลำดับข้อบกพร่องด้วยวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์ดังหมายเลข 10 แล้ว สามารถคลิกที่ปุ่ม Next >> (หมายเลข 12) เพื่อเริ่มการจัดลำดับได้ทันที เครื่องมือจะแสดงหน้าต่างสถานะการจัดลำดับข้อบกพร่องด้วยวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์ดังรูปที่ 40

จากนั้นเครื่องมือจะเข้าสู่กระบวนการจัดลำดับข้อบกพร่องดังรูปที่ 40 โดยจะแสดงแถบสถานะการทำนายข้อบกพร่อง (หมายเลข 13) ซึ่งผู้ใช้จะต้องรอจนกว่าแถบสถานะการทำนายจะครบ 100% เมื่อแถวสถานะครบ 100% แล้วจะแสดงข้อความ “Ranking Completed !!!” (หมายเลข 14) ซึ่งหมายความว่าเครื่องมือทำนายข้อบกพร่องเสร็จแล้ว ผู้ใช้สามารถเลือกดูผลลัพธ์การทำนายข้อบกพร่องได้โดย คลิกที่ปุ่ม Next >> (หมายเลข 15) เครื่องมือก็จะแสดงผลลัพธ์การทำนายข้อบกพร่องดังรูปที่ 41

หลังจากที่ผู้ใช้เรียกดูผลลัพธ์แล้ว สามารถนำออกไฟล์ เพื่อบันทึกผลลัพธ์ไปใช้งานได้ในภายหลัง โดยคลิกที่ปุ่ม Export (หมายเลข 16)



รูปที่ 40 หน้าต่างแสดงสถานะการจัดลำดับข้อบกพร่องด้วยวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์

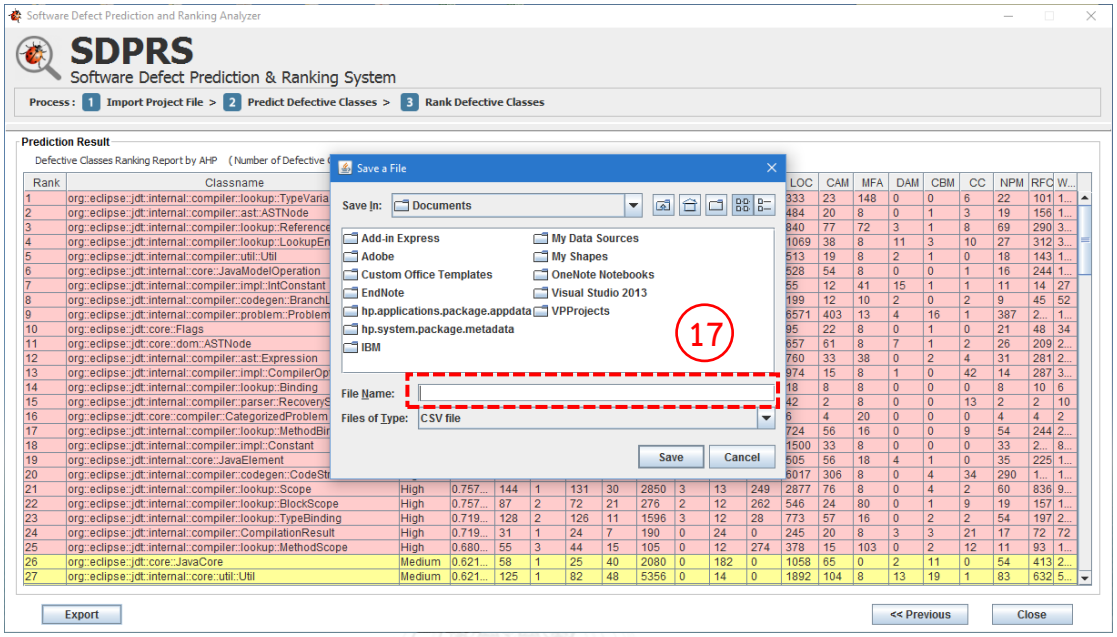


Rank	Classname	Severity	AHP	CBO	DIT	CA	CE	LCOM	NOC	MOA	AMC	LOC	CAM	MFA	DAM	CBM	CC	NPM	RFC	W...
1	org::eclipse::jdt::internal::compiler::lookup::TypeVariableBinding	High	0.834...	25	4	20	9	253	1	6	49	333	23	148	0	0	6	22	101	1...
2	org::eclipse::jdt::internal::compiler::ast::ASTNode	High	0.834...	114	1	102	18	190	6	131	249	484	20	8	0	1	3	19	156	1...
3	org::eclipse::jdt::internal::compiler::lookup::ReferenceBinding	High	0.834...	102	3	96	12	2926	8	11	41	840	77	72	3	1	8	69	290	3...
4	org::eclipse::jdt::internal::compiler::lookup::LookupEnvironment	High	0.834...	84	1	68	30	703	0	27	140	1069	38	8	11	3	10	27	312	3...
5	org::eclipse::jdt::internal::compiler::util::Util	High	0.834...	61	1	44	17	171	0	6	12	513	19	8	2	1	0	18	143	1...
6	org::eclipse::jdt::internal::core::JavaModelOperation	High	0.834...	51	1	36	17	1431	14	18	0	528	54	8	0	0	1	16	244	1...
7	org::eclipse::jdt::internal::compiler::impl::IntConstant	High	0.834...	5	2	5	0	66	0	16	152	55	12	41	15	1	1	11	14	27
8	org::eclipse::jdt::internal::compiler::codegen::BranchLabel	High	0.834...	25	2	20	6	66	1	6	3	199	12	10	2	0	2	9	45	52
9	org::eclipse::jdt::internal::compiler::problem::ProblemReporter	High	0.834...	156	2	129	36	81003	1	5	4	6571	403	13	4	16	1	387	2...	1...
10	org::eclipse::jdt::core::Flags	High	0.834...	19	1	18	1	231	0	20	0	95	22	8	0	1	0	21	48	34
11	org::eclipse::jdt::core::dom::ASTNode	High	0.804...	150	1	135	18	1830	19	102	0	657	61	8	7	1	2	26	209	2...
12	org::eclipse::jdt::internal::compiler::ast::Expression	High	0.804...	108	3	92	18	528	15	4	380	760	33	38	0	2	4	31	281	2...
13	org::eclipse::jdt::internal::compiler::impl::CompilerOptions	High	0.788...	33	1	27	6	105	0	218	0	974	15	8	1	0	42	14	287	3...
14	org::eclipse::jdt::internal::compiler::lookup::Binding	High	0.784...	91	1	91	0	28	6	28	0	18	8	8	0	0	0	8	10	6
15	org::eclipse::jdt::internal::compiler::parser::RecoveryScannerData	High	0.784...	2	1	2	0	1	0	13	0	42	2	8	0	0	13	2	2	10
16	org::eclipse::jdt::core::compiler::CategorizedProblem	High	0.784...	1	1	1	0	6	1	16	547	6	4	20	0	0	0	4	4	2
17	org::eclipse::jdt::internal::compiler::lookup::MethodBinding	High	0.784...	66	2	56	19	1540	4	9	28	724	56	16	0	0	9	54	244	2...
18	org::eclipse::jdt::internal::compiler::impl::Constant	High	0.778...	47	1	34	13	528	9	1	151	1500	33	8	0	0	0	33	2...	8...
19	org::eclipse::jdt::internal::core::JavaElement	High	0.766...	102	2	76	29	1540	2	23	16	505	56	18	4	1	0	35	225	1...
20	org::eclipse::jdt::internal::compiler::codegen::CodeStream	High	0.757...	86	1	68	25	46665	1	42	0	6017	306	8	0	4	34	290	1...	1...
21	org::eclipse::jdt::internal::compiler::lookup::Scope	High	0.757...	144	1	131	30	2850	3	13	249	2877	76	8	0	4	2	60	836	9...
22	org::eclipse::jdt::internal::compiler::lookup::BlockScope	High	0.757...	87	2	72	21	276	2	12	262	546	24	80	0	1	9	19	157	1...
23	org::eclipse::jdt::internal::compiler::lookup::TypeBinding	High	0.719...	128	2	126	11	1596	3	12	28	773	57	16	0	2	2	54	197	2...
24	org::eclipse::jdt::internal::compiler::CompilationResult	High	0.719...	31	1	24	7	190	0	24	0	245	20	8	3	3	21	17	72	72
25	org::eclipse::jdt::internal::compiler::lookup::MethodScope	High	0.680...	55	3	44	15	105	0	12	274	378	15	103	0	2	12	11	93	1...
26	org::eclipse::jdt::core::JavaCore	Medium	0.621...	58	1	25	40	2080	0	182	0	1058	65	0	2	11	0	54	413	2...
27	org::eclipse::jdt::internal::core::util::Util	Medium	0.621...	125	1	82	48	5356	0	14	0	1892	104	8	13	19	1	83	632	5...

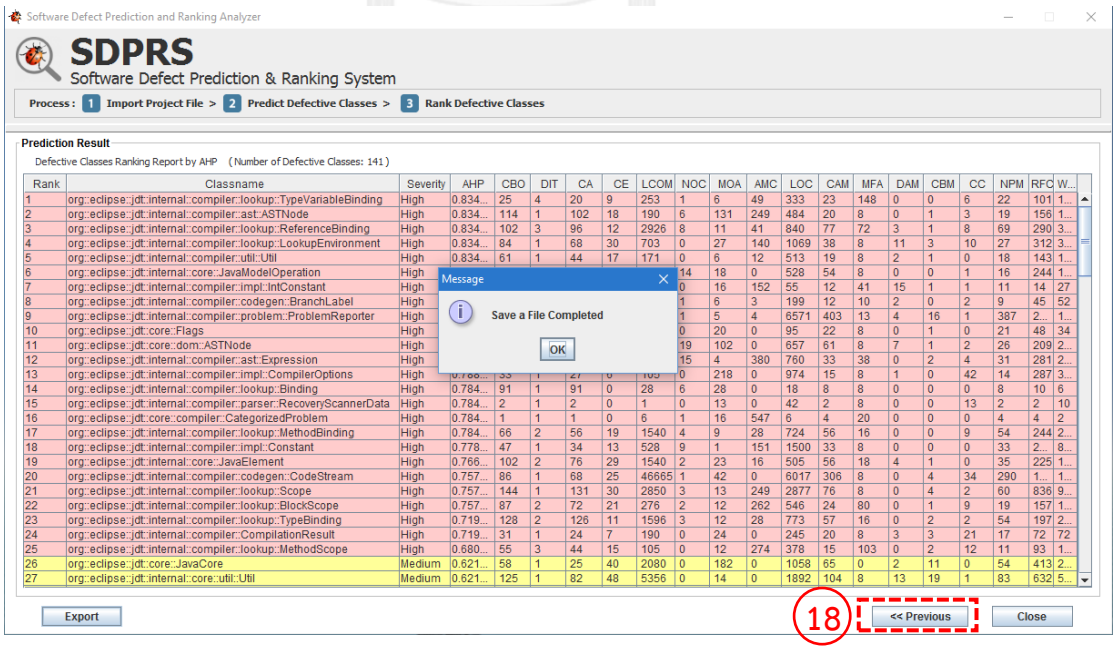
รูปที่ 41 หน้าต่างแสดงผลลัพธ์การจัดลำดับข้อบกพร่องด้วยวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์

หลังจากผู้ใช้กดปุ่ม Export (หมายเลข 16) แล้ว เครื่องมือจึงจะแสดงหน้าต่างสำหรับบันทึกไฟล์ดังรูปที่ 42 ให้ผู้ใช้กรอกที่อยู่สำหรับบันทึกไฟล์ลงในช่องหมายเลข 17 หลังจากผู้ใช้บันทึกไฟล์เสร็จ เครื่องมือจะแสดงหน้าต่างแสดงข้อความแจ้งเตือนเมื่อบันทึกไฟล์ผลลัพธ์การทำนาย

ข้อบกพร่องสำเร็จดังรูปที่ 43 หรือผู้ใช้ต้องการกลับไปจัดลำดับข้อบกพร่องด้วยวิธี SSL ก็สามารถทำได้โดยคลิกที่ปุ่ม << Previous (หมายเลข 18) เพื่อกลับไปยังหน้าเมนูการจัดลำดับข้อบกพร่อง



รูปที่ 42 หน้าต่างแสดงการบันทึกไฟล์ผลลัพธ์การจัดลำดับข้อบกพร่องด้วยวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์



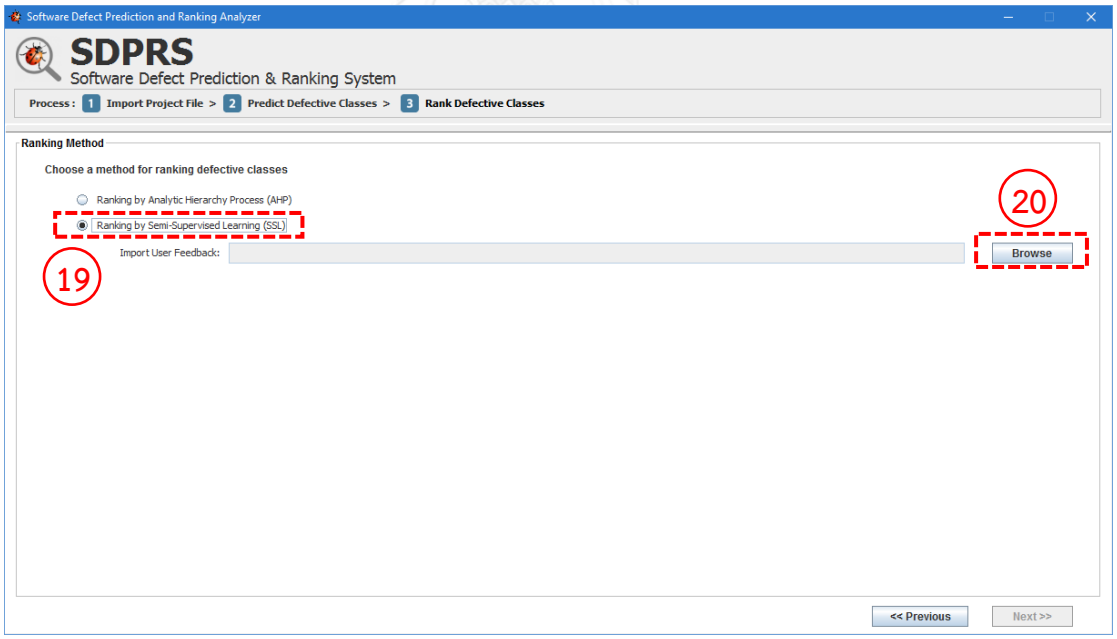
รูปที่ 43 หน้าต่างแสดงข้อความแจ้งเตือนเมื่อบันทึกไฟล์ผลลัพธ์การจัดลำดับข้อบกพร่องด้วยวิธีกระบวนการลำดับชั้นเชิงวิเคราะห์สำเร็จ

4) ขั้นตอนการจัดลำดับข้อบกพร่องด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน

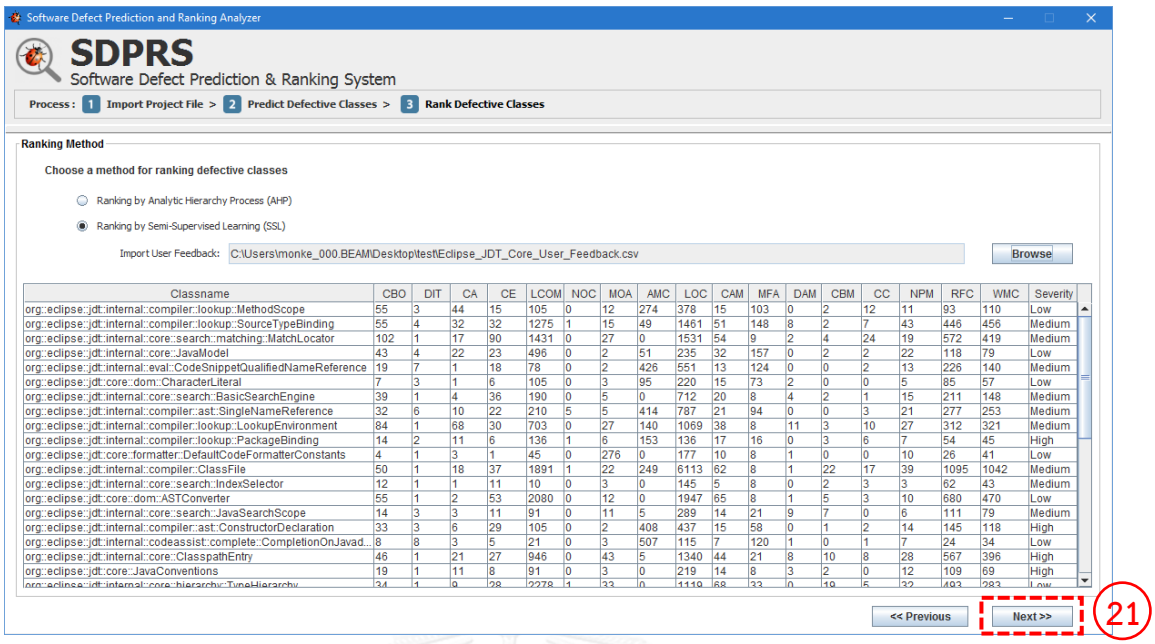
ขั้นตอนนี้เป็นการจัดลำดับข้อบกพร่องด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอนดังรูปที่ 44 ผู้ใช้สามารถคลิกที่ปุ่มหมายเลข 19 เพื่อเลือกการจัดลำดับข้อบกพร่องด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน

จากนั้นผู้ใช้จะต้องนำเข้าไฟล์ข้อมูลป้อนกลับจากผู้ใช้ เพื่อทำไปใช้ในการหาระดับความรุนแรงของข้อบกพร่อง โดยคลิกที่ปุ่ม Browse (หมายเลข 20) เพื่อนำเข้าไฟล์

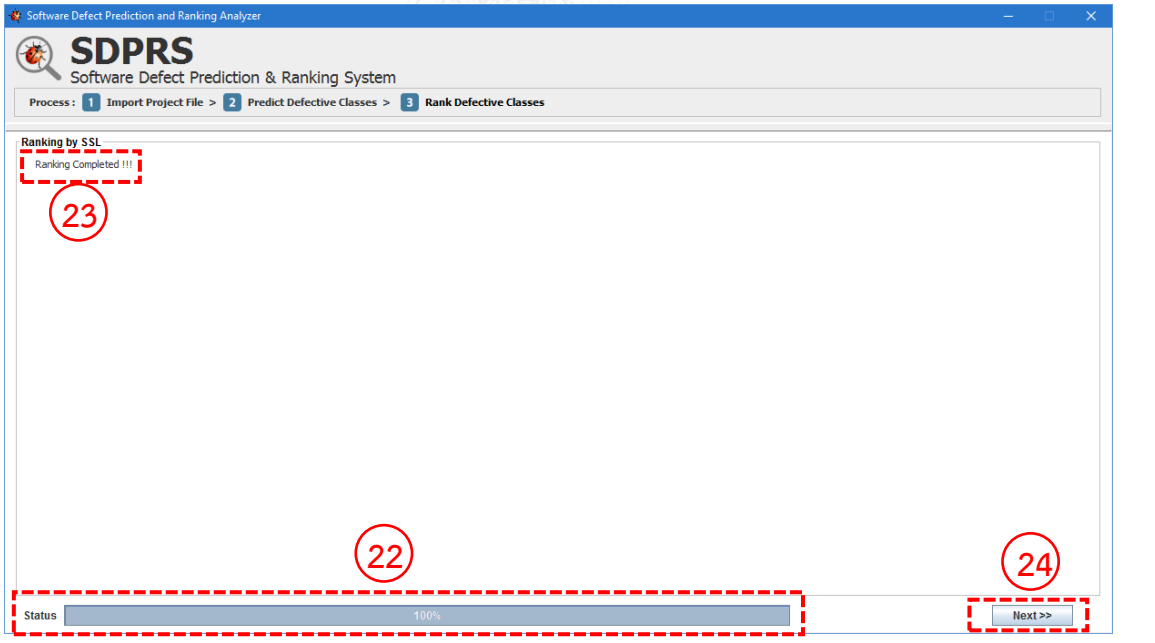
จากนั้นเครื่องมือจะแสดงรายละเอียดข้อมูลนำเข้าดังรูปที่ 45 ผู้ใช้สามารถเริ่มการจัดลำดับข้อบกพร่องได้ทันที โดยคลิกที่ปุ่ม Next >> (หมายเลข 21) จากนั้นเครื่องมือจะเข้าสู่กระบวนการจัดลำดับข้อบกพร่องดังรูปที่ 46 โดยจะแสดงแถบสถานะการทำนายข้อบกพร่อง (หมายเลข 22) ซึ่งผู้ใช้จะต้องรอจนกว่าแถบสถานะการทำนายจะครบ 100% เมื่อแถบสถานะครบ 100% แล้วจะแสดงข้อความ “Ranking Completed !!!” (หมายเลข 23) ซึ่งหมายความว่าเครื่องมือทำนายข้อบกพร่องเสร็จแล้ว ผู้ใช้สามารถเลือกดูผลลัพธ์การทำนายข้อบกพร่องได้โดย คลิกที่ปุ่ม Next >> (หมายเลข 24) เครื่องมือก็จะแสดงผลลัพธ์การทำนายข้อบกพร่องดังรูปที่ 47



รูปที่ 44 หน้าต่างแสดงเมนูการจัดลำดับข้อบกพร่องด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน



รูปที่ 45 หน้าต่างแสดงรายละเอียดข้อมูลนำเข้าของการจัดลำดับข้อบกพร่องด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน

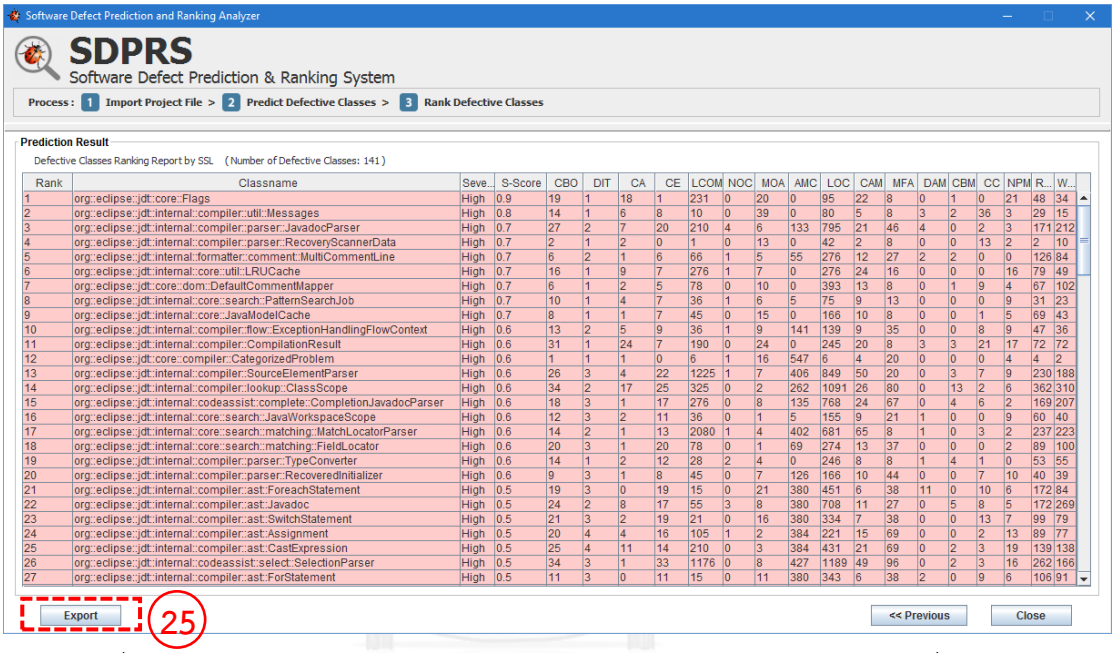


รูปที่ 46 หน้าต่างแสดงแถบสถานะการจัดลำดับข้อบกพร่องด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน

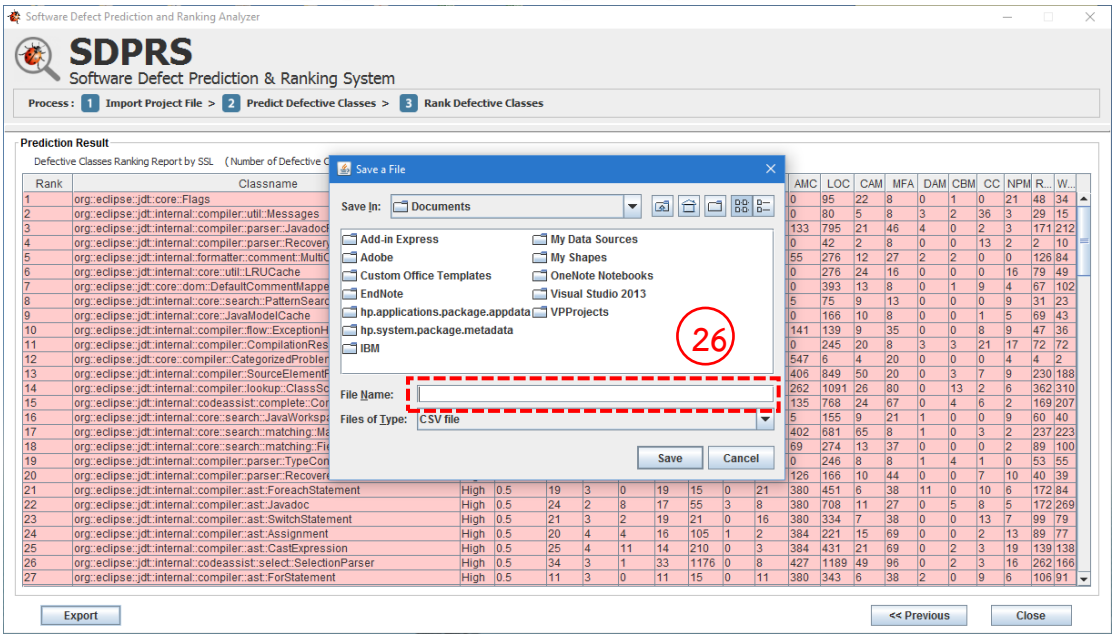
หลังจากที่ผู้ใช้เรียกดูผลลัพธ์แล้ว สามารถนำออกไฟล์ เพื่อบันทึกผลลัพธ์ไปใช้งานได้ในภายหลัง โดยคลิกที่ปุ่ม Export (หมายเลข 25)

หลังจากผู้ใช้กดปุ่ม Export (หมายเลข 25) แล้ว เครื่องมือจึงจะแสดงหน้าต่างสำหรับบันทึกไฟล์ดังรูปที่ 48 ให้ผู้ใช้กรอกที่อยู่สำหรับบันทึกไฟล์ลงในช่องหมายเลข 26 หลังจากผู้ใช้บันทึกไฟล์เสร็จ เครื่องมือจะแสดงหน้าต่างแสดงข้อความแจ้งเตือนเมื่อบันทึกไฟล์ผลลัพธ์การทำนายข้อบกพร่องสำเร็จดัง

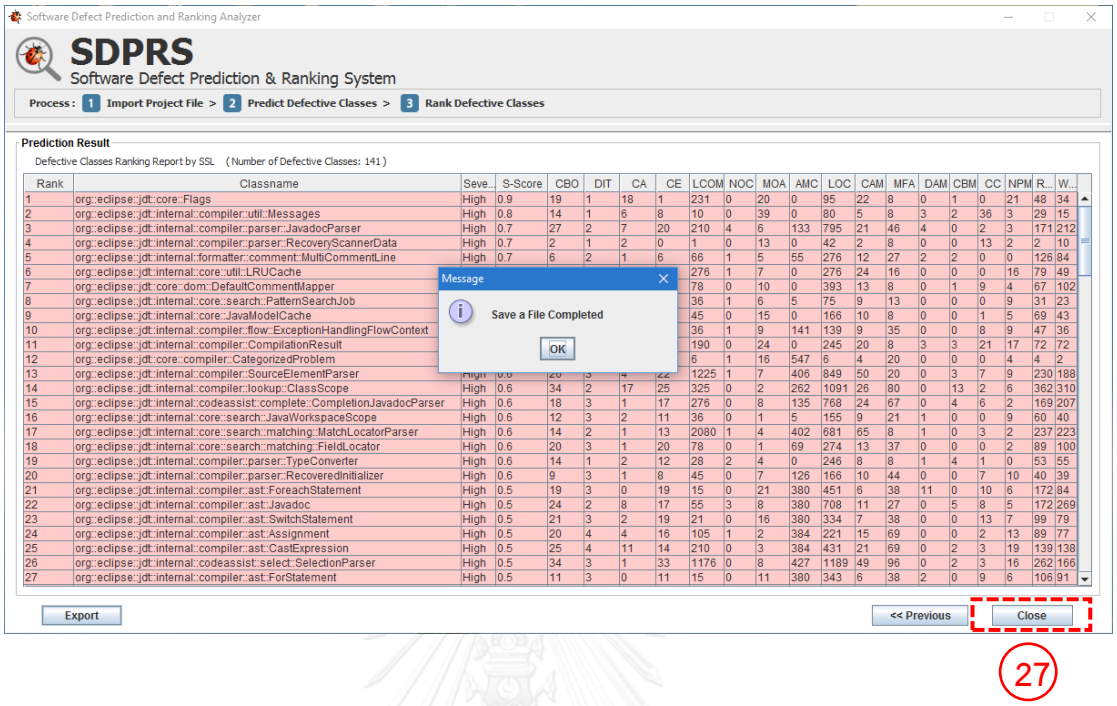
รูปที่ *49* หรือผู้ใช้ต้องการปิดเครื่องมือ ก็สามารถทำได้โดยคลิกที่ปุ่ม Close (หมายเลข 27) เพื่อจบการทำงาน



รูปที่ 47 หน้าต่างแสดงผลลัพธ์การจัดลำดับข้อบกพร่องด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน



รูปที่ 48 หน้าต่างแสดงการบันทึกไฟล์ผลลัพธ์การจัดลำดับข้อบกพร่องด้วยวิธีการเรียนรู้แบบกึ่งมีผู้สอน



รูปที่ 49 หน้าต่างแสดงข้อความแจ้งเตือนเมื่อบันทึกไฟล์ผลลัพธ์การจัดลำดับข้อบกพร่องวิธีการเรียนรู้แบบกึ่งมีผู้สอน

ประวัติผู้เขียนวิทยานิพนธ์

นายธีรวิทย์ เชยกีวงศ์ เกิดเมื่อวันที่ 16 ตุลาคม พ.ศ. 2533 ณ จังหวัดราชบุรี สำเร็จการศึกษาปริญญาวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์และระบบคอมพิวเตอร์ ภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์และเทคโนโลยีอุตสาหกรรม มหาวิทยาลัยศิลปากร ในปีการศึกษา 2555 (เกียรตินิยมอันดับ 2) และเข้าศึกษาในหลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิศวกรรมซอฟต์แวร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2556