การระบุตัวเลือกสินทรัพย์ร่วมในสายผลิตภัณฑ์ซอฟต์แวร์ด้วยการตรวจหาสำเนาโค้ด

นายธนิต เหรียญตระกูล

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

IDENTIFYING COMMON ASSET CANDIDATES IN SOFTWARE PRODUCT LINE BY CODE CLONE DETECTION

Mr. Tanit Reantragoon

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 2011

หัวข้อวิทยานิพนธ์	การระบุตัวเลือกสินทรัพย์ร่วมในสายผลิตภัณฑ์ซอฟต์แวร์ด้วยการตรวจหาสำเนาโค้ด
โดย	นายธนิต เหรียญตระกูล
สาขาวิชา	วิศวกรรมซอฟต์แวร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร.พรศิริ หมื่นไชยศรี

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

…….. คณบดีคณะวิศวกรรมศาสตร์
(รองศาสตราจารย์ ดร.บุญสม เลิศหิรัญวงศ์)

คณะกรรมการสอบวิทยานิพนธ์

…… ประธานกรรมการ
(ศาสตราจารย์ ดร.บุญเสริม กิจศิริกุล)

…….. อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(รองศาสตราจารย์ ดร.พรศิริ หมื่นไชยศรี)

…… กรรมการ
(ผู้ช่วยศาสตราจารย์ นครทิพย์ พร้อมพูล)

…… กรรมการภายนอกมหาวิทยาลัย
(รองศาสตราจารย์ ดร.วีระ บุญจริง)

ธนิต เหรียญตระกูล : การระบุตัวเลือกสินทรัพย์ร่วมในสายผลิตภัณฑ์ซอฟต์แวร์ด้วยการตรวจหาสำเนาโค้ด. (IDENTIFYING COMMON ASSET CANDIDATES IN SOFTWARE PRODUCT LINE BY CODE CLONE DETECTION) อ. ที่ปรึกษาวิทยานิพนธ์หลัก : รศ.ดร.พรศิริ หมื่นไชยศรี, 63 หน้า.

สายผลิตภัณฑ์ซอฟต์แวร์เป็นแนวคิดที่ถูกนำเสนอขึ้นสำหรับการจัดการกลุ่มของซอฟต์แวร์ที่มีความเกี่ยวข้องกันอย่างเป็นระบบเพื่อจะลดค่าใช้จ่ายและแรงงานในการพัฒนาซอฟต์แวร์ ซึ่งการพัฒนาสินทรัพย์ร่วมเพื่อนำไปใช้ซ้ำสำหรับการประกอบและปรับแต่งเป็นผลิตภัณฑ์ นั้น ถือเป็นหนึ่งในกิจกรรมหลักของสายผลิตภัณฑ์ซอฟต์แวร์ โดยผู้เชี่ยวชาญโดเมนอาจวิเคราะห์และสร้างสินทรัพย์ร่วมขึ้นมาใหม่ตั้งแต่เริ่มแรก หรือระบุสินทรัพย์ร่วมจากผลิตภัณฑ์ที่มีอยู่แล้วก็ได้ ซึ่งในระดับซอร์สโค้ดนั้นได้มีการประยุกต์ใช้การตรวจหาสำเนาโค้ดระหว่างผลิตภัณฑ์ เพื่อระบุตัวเลือกสินทรัพย์ร่วมจากผลิตภัณฑ์ที่มีอยู่ อย่างไรก็ตามในกรณีที่สำเนาโค้ดที่ตรวจพบมีจุดแตกต่างอยู่บ้าง จุดแตกต่างนั้นจำเป็นต้องถูกแยกออกมา เพื่อให้สามารถนำสำเนาโค้ดนั้นไปพัฒนาเป็นสินทรัพย์ร่วมได้ งานวิจัยนี้มีจุดประสงค์เพื่อแยกจุดแตกต่างดังกล่าวออกจากสำเนาโค้ดที่ตรวจพบ โดยการจำแนกผลสำเนาโค้ดประเภทที่มีความแตกต่างในระดับเมท็อดแล้วประยุกต์ใช้แบบอย่างการออกแบบ เมท็อดแม่แบบ ทำให้ได้ตัวเลือกสินทรัพย์ร่วมในรูปแบบคลาสแม่ ซึ่งมีการโอเวอร์ไรด์ด้วยโค้ดที่แตกต่างได้ที่คลาสลูกสำหรับแต่ละผลิตภัณฑ์

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต.....

สาขาวิชา วิศวกรรมซอฟต์แวร์ ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก.....

ปีการศึกษา 2554 .

5170455921 : MAJOR SOFTWARE ENGINEERING

KEYWORDS: SOFTWARE PRODUCT LINE / CODE CLONE DETECTION / COMMON ASSET

TANIT REANTRAGOON : IDENTIFYING COMMON ASSET CANDIDATES IN SOFTWARE PRODUCT LINE BY CODE CLONE DETECTION. ADVISOR : ASSOC. PROF. PORNSIRI MUENCHAISRI, Ph.D., 63 pp.

Software Product Line (SPL) was proposed as an approach to manage a group of related software systematically to reduce cost and effort in software development. One of key activities in SPL is common asset development, which is reusable to build and customize products. Domain experts may analyze and develop common assets from beginning or identify them from existing products. In source-code level, code clone detection is applied for common asset candidates identification. However, if detected code clones contain some differences, differences need to be separated from common parts in order to develop them as common assets. The objective of this research is to set differences apart from detected code clones. By classifying methods that have differences and then applying the “template method” design pattern, common asset candidates are formed as parent class. Then differences codes can be used by overriding method in child classes for individual products.

Department : Computer Engineering Student’s Signature.....

Field of Study : Software Engineering Advisor’s Signature.....

Academic Year : 2011 .

กิตติกรรมประกาศ

ขอขอบพระคุณอาจารย์ที่ปรึกษาวิทยานิพนธ์ รองศาสตราจารย์ ดร.พรศิริ หมื่นไชยศรี ที่ได้กรุณาให้คำปรึกษาแนะแนวทางการวิจัย และให้การสนับสนุนในด้านต่างๆ ในการทำวิทยานิพนธ์ชิ้นนี้จนสำเร็จ

ขอขอบพระคุณ ศาสตราจารย์ ดร.บุญเสริม กิจศิริกุล ผู้ช่วยศาสตราจารย์ นครทิพย์ พร้อมพูล และรองศาสตราจารย์ ดร.วีระ บุญจริง คณะกรรมการสอบวิทยานิพนธ์ ที่ได้กรุณาเสียสละเวลาให้คำแนะนำ และตรวจสอบความถูกต้องสมบูรณ์ของวิทยานิพนธ์ชิ้นนี้

ขอขอบพระคุณคณาจารย์ทุกท่านในภาควิชาวิศวกรรมคอมพิวเตอร์ ที่ได้ให้ความรู้ และคำแนะนำในการเรียนและทำวิจัย

สุดท้ายนี้ ขอกราบขอบพระคุณสมาชิกในครอบครัว และมิตรสหายที่ได้ช่วยเหลือและให้กำลังใจมาโดยตลอด รวมถึงขอขอบพระคุณท่านอื่นที่มิได้กล่าวชื่อไว้ ณ ที่นี้ สำหรับการสนับสนุนให้วิทยานิพนธ์ลุล่วงไปได้ด้วยดี

สารบัญ

หน้า

หน้า

สารบัญตาราง

สารบัญรูป

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

สายผลิตภัณฑ์ซอฟต์แวร์ (Software Product Line) [1] เป็นแนวคิดที่ถูกนำเสนอขึ้นเพื่อสนับสนุนการนำซอฟต์แวร์มาใช้ซ้ำ (Software Reuse) อย่างเป็นระบบโดยจำกัดการสร้างผลิตภัณฑ์ในโดเมน (Domain) เดียวกัน เพื่อสร้างเป็นสินทรัพย์หลัก (Core asset) สำหรับนำไปใช้ซ้ำในการพัฒนาและประกอบเป็นผลิตภัณฑ์ต่อไป โดยการสร้างสินทรัพย์หลักเกิดขึ้นจากการวิเคราะห์หาส่วนร่วม (Commonality) และส่วนแปรผัน (Variability) ในกลุ่มผลิตภัณฑ์ ซึ่งในที่นี้เราได้แยกสินทรัพย์หลักออกเป็นกลุ่มย่อย ซึ่งได้แก่ สินทรัพย์ร่วม (Common asset) และสินทรัพย์แปรผัน (variable asset) ซึ่งเกิดจากส่วนร่วมและส่วนแปรผันในกลุ่มผลิตภัณฑ์ เราจึงสามารถพัฒนาผลิตภัณฑ์โดยการใช้ซ้ำจากสินทรัพย์ร่วม และปรับแต่งให้ตรงกับความต้องการของแต่ละผลิตภัณฑ์ด้วยสินทรัพย์แปรผัน ด้วยแนวคิดนี้เอง สายผลิตภัณฑ์ซอฟต์แวร์จึงช่วยเพิ่มผลิตภาพและคุณภาพ รวมถึงช่วยลดต้นทุน แรงงานที่ใช้ และระยะเวลาการผลิตซอฟต์แวร์ก่อนออกสู่ตลาด

จากแนวคิดดังกล่าว การพัฒนาสินทรัพย์หลัก (Core asset development) จึงเป็นกิจกรรมที่มีความสำคัญในสายผลิตภัณฑ์ซอฟต์แวร์ โดยทิศทางของการพัฒนาสินทรัพย์หลักอาจเป็นไปในแนวทางเชิงรุก (Proactive approach) ในกรณีที่มีการพัฒนาสินทรัพย์หลักขึ้นมาก่อนจึงค่อยนำไปใช้พัฒนาผลิตภัณฑ์ หรือแนวทางเชิงปฏิกิริยา (Reactive approach) ในกรณีที่ผลิตภัณฑ์ถูกพัฒนามาก่อน จึงค่อยสกัดบางส่วนออกมาพัฒนาเป็นสินทรัพย์หลัก อย่างไรก็ตามในแนวทางเชิงปฏิกิริยา การวิเคราะห์ผลิตภัณฑ์ที่มีอยู่ เพื่อพัฒนาเป็นสินทรัพย์ร่วมและสินทรัพย์แปรผันอาจทำได้ยาก เนื่องจากเอกสารการออกแบบมีไม่เพียงพอหรือล้าสมัย ทำให้ต้องใช้ข้อมูลนำเข้าจากซอร์สโค้ดเป็นหลัก ซึ่งใช้เวลาในการทำความเข้าใจและวิเคราะห์ ด้วยเหตุผลดังกล่าวจึงมีการนำเทคนิคการตรวจหาสำเนาโค้ด (Code clone detection)มาใช้ในการระบุส่วนร่วมสำหรับพัฒนาเป็นสินทรัพย์ร่วม [2], [3]

การตรวจหาสำเนาโค้ด คือการตรวจหาชิ้นส่วนโค้ดที่เหมือนหรือคล้ายกัน หรือเรียกว่าสำเนาโค้ด (Code clone) ซึ่งมักพบได้ทั่วไปในการพัฒนาซอฟต์แวร์ [4], [5] ทำให้การตรวจหาสำเนาโค้ดถูกนำไปใช้ในหลายขอบเขตและวัตถุประสงค์ เช่น การลดโค้ดซ้ำซ้อนในระบบ (Duplicated code reduction) การตรวจหาการลอกเลียนวรรณกรรม (Plagiarism Detection) เป็นต้น แม้ว่าสำเนาโค้ดมักจะเกิดจากการทำสำเนาและแปะ (Copy and paste) แต่นักเขียนโปรแกรมมักดัดแปลงสำเนาโค้ดเพื่อนำไปใช้ซ้ำ ทำให้เกิดความแตกต่างระหว่างคู่สำเนาโค้ด ซึ่ง

เป็นอุปสรรคต่อการสร้างสินทรัพย์ร่วมจากสำเนาโค้ดที่ตรวจหาได้ ว่าจะตัดสินใจแยกความแตกต่างเหล่านั้นออกมาอย่างไร

ในบทความ [2] จึงได้ใช้การตรวจหาสำเนาโค้ดเพื่อหาส่วนร่วมระหว่างผลิตภัณฑ์ แล้วนำมาคำนวณค่าความคล้าย โดยให้ฟังก์ชันหน่วยย่อยที่สุดที่ใช้เปรียบเทียบ นิยามของค่าความคล้ายที่ใช้ อิงกับค่าระยะทางเลเวนสไตน์ (Levenshtein distance) จากนั้นจึงสร้างเป็นกราฟความคล้ายเพื่อแสดงความสัมพันธ์ของคู่ฟังก์ชันเหมือนระหว่างผลิตภัณฑ์ และแสดงเป็นตัวเลือกสินทรัพย์ร่วม (Common Asset Candidates) ในที่สุด อย่างไรก็ตามบทความ [2] เสนอเพียงข้อมูลความคล้ายระหว่างผลิตภัณฑ์เพื่อให้ผู้พัฒนาสินทรัพย์ตัดสินใจเลือก ว่าจะนำส่วนใดไปใช้ แต่ไม่ได้กล่าวถึงขั้นตอนหลังจากนั้นว่าจะพัฒนาเป็นสินทรัพย์ร่วมได้อย่างไร ในกรณีที่มีความแตกต่างเกิดขึ้นแทรกอยู่ในส่วนร่วม

งานวิจัยนี้ได้นำเสนอการแก้ปัญหาในการระบุตัวเลือกสินทรัพย์ร่วมในระดับโค้ดดังกล่าว โดยการแยกความแตกต่างออกจากส่วนร่วมที่ตรวจพบ โดยใช้แนวคิดการจำแนกประเภทของสำเนาโค้ดด้วยวิธีการตรวจหาสำเนาโค้ดหลายวิธี [6] เพื่อแยกสำเนาโค้ดที่มีความแตกต่างออกมาในระดับเมท็อด แล้วทำการก่อแบบตัวเลือกสินทรัพย์ โดยใช้แบบอย่างการออกแบบ (Design pattern) ประเภทเมท็อดแม่แบบ (Template method) ทำให้ได้ตัวเลือกสินทรัพย์ร่วมในรูปแบบคลาสแม่ (Parent class) ซึ่งมีการโอเวอร์ไรด์ด้วยโค้ดที่แตกต่างได้ที่คลาสลูก (Child class) สำหรับแต่ละผลิตภัณฑ์

1.2 วัตถุประสงค์ของการวิจัย

งานวิจัยนี้มีวัตถุประสงค์เพื่อออกแบบวิธีการและพัฒนาเครื่องมือระบุตัวเลือกสินทรัพย์ร่วมจากซอร์สโค้ดระหว่างผลิตภัณฑ์ โดยประยุกต์ใช้เทคนิคการตรวจหาสำเนาโค้ดในการระบุส่วนร่วม และใช้แบบอย่างการออกแบบประเภทเมท็อดแม่แบบในการแยกความแตกต่างออกจากสำเนาโค้ดที่ตรวจพบและก่อแบบเป็นตัวเลือกสินทรัพย์ร่วม

1.3 ขอบเขตของการวิจัย

1. ใช้เทคนิคการตรวจหาสำเนาโค้ดในการเปรียบเทียบระหว่างผลิตภัณฑ์เพื่อหาส่วนร่วมสำหรับสายผลิตภัณฑ์ซอฟต์แวร์ ของสองผลิตภัณฑ์
2. สำเนาโค้ดในงานวิจัยนี้ จำกัดขอบเขตที่สนใจเฉพาะสำเนาโค้ดประเภทที่ 3 เท่านั้น
3. ข้อมูลนำเข้าที่ใช้ คือ ซอร์สโค้ดของผลิตภัณฑ์ที่จะนำมาเปรียบเทียบ ในภาษาจาวา (Java)
4. ข้อมูลนำออกที่ได้ คือ ตัวเลือกสินทรัพย์ร่วม ซึ่งถูกรวบรวมไว้เป็นคลาส

5. ทำการประเมินผลวิธีการโดยการนำตัวอย่างซอฟต์แวร์อย่างน้อยหนึ่งโดเมน ซึ่งซอฟต์แวร์ที่เป็นสมาชิกในโดเมนมีอย่างน้อย 2 ซอฟต์แวร์

1.4 ขั้นตอนและวิธีดำเนินการวิจัย

1. ศึกษางานวิจัยที่เกี่ยวข้องกับการสร้างสายผลิตภัณฑ์ซอฟต์แวร์จากผลิตภัณฑ์ที่มีอยู่
2. ศึกษารูปแบบและขั้นตอนของเทคนิคการตรวจหาสำเนาโค้ดที่มีการวิจัยอยู่ในปัจจุบัน
3. ศึกษางานวิจัยที่มีการนำเทคนิคการตรวจหาสำเนาโค้ดมาประยุกต์ใช้กับสายผลิตภัณฑ์ซอฟต์แวร์
4. ศึกษาองค์ความรู้ และเทคนิคอื่นๆ ที่จะมาใช้ร่วม
5. กำหนดรายละเอียดของวิธีการ ออกแบบและพัฒนาเครื่องมือ
6. ทำการทดลองวิธีการกับตัวอย่างซอฟต์แวร์
7. วิเคราะห์ผลลัพธ์ที่ได้จากการทดลอง
8. เขียนบทความงานวิจัย

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. ลดภาระของผู้เชี่ยวชาญในการวิเคราะห์โค้ดของผลิตภัณฑ์เพื่อระบุตัวเลือกสินทรัพย์ร่วม
2. นำเสนอแนวทางจัดการกับความแตกต่างระหว่างชิ้นส่วนโค้ดที่ตรวจหาได้ เพื่อใช้สร้างเป็นตัวเลือกสินทรัพย์ร่วม

1.6 ลำดับการจัดเรียงเนื้อหาในวิทยานิพนธ์

วิทยานิพนธ์นี้แบ่งเนื้อหาออกเป็น 6 บทดังต่อไปนี้ บทที่ 1 คือ บทนำซึ่งกล่าวถึงความเป็นมาและความสำคัญของปัญหา รวมถึงวัตถุประสงค์ของการวิจัย บทที่ 2 กล่าวถึงทฤษฎีพื้นฐานและงานวิจัยที่เกี่ยวข้องในงานวิจัยนี้ บทที่ 3 เป็นการอธิบายแนวคิดและวิธีการวิจัย บทที่ 4 แสดงรายละเอียดการออกแบบเครื่องมือสนับสนุนแนวคิด บทที่ 5 กล่าวถึงการประเมินแนวคิดและเครื่องมือสนับสนุน และบทที่ 6 เป็นส่วนของการสรุปผลการวิจัยและข้อเสนอแนะ

1.7 ผลงานที่ตีพิมพ์จากวิทยานิพนธ์

ส่วนหนึ่งของวิทยานิพนธ์นี้ได้รับการตีพิมพ์เป็นบทความทางวิชาการในหัวข้อเรื่อง “Identifying Core Asset Candidates in Existing Products Source Code” ในงานประชุมวิชาการ “The 2012 4th International Conference on Computer Research and Development (ICCRD 2012)” ณ เฉิงตู ประเทศจีน ระหว่างวันที่ 5-6 พฤษภาคม 2555

1.8 ศัพท์เฉพาะ

โดเมน คือ ขอบเขตเฉพาะขององค์ความรู้ ความชำนาญ หรือกลุ่มของฟังก์ชันการทำงานที่เกี่ยวข้องกัน

สินทรัพย์หลัก คือ เอกสารความต้องการ การออกแบบ ซอร์สโค้ด หรือสิ่งอื่นๆที่เกี่ยวข้องกับการพัฒนาสายผลิตภัณฑ์ซอฟต์แวร์ ซึ่งสามารถนำมาใช้กับผลิตภัณฑ์มากกว่าหนึ่งผลิตภัณฑ์

ส่วนร่วม คือ ส่วนที่เหมือนกันเมื่อเปรียบเทียบกันระหว่างผลิตภัณฑ์

ส่วนแปรผัน คือ ส่วนที่แตกต่างกันออกไป เมื่อเปรียบเทียบระหว่างผลิตภัณฑ์

สินทรัพย์ร่วม คือ สินทรัพย์หลักอย่างหนึ่งซึ่งเกิดจากการวิเคราะห์หาส่วนร่วม เพื่อนำมาใช้ซ้ำ

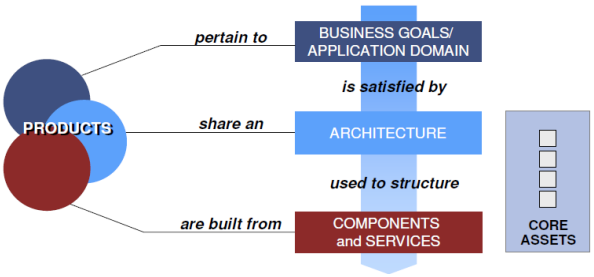
สินทรัพย์แปรผัน คือ สินทรัพย์หลักอย่างหนึ่ง ซึ่งเกิดจากการวิเคราะห์หาส่วนแปรผัน ซึ่งสามารถนำไปใช้สำหรับลักษณะจำเพาะในแต่ละผลิตภัณฑ์

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ทฤษฎีที่เกี่ยวข้อง

2.1.1 สายผลิตภัณฑ์ซอฟต์แวร์ [1]

สายผลิตภัณฑ์ซอฟต์แวร์จัดเป็นแนวคิดหนึ่งในการพัฒนาซอฟต์แวร์ด้วยการใช้ซ้ำอย่างเป็นระบบด้วยการรวมกลุ่มของผลิตภัณฑ์ที่มีความสามารถในการตอบสนองต่อความต้องการในขอบเขตของธุรกิจใดธุรกิจหนึ่ง หรือที่เรียกว่า โดเมน โดยกำหนดให้กลุ่มผลิตภัณฑ์อยู่ในโครงสร้างสถาปัตยกรรมร่วมกันที่เอื้อต่อการใช้ซ้ำร่วมกัน และสร้างเป็นสินทรัพย์หลักเพื่อใช้ในการพัฒนาและประกอบเป็นผลิตภัณฑ์อย่างเป็นระบบตามแผนการผลิตผลิตภัณฑ์ที่ถูกระบุไว้ล่วงหน้า โดยสินทรัพย์หลักมีทั้งส่วนร่วมระหว่างผลิตภัณฑ์ซึ่งสำคัญในแง่ของการใช้ซ้ำ และส่วนแปรผันซึ่งแตกต่างกันไปในแต่ละผลิตภัณฑ์ เมื่อรวมทั้งสองส่วนนี้เข้าด้วยกันจึงสามารถนำไปใช้พัฒนาและประกอบเป็นผลิตภัณฑ์ใหม่ต่อไป รูปที่ 1 แสดงถึงแนวคิดดังกล่าว โดยแสดงความเชื่อมโยงระหว่างผลิตภัณฑ์ซึ่งเกี่ยวข้องในเป้าหมายทางธุรกิจ หรือโดเมนการใช้งานเดียวกัน จึงใช้สถาปัตยกรรมร่วมกัน เพื่อสร้างเป็นส่วนประกอบ และบริการ รวมกันเป็นสินทรัพย์หลักสำหรับสร้างเป็นผลิตภัณฑ์ต่างๆ



รูปที่ 1 แนวคิดหลักของสายผลิตภัณฑ์ซอฟต์แวร์ [1]

รูปที่ 2 กิจกรรมที่จำเป็นในการพัฒนาสายผลิตภัณฑ์ซอฟต์แวร์ [1]

กิจกรรมหลักที่จำเป็นในสายผลิตภัณฑ์ซอฟต์แวร์ประกอบด้วย 3 กิจกรรมหลัก ดังแสดงในรูปที่ 2 ซึ่งได้แก่

1) **กิจกรรมการพัฒนาสินทรัพย์หลัก** (Core Asset Development Activities)

เป็นการสร้างสินทรัพย์หลัก ซึ่งประกอบด้วยส่วนร่วมและส่วนแปรผันสำหรับนำมาประกอบเป็นผลิตภัณฑ์ในภายหลัง อาจเรียกขั้นตอนนี้ได้ว่า วิศวกรรมโดเมน (Domain Engineering) จากปัจจัยหลายอย่างเช่น ข้อจำกัดของผลิตภัณฑ์และการผลิต จะถูกนำมาใช้ทำให้เกิดเป็นผลลัพธ์ ได้แก่ ขอบเขตของสายผลิตภัณฑ์ สินทรัพย์หลัก และแผนการผลิต ซึ่งจะถูกนำไปใช้ในกิจกรรมการพัฒนาผลิตภัณฑ์ต่อไป

2) **กิจกรรมการพัฒนาผลิตภัณฑ์** (Product Development Activities)

เป็นกิจกรรมการพัฒนาผลิตภัณฑ์โดยการนำสินทรัพย์ที่มีอยู่มาประกอบกันเป็นผลิตภัณฑ์ อาจเรียกขั้นตอนนี้ได้ว่า วิศวกรรมเชิงการประยุกต์ใช้ (Application Engineering) ในกิจกรรมนี้จะนำสินทรัพย์ร่วมมาใช้เป็นชิ้นส่วนตั้งต้น แล้วเพิ่มเติมสินทรัพย์เฉพาะเพื่อให้เป็นไปตามความต้องการของแต่ละผลิตภัณฑ์ ทำให้ได้ผลิตภัณฑ์ที่มีลักษณะเฉพาะออกมา

3) **กิจกรรมการจัดการ** (Management Activities)

เป็นกิจกรรมที่ทำการจัดสรรทรัพยากร ประสานงาน และควบคุมทั้งกิจกรรมการพัฒนาสินทรัพย์ร่วม และการพัฒนาผลิตภัณฑ์ ซึ่งจะทำทั้งในเชิงเทคนิค และเชิงองค์กร

แนวทางในการพัฒนาสายผลิตภัณฑ์ อาจจำแนกได้ตามลำดับของการพัฒนาซึ่งได้แก่

- แนวคิดเชิงรุก จะเริ่มต้นจากการพัฒนาสินทรัพย์หลักก่อน เพื่อนำไปใช้พัฒนาเป็นผลิตภัณฑ์ภายหลัง
- แนวคิดเชิงปฏิกิริยา เป็นแนวคิดที่เริ่มต้นจากผลิตภัณฑ์ที่มีอยู่ก่อนแล้ว จึงสกัดหรือพัฒนาสินทรัพย์หลักจากผลิตภัณฑ์เหล่านั้น เพื่อใช้สร้างผลิตภัณฑ์อื่นต่อไป
- แนวคิดส่วนเพิ่ม เป็นแนวคิดที่ผสมผสานระหว่างแนวคิดเชิงรุก และแนวคิดเชิงปฏิกิริยา กล่าวคือทำการพัฒนาสินทรัพย์หลักสลับกับการพัฒนาผลิตภัณฑ์ โดยจะทำอย่างใดก่อนก็ได้ จากนั้นจึงนำสิ่งที่มีอยู่มาพัฒนาต่อยอดต่อไป

ปัจจัยสำคัญต่อความสำเร็จในสายผลิตภัณฑ์ซอฟต์แวร์อีกประการ คือ สถาปัตยกรรมสายผลิตภัณฑ์ ซึ่งมีวิธีการออกแบบทางสถาปัตยกรรมหลายวิธีถูกนำเสนอดังที่บทความสำรวจงานวิจัยได้รวบรวมไว้ [7], [8]

2.1.2 การตรวจหาสำเนาโค้ด

สำเนาโค้ด คือ ชิ้นส่วนซอร์สโค้ดที่ถูกสำเนาเพื่อการใช้ซ้ำและอาจทำการแก้ไขให้เข้ากับการใช้งานใหม่ พบได้ในการพัฒนาซอฟต์แวร์โดยทั่วไปซึ่งนิยมใช้การทำสำเนาและแปะ โดยอาจจะมีการแก้ไขหรือไม่ก็ได้ ซึ่งมักไม่มีการบันทึกไว้ว่าสำเนาโค้ดถูกใช้ไปในซอฟต์แวร์ส่วนใดบ้าง ในกรณีที่สำเนาโค้ดมีข้อผิดพลาดเกิดขึ้น จึงอาจแก้ไขข้อผิดพลาดได้ไม่ครบในทุกๆ สำเนาโค้ด ด้วยเหตุผลดังกล่าว เทคนิคการตรวจหาสำเนาโค้ดจึงถูกนำมาประยุกต์ใช้งานได้หลากหลายด้าน [4], [5]

1) นิยามและประเภทของสำเนาโค้ด

Baxter และคณะ [9] ได้กล่าวไว้ว่า สำเนาโค้ด คือ "ชิ้นส่วนโค้ดที่มีความคล้ายกันในบางนิยามของความคล้าย (Similarity)" ซึ่งอาจกล่าวได้ว่าความคล้ายอาจอยู่ในลักษณะของโครงสร้างวากยสัมพันธ์ หรือคล้ายกันในเชิงความหมาย นอกเหนือจากนิยามดังกล่าวแล้ว Kamiya และคณะ [10] ยังได้นิยามระดับของสำเนาโค้ดเป็นระดับ"เหมือน" (Identical) สำหรับโค้ดที่ถูกสำเนาโดยตรง (Exact copy) และ "คล้าย" สำหรับสำเนาที่ถูกแก้ไขเพียงเล็กน้อย อย่างไรก็ตามนิยามของความคล้ายยังเป็นที่ถกเถียงกันว่า ชิ้นส่วนโค้ดแตกต่างเท่าใดจึงจะยังถือว่าเป็นสำเนาโค้ดอยู่ เพื่อหลีกเลี่ยงความคลุมเครือของนิยามข้างต้น ได้มีการจำแนกประเภทของสำเนาโค้ด ออกเป็น 3 ประเภทในเชิงข้อความ [5], [11] และมีการเพิ่มเติมอีกประเภทในเชิงฟังก์ชัน [4], [11] รายละเอียดของสำเนาโค้ดทั้ง 4 ประเภท ได้แก่

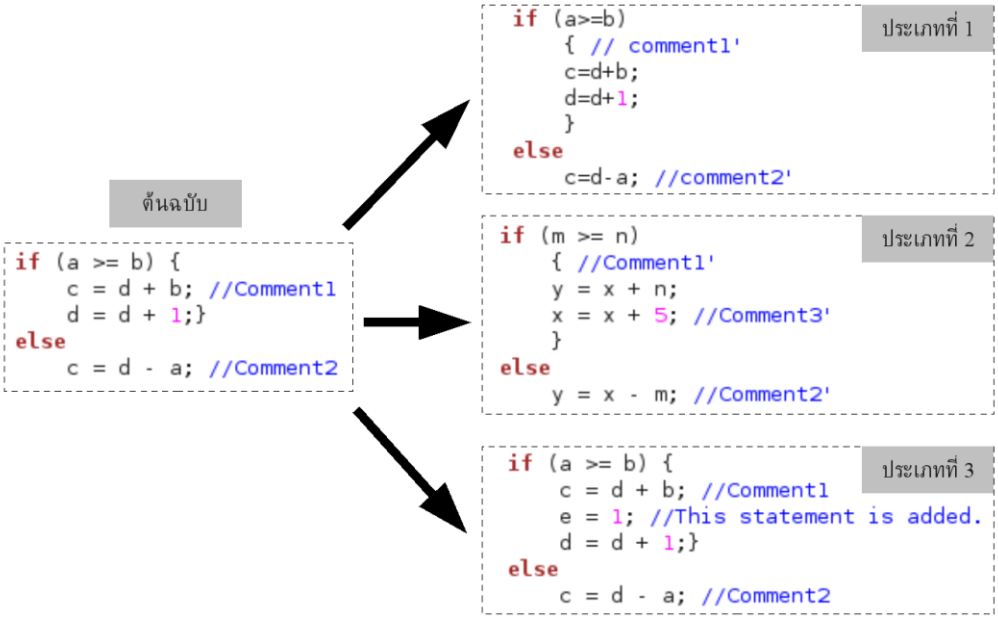
1.1) **ประเภทที่ 1** เป็นชิ้นส่วนโค้ดที่ถูกคัดลอกมาโดยสมบูรณ์ ไม่มีความแตกต่าง ยกเว้นช่องว่าง (Whitespace) และหมายเหตุ (Comment)

1.2) **ประเภทที่ 2** ชิ้นส่วนโค้ดยังมีความคล้ายกันในเชิงวากยสัมพันธ์หรือเชิงโครงสร้าง แต่อาจมีการเปลี่ยนแปลงชื่อ ในส่วนตัวระบุ (Identifier) สัญยพจน์ (Literal) หรือแบบชนิด (Type) ไม่รวมช่องว่าง และหมายเหตุ

1.3) **ประเภทที่ 3** ชิ้นส่วนโค้ดมีการแก้ไข โดยการเพิ่ม ลบ หรือเปลี่ยนแปลงข้อความสั่ง (Statement) ทำให้เกิดความแตกต่างในบรรทัดใดๆ ในชิ้นส่วนโค้ด สำเนาโค้ดชนิดนี้อาจกล่าวได้ว่าเป็นสำเนาประเภทที่ 1 ซึ่งมีความแตกต่างแทรกอยู่ในชิ้นส่วนโค้ด

ตัวอย่างของคู่สำเนาโค้ดประเภทที่ 1 2 และ 3 ถูกแสดงไว้ในรูปที่ 3 โดยชิ้นส่วนโค้ดต้นฉบับอยู่ทางด้านซ้าย และสำเนาโค้ดทั้งสามประเภทถูกแสดงไว้ทางด้านขวา ตามลำดับ

1.4) **ประเภทที่ 4** ชิ้นส่วนโค้ดใช้วิธีการเขียนที่แตกต่างกันโดยสิ้นเชิง แต่ให้พฤติกรรมหรือผลลัพธ์การทำงานที่เหมือนกัน ดังตัวอย่างในรูปที่ 4 ซึ่งเป็นฟังก์ชันหาแฟคตอเรียลซึ่งให้ผลเหมือนกันทั้งสองฟังก์ชัน แต่ (1) ใช้คำสั่ง for (2) ใช้การคำนวณแบบเรียกซ้ำ (Recursive)

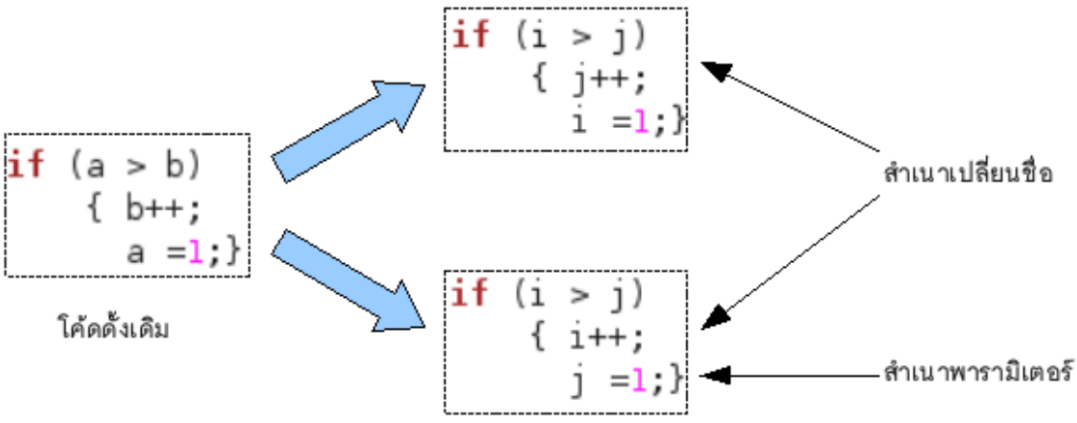


รูปที่ 3 ตัวอย่างคู่สำเนาโค้ดเชิงข้อความ ประเภทที่ 1, 2 และ และ 3 [4]

```
int i, j=1;
    for (i=1; i<=VALUE; i++)
        j=j*i;
```

```
int factorial(int n) {
    if (n == 0) return 1 ;
    else        return n * factorial(n-1) ;
}
```

รูปที่ 4 ตัวอย่างคู่สำเนาโค้ดเชิงฟังก์ชัน (ประเภทที่ 4) [4]



รูปที่ 5 ตัวอย่างโค้ด สำเนาเปลี่ยนชื่อและสำเนาพารามิเตอร์ [4]

นอกเหนือจากการเรียกสำเนาโค้ดตาม 4 ประเภทที่กล่าวข้างต้นแล้ว ในหลายงานวิจัยอาจจะมีการเรียกประเภทของสำเนาโค้ดด้วยถ้อยคำที่แตกต่างออกไป [4] ดังนี้

สำเนาตรงตัว (Exact clone) คือ สำเนาโค้ดที่เหมือนกันอย่างเห็นได้ชัด อาจจะมีข้อแตกต่างบางอย่างในส่วนหมายเหตุ ช่องว่าง และผัง อาจจัดประเภทได้เป็นสำเนาโค้ดประเภทที่ 1

สำเนาเปลี่ยนชื่อ (Renamed clone) เป็นสำเนาโค้ดประเภทที่ 2 กล่าวคือ อาจมีข้อแตกต่างในส่วนชื่อของตัวแปร ค่าสัญพจน์ หมายเหตุ หรือช่องว่าง แต่ไม่สนใจว่าชื่อตัวแปรจะมีการสลับ หรือสอดคล้องกันหรือไม่ ดังแสดงในรูปที่ 5

สำเนาพารามิเตอร์ (Parameterized clone) เป็นสำเนาเปลี่ยนชื่ออย่างหนึ่ง ที่มีชื่อตัวแปรต้องมีความสอดคล้องตรงกัน ดังตัวอย่างโค้ดที่แสดงในรูปที่ 5 ซึ่งเปลี่ยนชื่อตัวแปรจาก a เป็น i และ b เป็น j แต่ไม่มีการเปลี่ยนแปลงในชุดคำสั่ง นั่นคือมีเฉพาะการเปลี่ยนชื่อตัวแปรแต่ไม่มีการสลับตำแหน่ง(สังเกตได้จากตัวแปร i และ j) บางครั้งอาจเรียกสำเนาโค้ดแบบนี้ว่าเป็น p-match

สำเนาเกือบเหมือน (Near-miss clone) เป็นสำเนาโค้ดที่มีความแตกต่างจากต้นฉบับเพียงเล็กน้อย แต่ไม่ถึงกับเป็นสำเนาตรงตัว กล่าวคือมีการเปลี่ยนแปลงที่ไม่ส่งผลให้โครงสร้างทางไวยกรณ์เปลี่ยนไป โดยทั่วไปแล้วสำเนาโค้ดประเภทที่ 2 ถือได้ว่าเป็นสำเนาเกือบเหมือนทั้งหมด อย่างไรก็ตามบางครั้งก็มีการจำแนกว่าการเพิ่ม ลบ หรือแก้ไขข้อความคำสั่งอาจไม่เพียงพอที่จะทำให้เกิดสำเนาโค้ดใหม่ บางครั้งจึงกล่าวได้ว่าสำเนาโค้ดประเภทที่ 3 เป็นสำเนาเกือบเหมือนได้ด้วย

สำเนามีระยะห่าง (Gapped clone) คือสำเนาที่มีการเพิ่ม ลด หรือแก้ไข จนเกิดเป็นระยะห่างขึ้น (Gap) โดยชิ้นส่วนโค้ดนอกเหนือจากระยะห่างดังกล่าวอาจมีการแก้ไขด้วยก็ได้ จากนิยามดังกล่าวจึงจัดได้ว่าสำเนาแบบนี้เป็นสำเนาประเภทที่ 3

สำเนาเชิงโครงสร้าง (Structural clone) จัดเป็นสำเนาที่จำแนกจากความคล้ายในเชิงโครงสร้าง โดยอาจจะมีขอบเขตของสำเนาในหลายระดับไวยกรณ์ เช่น ขอบเขตฟังก์ชัน ขอบเขตข้อความคำสั่ง หรือ ขอบเขตคลาส เป็นต้น

สำเนาฟังก์ชัน (Function clone) จัดเป็นสำเนาเชิงโครงสร้างประเภทหนึ่งซึ่งขอบเขตที่สนใจอยู่ในระดับฟังก์ชัน

สำเนาไม่ติดกัน (Non-contiguous clone) โดยทั่วไปแล้ว สำเนาโค้ดชนิดนี้อาจมองได้ว่าเป็นสำเนามีระยะห่างซึ่งจัดเป็นสำเนาโค้ดประเภทที่ 3 และกิจกรรมที่ใช้ได้กับสำเนามีระยะห่างจึงสามารถนำมาใช้กับสำเนาชนิดนี้ได้ด้วย เพียงแต่บางครั้งชิ้นส่วนสำเนาโค้ดอาจถูกแยกออกจาก

กันในรูปแบบที่แตกต่างออกไป จากรูปที่ 6 จะเห็นได้ว่าเฉพาะ (1) และ (2) เท่านั้นที่เป็นสำเนาไม่ติดกัน

(1)
```
while (isalpha(c) ||
       c== '_' || c== '-') {
  if (p == token_buffer + maxtoken)
      p = grow_token_buffer(p);
  if (c == '-') c = '_';
  *p++ =c;
  c = getc(finput);
  }
```

(2)
```
while (isdigit(c)) {
  if (p == token_buffer + maxtoken)
      p = grow_token_buffer(p);
  numval = numval*20 + c -'0';
  *p++ =c;
  c = getc(finput);
  }
```

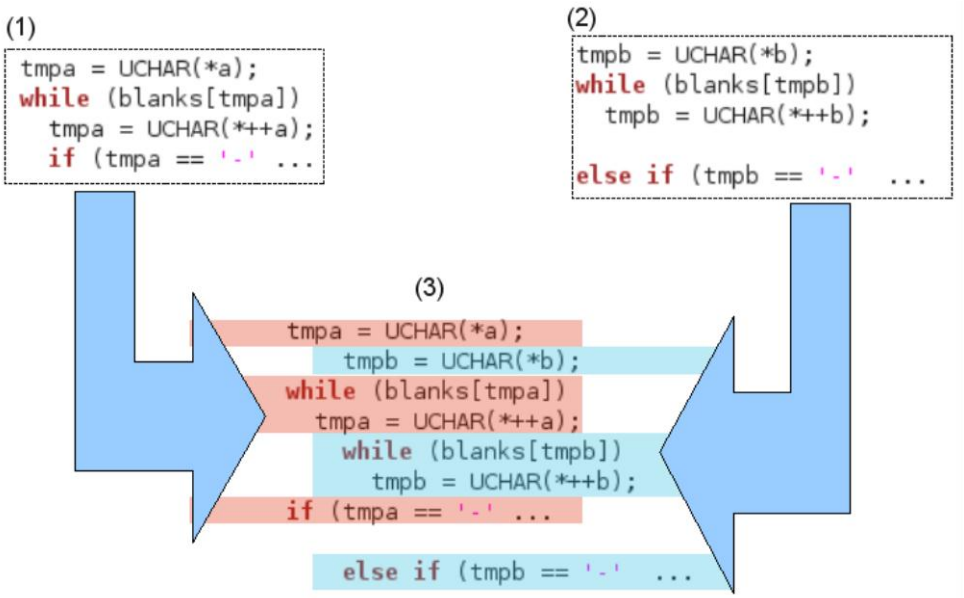
(3)
```
while (c != '>') {
  if (c == EOF) fatal();
  if (c == '\n') {
      warn("unterminated type name");
      ungetc(c, finput);
      break;
  }
  if (p == token_buffer + maxtoken)
      p = grow_token_buffer(p);
  *p++ =c;
  c = getc(finput);
}
```

รูปที่ 6 ตัวอย่างโค้ด สำเนาไม่ติดกัน [4]

สำเนาสลับลำดับ (Reordered clone) เป็นสำเนาโค้ดที่มีการสลับลำดับของข้อความคำสั่งหรือชิ้นส่วนโค้ด โดยที่ความขึ้นต่อกันของการควบคุมหรือข้อมูลไม่มีการเปลี่ยนแปลง ซึ่งสำเนาโค้ดรูปแบบนี้มักถือว่าเป็นสำเนาโค้ดประเภทที่ 4 หรือถ้ามองว่าชิ้นส่วนโค้ดที่ถูกสลับลำดับไปนั้นเป็นระยะห่าง ก็อาจถือได้ว่าเป็นสำเนาโค้ดประเภทที่ 3 ได้เช่นกัน

สำเนาสานต่อกัน (Intertwined clone) เป็นสำเนาโค้ดรูปแบบหนึ่งที่เกิดจากการรวมสองชิ้นส่วนโค้ดสานต่อในลักษณะที่เหลื่อมกัน จัดได้ว่าเป็นสำเนาโค้ดประเภทที่ 4 ดังตัวอย่างในรูปที่ 7 แสดงการรวมชิ้นส่วน (1) และ (2) เข้าเป็น (3)

สำเนาเชิงโครงสร้างในระดับการออกแบบ (Design level structural clone) เป็นสำเนาโค้ดที่ประกอบด้วยสำเนาโค้ดทั่วไปต่างๆ ที่ได้อธิบายไว้ก่อนหน้านี้ไม่ว่าจะเป็นประเภทที่ 1, 2, 3 หรือ 4มารวมกันทำให้เกิดสำเนาที่คล้ายกันของโครงสร้างในระดับการออกแบบ



รูปที่ 7 ตัวอย่างโค้ด สำเนาสานต่อกัน [4]

สำเนาซึ่งพบทั่วไป (Ubiquitous clone) เป็นถ้อยคำที่ใช้เรียกสำเนาโค้ดกลุ่มที่พบบ่อยในซอร์สโค้ดหลายๆ ไฟล์ในระบบเดียวกัน ซึ่งมักจะเป็นสำเนาโค้ดสั้นที่ทำหน้าที่เฉพาะบางอย่างที่คล้ายคลึงกัน เช่น เมธอดที่ใช้ในการเรียกคืนวัตถุ (Getter method) เป็นต้น

2) กระบวนการในการตรวจหาสำเนาโค้ด

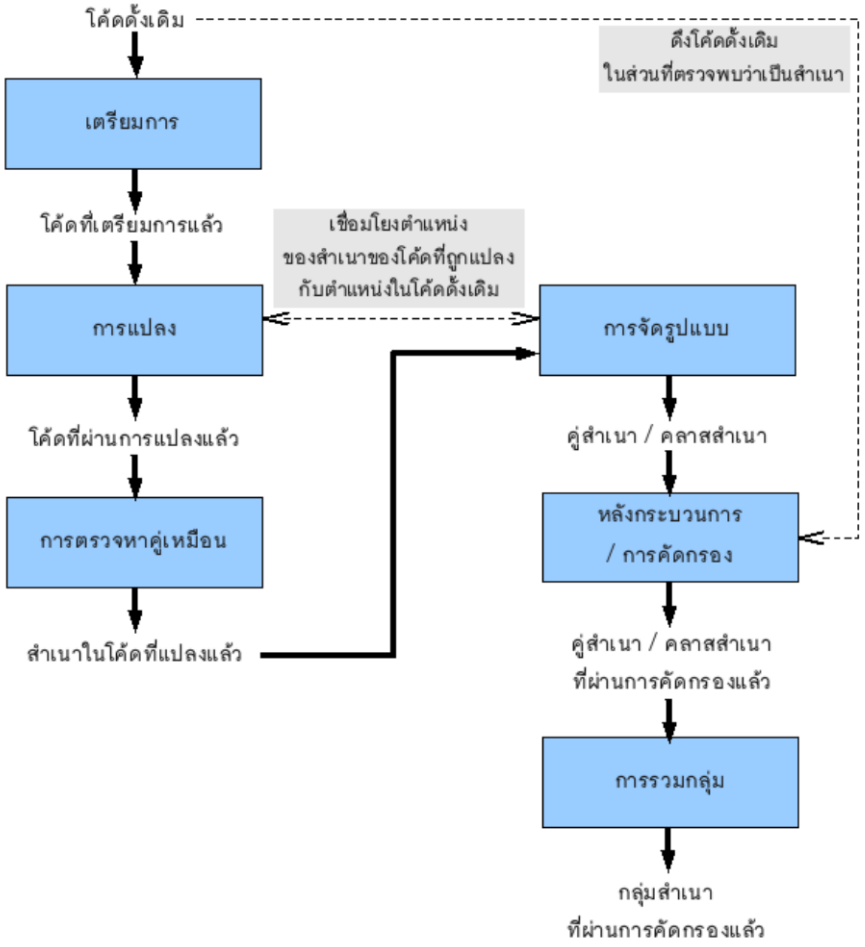
ในการตรวจหาสำเนาโค้ด มักจะทำการแปลงโค้ดจากรูปแบบข้อความให้อยู่ในรูปแบบอื่นก่อน แล้วทำการเปรียบเทียบชิ้นส่วนโค้ดและสรุปผลการตรวจหาสำเนาโค้ด อย่างไรก็ตาม อาจมีการใช้กระบวนการอื่นๆ ระหว่างขั้นตอนเพื่อเพิ่มประสิทธิภาพ และลดขอบเขตการคำนวณ ดังที่ได้จำแนกไว้โดยสังเขปใน [4], [11] ดังแสดงในรูปที่ 8

2.1) เตรียมการ (Preprocessing)

เป็นขั้นตอนที่ทำการแบ่งส่วนของโค้ด และพิจารณาขอบเขตของการเปรียบเทียบ โดยมีวัตถุประสงค์หลัก ได้แก่

- ลบซอร์สโค้ดส่วนที่ไม่สนใจออก เช่น ส่วนของภาษาอื่นที่ฝังตัวอยู่ (ในกรณีที่เทคนิคนั้นๆ ขึ้นกับภาษาที่ตรวจสอบ) หรือโค้ดที่มีแนวโน้มจะทำให้เกิดการตรวจพบสำเนาโค้ดที่ผิดพลาด
- พิจารณาหน่วยของโค้ดที่ต่อเนื่องยาวที่สุด ซึ่งอาจเป็นได้หลากหลายทั้งระดับไฟล์ คลาส ฟังก์ชัน บล็อคคำสั่ง หรือข้อความคำสั่ง

- พิจารณาระดับหน่วยของโค้ดที่ใช้ในการเปรียบเทียบ ซึ่งขึ้นอยู่กับว่าเทคนิคที่ใช้ขึ้นกับระดับของการเปรียบเทียบมากแค่ไหน เช่น ถ้าเป็นเทคนิคที่ใช้ตัววัด (Metric-based comparison) จะสามารถใช้ได้ในทุกระดับโค้ด



รูปที่ 8 กระบวนการทั่วไปในการตรวจหาสำเนาโค้ด [4]

2.2) **การแปลง** (Transformation)

ในขั้นตอนนี้จะทำให้โค้ดเป็นบรรทัดฐานเดียวกันเพื่อกำจัดความแตกต่างที่เกิดจากรูปแบบการเขียนโปรแกรมที่แตกต่างกัน เช่น ผัง หมายเหตุ หรือการเว้นวรรค เป็นต้น จากนั้นในกรณีที่ใช้เทคนิคที่ซับซ้อนมากกว่าการเปรียบเทียบข้อความในโค้ดจะมีการแปลงซอร์สโค้ดให้ได้ข้อมูลในรูปแบบอื่นที่จำเป็นในการเปรียบเทียบด้วย (ดูรายละเอียดเกี่ยวกับประเภทของเทคนิคการตรวจหาสำเนาโค้ดได้ในหัวข้อ2.2.3) ในขั้นตอนการแปลงยังสามารถแบ่งได้เป็นขั้นตอนย่อยได้แก่ การสกัด (Extraction) และการทำให้เป็นบรรทัดฐาน (Normalization)

- การสกัด คือ การแปลงสภาพของซอร์สโค้ดให้อยู่ในสภาพที่เหมาะสมจะนำเข้าไปสู่การเปรียบเทียบตามเทคนิควิธีการที่ใช้ เช่น การทำให้เป็นโทเค็น (Tokenization) สำหรับวิธีที่ใช้โทเค็น การแจงส่วน (Parsing) สำหรับวิธีที่ใช้ต้นไม้วากยสัมพันธ์เชิงนามธรรม (Abstract syntax tree: AST) การวิเคราะห์การไหลของการควบคุมและข้อมูล (Control and data flow analysis) สำหรับวิธีที่ใช้กราฟพึ่งพาของโปรแกรม (Program dependency graph: PDG) หรือการคำนวณตัววัด (Metrics Calculation) สำหรับวิธีที่ใช้ตัววัดเป็นหลัก
- การทำให้เป็นบรรทัดฐาน คือ การกำจัดความแตกต่างในการจัดรูปแบบโค้ด ซึ่งไม่ทำให้พฤติกรรมของโค้ดเปลี่ยนไป เช่น ช่องว่าง หรือหมายเหตุ รวมถึงผัง (Layout) และวากยสัมพันธ์

2.3) การตรวจหาคู่เหมือน (Match detection)

การตรวจหาคู่เหมือนเป็นขั้นตอนที่จะทำการเปรียบเทียบซอร์สโค้ดที่ผ่านขั้นตอนการแปลงมาแล้วระหว่างทุกหน่วยย่อยเพื่อหาสำเนาโค้ดที่มีอยู่ โดยข้อมูลที่ใช้อาจจะเป็นซอร์สโค้ดโดยตรง หรือข้อมูลรูปแบบอื่นที่ได้จากขั้นตอนการแปลง เทคนิคการเปรียบเทียบที่ใช้จะเป็นไปตามความเหมาะสมว่าซอร์สโค้ดที่แปลงมาแล้วอยู่ในรูปแบบไหน ซึ่งมีหลากหลาย เช่น ต้นไม้ซัฟฟิก (Suffix Tree) การจับคู่แบบอย่างเชิงพลวัต (Dynamic Pattern Matching) และการเปรียบเทียบค่าแฮช (Hash-value Comparison) เป็นต้น

สิ่งที่ได้จากขั้นตอนการหาคู่เหมือน คือ คู่สำเนาโค้ด ซึ่งในกรณีที่มีคู่สำเนาโค้ดที่อยู่ติดกันก็จะถูกรวมเป็นสำเนาโค้ดขนาดใหญ่ชิ้นเดียว (Merging) อย่างไรก็ตามคู่สำเนาโค้ดที่ได้จากขั้นตอนนี้จะอยู่ในรูปที่ถูกแปลงจากซอร์สโค้ดดั้งเดิมมาแล้ว เพื่อให้เข้าใจได้ง่ายจึงต้องทำการเชื่อมโยงกับซอร์สโค้ดดั้งเดิมในขั้นตอนการจัดรูปแบบ

2.4) การจัดรูปแบบ (Formatting)

คู่สำเนาโค้ดที่ได้จากการตรวจหาคู่เหมือนอาจจะอยู่ในรูปที่ถูกแปลงไปแล้ว ในขั้นตอนการจัดรูปแบบจะเชื่อมโยงคู่สำเนาเข้ากับซอร์สโค้ดดั้งเดิม ทำให้ได้รายการคู่สำเนา (Clone pair list) ซึ่งระบุถึงตำแหน่งเริ่มต้นและสิ้นสุดของแต่ละสำเนาโค้ดในซอร์สโค้ดดั้งเดิม

2.5) หลังกระบวนการ/การคัดกรอง (Post-processing/Filtering)

เป็นขั้นตอนในการจัดลำดับหรือคัดกรองสำเนาที่ตรวจหามาได้ เพื่อดึงส่วนที่ตรวจผิดพลาดออก อาจจะทำโดยอาศัยผู้ชำนาญ หรือใช้ระบบการแก้ปัญหาจากประสบการณ์แบบอัตโนมัติ (Automated heuristics)

2.6) การรวมกลุ่ม (Aggregation)

เป็นการรวมกลุ่มของข้อมูลการตรวจสอบสำเนา โดยอาจจะรายงานเป็นคู่สำเนา หรือคลาสสำเนา (Clone class) ก็ได้ เพื่อสรุปเป็นรายงานแสดงถึงภาพรวม

จากกระบวนการข้างต้น สำเนาโค้ดที่ตรวจหาได้อาจมีจำนวนมาก ทำให้ยากต่อการทำความเข้าใจ จึงอาจต้องมีการรวมกลุ่มของข้อมูลเพื่อลดปริมาณข้อมูลที่จะนำเสนอหรือนำมาใช้วิเคราะห์ โดยการรวมกลุ่มอาจอยู่ในระดับรายละเอียด เช่น คู่สำเนา (Clone Pairs) หรือจะแสดงอยู่ในระดับภาพรวม เช่น การลงจุดแบบกระจาย (Scatter Plot)

3) เทคนิคการตรวจหาสำเนาโค้ด

เทคนิคที่นำมาใช้ตรวจหาสำเนาโค้ดนั้นมีหลากหลาย ในที่นี้ได้รวบรวมข้อมูลการแบ่งหมวดหมู่ตาม [5] และเพิ่มเติมรายละเอียดเนื้อหาบางส่วนจาก [4] ซึ่งแบ่งประเภทของเทคนิคที่ใช้ได้ ดังนี้

3.1) การเปรียบเทียบเชิงข้อความ (Textual comparison)

เนื่องจากเป็นวิธีที่เปรียบเทียบในเชิงข้อความหรือเชิงคำศัพท์เป็นหลัก และบ่อยครั้งที่โค้ดจะถูกตรวจสอบโดยตรง โดยไม่ได้ผ่านขั้นตอนการแปลงหรือการทำให้เป็นบรรทัดฐาน ทำให้สำเนาที่พบอาจไม่เป็นไปตามโครงสร้างของภาษา

มีการใช้วิธีการที่หลากหลายในการเปรียบเทียบประเภทนี้ เช่น การใช้การเทียบบรรทัดต่อบรรทัดด้วยการจับคู่แบบอย่างพลวัต (Dynamic Pattern Matching) [13] การใช้ค่าลายนิ้วมือ (Fingerprint) [14] หรือใช้เทคนิคดรรชนีความหมายแฝง (Latent Semantic Indexing) จากการค้นคืนข้อมูล (Information Retrieval) [15]

3.2) การเปรียบเทียบโทเค็น (Token comparison)

เป็นวิธีที่ใช้การแปลงโค้ดเป็นลำดับของโทเค็นด้วยตัวแปลศัพท์ (Lexer) แล้วจึงเปรียบเทียบลำดับย่อยที่เหมือนกัน ทำให้วิธีนี้มีความทนทานต่อความเปลี่ยนแปลงในโค้ดมากกว่าการเปรียบเทียบเชิงข้อความ ในแง่ของการจัดรูปแบบ (Formatting) และการเว้นวรรค (Spacing)

ขั้นตอนพื้นฐานของวิธีเปรียบเทียบโทเค็น (อ้างอิงจากงานของ Baker [16]) เริ่มต้นจากแปลงโค้ดเป็นโทเค็นทีละบรรทัดได้เป็นลำดับของโทเค็น แล้วคำนวณเป็นค่าซึ่งเป็นนามธรรมของตัวระบุ (Identifier) และสัญพจน์ (Literal) เรียกว่า ฟังก์เตอร์ (Functor) ส่วนค่าที่เป็นรูปธรรมของตัวระบุและสัญพจน์นั้นจะถูกใช้เป็นพารามิเตอร์ของฟังก์เตอร์ ฟังก์เตอร์และพารามิเตอร์ของมันจะถูกรวมกันเป็นต้นไม้ซัฟฟิก (Suffix Tree) ซึ่งแสดงส่วนต่อท้ายของโปรแกรมทั้งหมดในรูปแบบย่อ จากนั้นจึงทำการเปรียบเทียบสำเนาจากต้นไม้ย่อยของต้นไม้ซัฟฟิก จากขั้นตอน

ข้างต้น ในงานวิจัยอื่นที่ใช้โทเค็น อาจมีความแตกต่างในบางจุด เช่น ในเครื่องมือ CCFinder [10] จะมีการใช้กฎการแปลง (Transformation Rules) เพื่อเพิ่ม ลบ หรือแก้ไขโทเค็นให้เป็นไปในบรรทัดฐานเดียวกัน หรือในเครื่องมือ RTF [17] ใช้แถวลำดับซัฟฟิก (Suffix Array) แทนต้นไม้ซัฟฟิก เพื่อประสิทธิภาพในการจัดการหน่วยความจำ และการแปลงโทเค็นที่ยืดหยุ่นยิ่งขึ้น เป็นต้น

เนื่องจากวากยสัมพันธ์ไม่ได้ถูกนำมาคำนวณด้วย ทำให้สำเนาที่ตรวจพบอาจมีส่วนทับซ้อนในหน่วยวากยสัมพันธ์ ซึ่งทำให้ไม่สามารถแทนที่ได้ในสาระสำคัญระดับฟังก์ชัน แต่อาจแก้ไขข้อด้อยดังกล่าวได้ด้วยวิธีการในขั้นตอนก่อนกระบวนการและหลังกระบวนการ

3.3) การเปรียบเทียบตัววัด (Metric comparison)

เป็นการเปรียบเทียบโดยใช้ตัววัด (Metrics) และเวกเตอร์ตัววัด (Metric Vector) เช่น งานวิจัยของ Merloและคณะ [18], [19], [20], [21] แทนที่จะใช้การเปรียบเทียบด้วยโค้ดโดยตรง หรืออาจจะมีการใช้ค่าระยะทางเพื่อสำหรับเป็นแนวทางจำแนกโค้ดเหมือน เช่น ระยะทางยูคลีเดียน (Euclidean distance) [22], [23]

3.4) การเปรียบเทียบด้วยต้นไม้วากยสัมพันธ์เชิงนามธรรม (Comparison of Abstract Syntax Trees: AST)

แนวคิดหลักของวิธีนี้ คือ การจับคู่ต้นไม้ย่อยในต้นไม้วากยสัมพันธ์เชิงนามธรรม หรือ เอเอสที ซึ่งถูกสร้างขึ้นโดยตัวแจงส่วน (Parser) ซึ่งมักจะมีการใช้ร่วมกับเทคนิคอื่นเพื่อเพิ่มประสิทธิภาพ เช่น การใช้แฮชฟังก์ชัน เพื่อแบ่งต้นไม้ย่อยแล้วเปรียบเทียบกันในกลุ่ม [9] หรือในงานวิจัยของ Koschke และคณะ [24] ได้ใช้แนวคิดการใช้ต้นไม้ซัฟฟิกของ Baker ในการเปรียบเทียบจุดต่อเอเอสที (AST node) [16]

3.5) การเปรียบเทียบด้วยกราฟพึ่งพาของโปรแกรม (Comparison of Program Dependency Graphs: PDG)

ถึงแม้การเปลี่ยนลำดับของข้อความคำสั่งในสำเนาโค้ด หรือใช้สำเนาโค้ดเป็นแผ่นแบบ อาจจะไม่ได้เปลี่ยนแปลงความหมายของโปรแกรมก็ตาม แต่เทคนิคการเปรียบเทียบสำเนาที่ใช้วากยสัมพันธ์หรือโทเค็นเป็นหลัก ไม่สามารถตรวจหาสำเนารูปแบบดังกล่าวได้ กราฟพึ่งพาของโปรแกรม หรือ พีดีจี (PDG) ซึ่งไม่ขึ้นกับลำดับของข้อความคำสั่ง จึงถูกนำมาใช้ตรวจหาสำเนาโดยการหากราฟย่อยสมสัณฐาน (Isomorphism subgraph)

เนื่องจากการใช้กราฟในการแก้ปัญหา จัดเป็นรูปแบบหนึ่งของปัญหาเอ็นพี (NP problem) ซึ่งไม่สามารถวิธีแก้ได้โดยง่าย จึงต้องใช้การประมาณ หรือ ฮิวริสติก (Heuristics) ในการแก้ปัญหา

3.6) **วิธีอื่น ๆ**

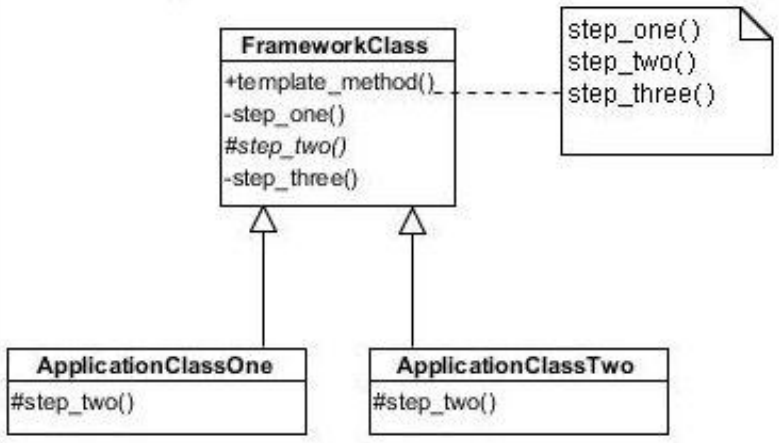
การเปรียบเทียบวิธีอื่นๆ เช่น การผสมผสานเทคนิคเชิงวากยสัมพันธ์และเชิงความหมาย [25] หรือการประยุกต์ใช้ความรู้ด้านเหมืองข้อมูล (Data mining) [26], [27]

2.1.3 เมท็อดแม่แบบ

เมท็อดแม่แบบเป็นรูปแบบการออกแบบอย่างหนึ่งซึ่งใช้กับการออกแบบซอฟต์แวร์เชิงวัตถุ [28] โดยมีจุดประสงค์เพื่อลดโค้ดที่ซ้ำซ้อน ในกรณีที่มีคลาสสองคลาสซึ่งมีบางเมท็อดคล้ายกัน โดยในเมท็อดนั้นๆ มีทั้งโค้ดส่วนที่เหมือนกัน และโค้ดส่วนที่ต่างกัน จึงสามารถใช้เมท็อดแม่แบบเพื่อแยกโค้ดส่วนที่แตกต่างออกไป ทำให้สามารถนำโค้ดอื่นๆ ที่เหมือนกันไปรวมไว้ในคลาสแม่ได้

โครงสร้างของเมท็อดแม่แบบ คือ การแยกเมท็อดที่คล้ายกัน ออกเป็นเมท็อดย่อยๆ รวมกลุ่มเป็นเมท็อดของโค้ดส่วนที่เหมือน และโค้ดส่วนที่ต่าง เพื่อย้ายเมท็อดย่อยเหล่านั้นไปไว้ในคลาสแม่ ยกเว้นเมท็อดที่มีโค้ดที่แตกต่าง หรือที่เรียกว่า เมท็อดฮุค (Hook method) ซึ่งจะเป็นเพียงเมท็อดนามธรรมในคลาสแม่ และโค้ดที่แตกต่างจะถูกย้ายไปไว้ในคลาสลูกแต่ละคลาส เพื่อโอเวอร์ไรด์เมท็อดฮุคที่ถูกประกาศไว้ในคลาสแม่นั่นเอง

ยกตัวอย่างดังรูปที่ 9 ถ้ามีคลาส *ApplicationClassOne* และ *ApplicationClassTwo* ซึ่งมีเมท็อด *template_method()* ที่คล้ายกันโดยมีโค้ดที่ต่างกันอยู่ตรงกลางของเมท็อด เราจึงสามารถแยกขั้นตอนออกเป็นเมท็อดย่อย *step_one(), step_two()* และ *step_three()* ตามลำดับ ซึ่งโค้ดส่วนที่แตกต่างจะถูกแยกอยู่ในเมท็อดย่อย *step_two()* ซึ่งอยู่ในคลาสลูกทั้ง *ApplicationClassOne* และ *ApplicationClassTwo* ส่วนในคลาสแม่ FrameworkClass นั้น จะเก็บเมท็อดย่อย *step_one()* และ *step_three()* ซึ่งมีโค้ดส่วนที่เหมือนกัน และมีการประกาศเมท็อดนามธรรมของ *step_two()* เพื่อให้สามารถโอเวอร์ไรด์ได้จากคลาสลูกแต่ละคลาส



รูปที่ 9 ตัวอย่างเมท็อดแม่แบบ

2.2 งานวิจัยที่เกี่ยวข้อง

2.2.1 Supporting the Grow-and-Prune Model in Software Product Lines Evolution Using Clone Detection [2]

บทความนี้ได้ต่อยอดจากจากแนวคิดในงานวิจัยของ Verhoef และคณะ [3] ที่ได้นำเสนอวิธีจัดการสายผลิตภัณฑ์ซอฟต์แวร์ตามแบบจำลองเติบโตและตัดทิ้ง (Grow-and-Prune model) ซึ่งใช้การตรวจหาสำเนาโค้ดเพื่อระบุส่วนร่วมระหว่างผลิตภัณฑ์ ตามแนวทางการพัฒนาสินทรัพย์หลักเชิงปฏิกิริยา โดยมีจุดประสงค์เพื่อระบุฟังก์ชันที่มีความคล้ายคลึงมากพอที่จะทำให้เป็นสินทรัพย์ ในสายผลิตภัณฑ์ซอฟต์แวร์ โดยอาศัยการใช้เทคนิคการตรวจหาสำเนาโค้ดแบบใช้โทเค็นเป็นหลัก แล้วนำสำเนาโค้ดที่ได้มาใช้ในการคำนวณค่าระยะทางเลเวนสไตน์ (Levenshtein distance) เพื่อแสดงค่าความคล้ายในแต่ละฟังก์ชัน (ระยะทางเลเวนสไตน์ คือ จำนวนครั้งน้อยที่สุดที่จำเป็นในการเปลี่ยนแปลงลำดับของโทเค็นหนึ่ง ให้เหมือนกับอีกลำดับหนึ่ง)

Mende และคณะได้นำมาจากแนวคิดของ Baker [16] ซึ่งเป็นวิธีที่อิงโทเค็นเป็นหลักมาใช้ในบทความนี้ ถึงแม้ว่าวิธีที่อิงกับวากยสัมพันธ์เป็นหลักจะมีความเที่ยงตรง (Precision) มากกว่า โดยให้เหตุผลไว้ว่าวิธีที่อิงโทเค็นเป็นหลัก มีการระลึกได้ (Recall) ที่สูง ซึ่งใช้ได้ดีในกรณีนี้ เนื่องใช้การพิจารณาในระดับฟังก์ชัน การระลึกได้จึงสำคัญมากกว่าความเที่ยงตรง อีกทั้งสำเนาโค้ดถูกใช้เป็นเพียงข้อมูลเบื้องต้นในการวิเคราะห์เท่านั้น จึงสามารถคัดกรองผลการตรวจหาที่ผิดพลาดออกไปได้ เพื่อปรับปรุงความแม่นยำ

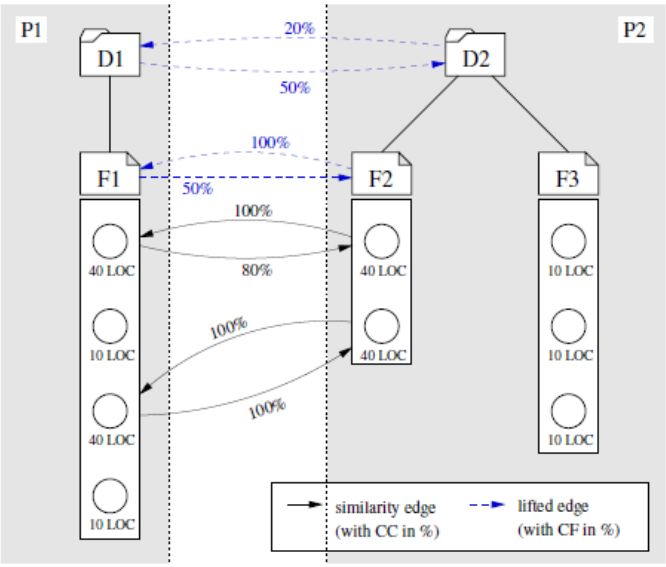
การคำนวณค่าความคล้าย (Similarity) ทำโดยการคำนวณค่าระยะทางเลเวนสไตน์แทนที่จะใช้ค่าไมโทซิสเดลต้า (Mitosis delta) ในงานวิจัยเดิมเนื่องจากไม่มีการอธิบายวิธีการคำนวณในเชิงเทคนิค ซึ่งการคำนวณค่าระยะทางระหว่างแต่ละตัวเลือกที่มีการสำเนาโค้ดอยู่ (Candidate) จะทำให้ได้ค่าความคล้าย $sim(f_x, f_y)$

โดยที่ $$sim(f_x, f_y) = \frac{\max(|sf_x|, |sf_y|) - \Delta f_{x,y}}{|sf_x|}$$

จากการหาค่าความคล้ายจะสามารถบอกความสัมพันธ์ระหว่างสองฟังก์ชันที่ทำการเปรียบเทียบได้ว่า มีโค้ดส่วนที่ใช้ร่วมกันอยู่มากน้อยเพียงใดโดยอาศัยค่า $shared(f_x, f_y)$

โดยที่ $$shared(f_x, f_y) = |sf_x| \cdot sim(f_x, f_y)$$

ซึ่งในขั้นตอนสุดท้าย จะแสดงค่าที่คำนวณมาได้ในรูปแบบกราฟความคล้าย (similarity graph) ในระดับฟังก์ชัน ไฟล์และไดเร็คทอรี ดังรูปที่ 10



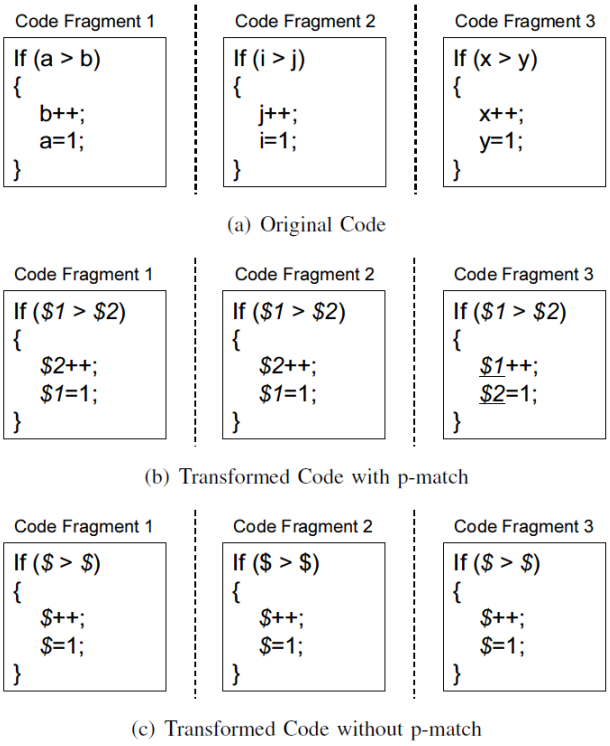
รูปที่ 10 ตัวอย่างกราฟความคล้ายระดับฟังก์ชันระหว่างระบบ P1 และ P2 [2]

กรณีศึกษาที่ใช้ประเมินผลวิธีการของงานวิจัยนี้ มีทั้งหมด 3 กรณีศึกษา ได้แก่ กลุ่มซอฟต์แวร์โอเพ่นซอร์สที่ได้แยกการพัฒนา (Forked) และอีกสองตัวอย่างจากสายผลิตภัณฑ์ในอุตสาหกรรม แง่มุมที่น่าสนใจในกรณีศึกษา เช่น

- การตั้งค่าจำนวนโทเค็นที่สั้นที่สุด หากน้อยเกินไป จะทำให้ขอบเขตการคำนวณเปรียบเทียบเพิ่มขึ้น เป็นผลให้ใช้เวลาในการตรวจหาเพิ่มขึ้น ในขณะที่ถ้ามากเกินไป จะทำให้สำเนาโค้ดที่มีขนาดเล็กกว่าค่าขีดเริ่มที่ตั้งไว้ ไม่ถูกตรวจพบ หรือที่เรียกกันว่า ผลลบเป็นเท็จ (False negative)
- จากกรณีศึกษาการพัฒนาอุปกรณ์ในรถยนต์ พบว่าในบางกรณีค่าเลเวนสไตน์อาจไม่เพียงพอที่จะบ่งชี้ในรายละเอียด คือ มีการตรวจพบสำเนาโค้ดที่แตกต่างเล็กน้อยจนถือว่าคู่สำเนาที่เหมือนกันได้ แต่เครื่องมือกลับรายงานว่าเป็นเพียงสำเนาที่คล้ายกัน
- การใช้เทคนิคแบบโทเค็น มีจุดอ่อนคือ อัตราตรวจพบส่วนที่ไม่ใช่สำเนา (False positive) สูง แต่สามารถใช้ได้ดีในกรณีที่มีการแยกแยะหน่วยย่อยของโปรแกรมในระดับฟังก์ชัน และการคัดกรองโดยใช้ค่าขีดเริ่ม (Threshold) ของค่าความคล้ายที่เหมาะสม

2.2.2 Problematic Code Clones Identification Using Multiple Detection Results [6]

บทความวิจัยนำเสนอวิธีการตรวจหาสำเนาโค้ดโดยสนใจชิ้นส่วนโค้ดที่มีการแก้ไขหลังทำสำเนาจนทำให้เกิดความแตกต่างจากชิ้นส่วนโค้ดดั้งเดิม ซึ่งทำให้เกิดปัญหาได้ในกรณีที่สำเนาโค้ดถูกดัดแปลงเพื่อใช้ซ้ำ แต่ไม่ถูกต้องตามบริบทดั้งเดิมของชิ้นส่วนโค้ดนั้นและนำไปสู่จุดบกพร่อง (Bug) ด้วยเหตุนี้ Yoshiki Higo และคณะได้แบ่งสำเนาโค้ดตามประเภทซึ่งอ้างอิงจากบทความวิจัย แต่ใช้เทคนิคจับคู่อิงตัวแปรเสริม (Parameterized match) เป็นเกณฑ์ในการแบ่งประเภทย่อยเพิ่มเติม



รูปที่ 11 ตัวอย่างการแปลงโค้ดโดยใช้ และไม่ใช้เทคนิคพี-แมตช์ [6]

เทคนิคจับคู่อิงตัวแปรเสริม หรือ พี-แมตช์ (p-match) เป็นเทคนิคที่ทำการแปลงโค้ดก่อนการตรวจหาเพื่อให้สำเนาโค้ดที่มีนัยสำคัญ โดยการแปลงตัวระบุทุกตัวให้เป็นโทเค็นพิเศษสำหรับแต่ละตัวระบุที่เป็นชื่อเดียวกัน ตัวอย่างถูกแสดงดังรูปที่ 11(a) ชิ้นส่วนโค้ดที่ 1, 2 และ 3 มีตัวระบุที่มีชื่อแตกต่างกัน แต่ในชิ้นส่วนโค้ดที่ 1 และ 2 นั้นมีแบบแผนการใช้ตัวระบุที่เหมือนกัน แม้ชื่อตัวระบุจะต่างกัน เมื่อใช้เทคนิคพี-แมตช์ดังรูปที่ 11(b) แล้ว เฉพาะชิ้นส่วนโค้ดที่ 1 และ 2 จึงถูกตรวจพบเป็นสำเนาโค้ด ในขณะที่ถ้าใช้การแทนตัวระบุด้วยโทเค็นตัวเดียวดังรูปที่ 11(c) ชิ้นส่วนโค้ดทั้งสามส่วนดูเหมือนกัน

ประเภทสำเนาโค้ดถูกจำแนกออกเป็น

- ประเภทที่ 1a คือ สำเนาโค้ดประเภทที่ 1 ที่เหมือนกันทุกประการ
- ประเภทที่ 1b คือ สำเนาโค้ดประเภทที่ 1 ที่มีผังของโค้ดแตกต่างกัน
- ประเภทที่ 2a คือ สำเนาโค้ดประเภทที่ 2 ที่มีแบบอย่างการใช้ตัวระบุคงเดิม และถูกตรวจจับได้ถ้าใช้เทคนิคพี-แมตช์
- ประเภทที่ 2b คือ สำเนาโค้ดประเภทที่ 2 ที่มีแบบอย่างการใช้ตัวระบุที่แตกต่างออกไป ซึ่งเทคนิคพี-แมตช์ไม่สามารถตรวจจับได้
- ประเภทที่ 3 คือสำเนาโค้ดที่มีการแทรก ลบ หรือแก้ไขข้อความสั่งบางส่วน

บทความนี้ได้ใช้วิธีการตรวจหาสำเนาโค้ดหลายวิธีการ ซึ่งมีความสามารถในการตรวจพบสำเนาโค้ดในแต่ละประเภทแตกต่างกันไป ตามการปรับแต่งเครื่องมือและเทคนิคที่ใช้ และจากความครอบคลุมของผลลัพธ์ที่ไม่เท่ากันในแต่ละวิธีการนี้เอง ผลลัพธ์สำเนาโค้ดที่ตรวจพบได้จึงถูกจำแนกตามประเภทของสำเนาโค้ด โดยการหักผลลัพธ์ส่วนที่ทับซ้อนระหว่างสองวิธีการที่เหมาะสม ทำให้ได้ส่วนต่างเป็นสำเนาโค้ดประเภทเป้าหมายที่ต้องการ ยกตัวอย่างเช่น ในตารางที่ 1 เราสามารถจำแนกสำเนาโค้ดประเภทที่ 1b ได้จากผลต่างระหว่างผลลัพธ์จากวิธีการตรวจหาแบบ α และ β เป็นต้น

ตารางที่ 1 แบบจำลองวิธีตรวจหาสำเนาโค้ดและผลลัพธ์ที่แตกต่าง [6]

Detection Method	Code Clone Type				
	1a	1b	2a	2b	3
method α	◯	×	×	×	×
method β	◯	◯	×	×	×
method γ	◯	◯	◯	×	×
method δ	◯	◯	◯	◯	×
method ϵ	◯	◯	◯	◯	◯

เพื่อแสดงกรณีศึกษาของกรณีดังกล่าว Yoshiki Higo และคณะจึงได้ทำการตรวจหาสำเนาโค้ดใน Linux kernel โดยใช้เครื่องมือ CCFinderX [10] และ Dude และลดจำนวนสำเนาโค้ดที่ตรวจพบซ้ำในตำแหน่งเดียวกัน หรือเหลื่อมกันบนตำแหน่งเดิม โดยใช้วิธีการคำนวณการทับกันระหว่างคู่สำเนาโค้ด (Overlap) [11]

2.2.3 NICAD: Accurate Detection of Near-miss Intentional Clones Using Flexible Pretty-printing and Code Normalization [29]

นิแค็ด (NiCad) เป็นเครื่องมือตรวจหาสำเนาโค้ดซึ่งใช้เทคนิคการเปรียบเทียบเชิงข้อความทำให้ใช้ทรัพยากรในการตรวจหาที่ค่อนข้างต่ำเมื่อเทียบกับเทคนิคประเภทอื่น โดยใช้ภาษาทีเอ็กซ์

แอล (TXL) [30] เป็นกลไกสำคัญในการแปลงโค้ดเพื่อใช้เปรียบเทียบชิ้นส่วนโค้ด ซึ่งต้องมีการกำหนดไวยกรณ์และกฎการแปลงสำหรับโค้ดที่เป็นเป้าหมายในการตรวจหาสำเนาโค้ด ซึ่งขั้นตอนการทำงานของนิแค็ดสามารถแบ่งออกได้เป็นสามขั้นตอนหลัก ได้แก่

1) การแจงส่วน (Parsing)

ในขั้นตอนนี้โค้ดจะถูกแจงส่วนเพื่อแยกโค้ดออกเป็นชิ้นส่วนย่อยที่สุดในการเปรียบเทียบ เช่น บล็อก หรือฟังก์ชัน รวมถึงการจัดชิ้นส่วนโค้ดให้มีช่องว่าง และรูปแบบผังให้เป็นมาตรฐานเดียวกัน โดยหมายเหตุจะถูกตัดออกไปและไม่นำมาเปรียบเทียบ โดยในขั้นตอนนี้จะมีการกำหนดจำนวนบรรทัดของชิ้นส่วนโค้ดน้อยที่สุดและมากที่สุดที่สนใจเพื่อลดการเปรียบเทียบที่ไม่จำเป็นลงด้วย หลังจากนั้นชิ้นส่วนโค้ดจะถูกเก็บในรูปแบบไฟล์เอ็กซ์เอ็มแอล โดยมีการระบุรายละเอียดของแต่ละชิ้นส่วนโค้ดจากไฟล์ต้นฉบับ เช่น ชื่อไฟล์ จำนวนบรรทัด หมายเลขบรรทัดเริ่มต้นและจบ เป็นต้น ชิ้นส่วนโค้ดเหล่านี้ถูกเรียกว่า ชิ้นส่วนที่อาจเป็นสำเนา (Potential clone)

2) การทำให้เป็นบรรทัดฐาน (Normalization)

คือขั้นตอนทางเลือกที่เพิ่มเติมขึ้นมาเพื่อช่วยให้ผลการตรวจหาสำเนาโค้ดดียิ่งขึ้นด้วยเงื่อนไขเฉพาะ ก่อนทำการเปรียบเทียบระหว่างชิ้นส่วนโค้ดในขั้นตอนต่อไป ซึ่งตัวเลือกในการทำให้เป็นบรรทัดฐานมีหลายอย่าง เช่น

- การเปลี่ยนชื่อ (Renaming) คือ การเปลี่ยนตัวระบุให้เป็นคำเฉพาะเพื่อช่วยปรับปรุงการตรวจหาสำเนาโค้ดที่เกิดจากการเปลี่ยนชื่อตัวระบุ โดยการเปลี่ยนชื่อสามารถแบ่งออกได้เป็นการเปลี่ยนชื่อแบบบอด (Blind renaming) ซึ่งเปลี่ยนตัวระบุเป็นคำเฉพาะเหมือนกันทุกตัว และการเปลี่ยนชื่อแบบสอดคล้องกัน (Consistent renaming) ซึ่งมีการเพิ่มลำดับตัวเลขสำหรับตัวระบุแต่ละตัว หรืออาจเรียกได้ว่าเป็นการทำพีแมตช์นั่นเอง
- การกรอง (Filtering) เป็นการกรองข้อความสั่งประเภทที่ไม่สนใจออกจากชิ้นส่วนที่อาจเป็นสำเนา

นอกเหนือจากตัวเลือกดังกล่าว ผู้ใช้สามารถสร้างวิธีการอื่นๆ เพิ่มโดยการสร้างกฎในภาษาทีเอ็กซ์แอลเพิ่มเติม

3) การเปรียบเทียบ (Comparison)

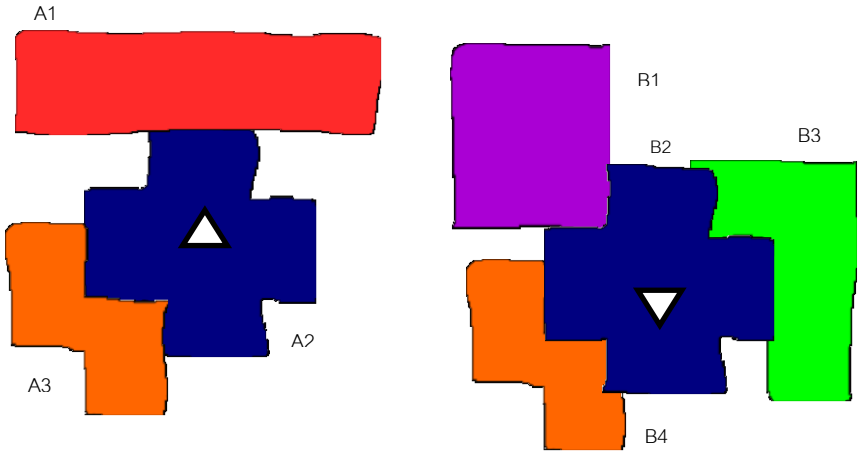
การเปรียบเทียบระหว่างชิ้นส่วนโค้ดในขั้นตอนแรก ทำโดยใช้ขั้นตอนวิธีหาลำดับย่อยร่วมกันที่ยาวที่สุด (Longest Common Subsequence algorithm) โดยกำหนดค่าขีดเริ่ม ซึ่งเป็นความแตกต่างมากที่สุดที่ให้ถือว่าชิ้นส่วนโค้ดที่นำมาเปรียบเทียบเป็นสำเนาโค้ด นอกจากนี้แล้วค่า

ขีดเริ่มยังมีส่วนช่วยลดภาระในการเปรียบเทียบ กล่าวคือชิ้นส่วนที่อาจเป็นสำเนาจะถูกแบ่งกลุ่มกัน (Clustering) ตามขนาดที่ใกล้เคียงกันในขอบเขตค่าขีดเริ่มที่กำหนด และถูกเปรียบเทียบเฉพาะในกลุ่มที่ถูกแบ่งไว้ เช่น ถ้ามีชิ้นส่วนที่อาจเป็นสำเนาขนาด 100 บรรทัดและค่าขีดเริ่มคือ 10% ชิ้นส่วนโค้ดนี้จะถูกรวมกลุ่มและเปรียบเทียบเฉพาะกับชิ้นส่วนโค้ดที่มีขนาดระหว่าง 90 และ 110 บรรทัด และชิ้นส่วนโค้ดขนาดอื่นๆ ก็จะถูกแยกกลุ่มออกไปและเปรียบเทียบเฉพาะในกลุ่ม ในลักษณะเดียวกัน

บทที่ 3
การออกแบบวิธีการระบุตัวเลือกสินทรัพย์ร่วมระหว่างผลิตภัณฑ์ในระดับโค้ด

การพัฒนาสินทรัพย์หลักถือเป็นกิจกรรมหลักสำหรับการพัฒนาสายผลิตภัณฑ์ซอฟต์แวร์ ซึ่งสินทรัพย์หลักนั้นประกอบด้วยส่วนร่วมซึ่งเป็นส่วนที่เหมือนกันระหว่างผลิตภัณฑ์ และส่วนแปรผันซึ่งแตกต่างกันออกไปในแต่ละผลิตภัณฑ์ การใช้ซ้ำจึงขึ้นอยู่กับการวิเคราะห์หาส่วนร่วมเป็นหลัก โดยเฉพาะอย่างยิ่งในแนวทางปฏิกิริยา ซึ่งใช้การระบุส่วนร่วมระหว่างผลิตภัณฑ์ที่มีอยู่แล้วเพื่อพัฒนาเป็นสินทรัพย์ร่วม และถึงแม้ว่าจะมีงานวิจัยในหัวข้อดังกล่าวมากมายก็ตาม แต่ส่วนใหญ่มักจะมุ่งเน้นไปที่ระดับกระบวนการ และสถาปัตยกรรม โดยละรายละเอียดในระดับโค้ดเอาไว้

ในบทความ [2] ได้นำเสนอการใช้เทคนิคตรวจหาสำเนาโค้ดในการระบุส่วนร่วมระหว่างผลิตภัณฑ์ซึ่งใช้โค้ดเป็นข้อมูลนำเข้าหลัก และใช้ระยะห่างเลเวนสไตน์ในการคำนวณค่าความคล้ายระหว่างฟังก์ชันเพื่อเป็นข้อมูลในการตัดสินใจสำหรับผู้พัฒนาสินทรัพย์ว่าควรจะเลือกส่วนใดมาสกัดเป็นสินทรัพย์หลัก



(ก) ผลิตภัณฑ์ A (ข) ผลิตภัณฑ์ B

รูปที่ 12 แนวความคิดของปัญหางานวิจัย ด้วยตัวอย่าง (ก)ผลิตภัณฑ์ A และ(ข)ผลิตภัณฑ์ B

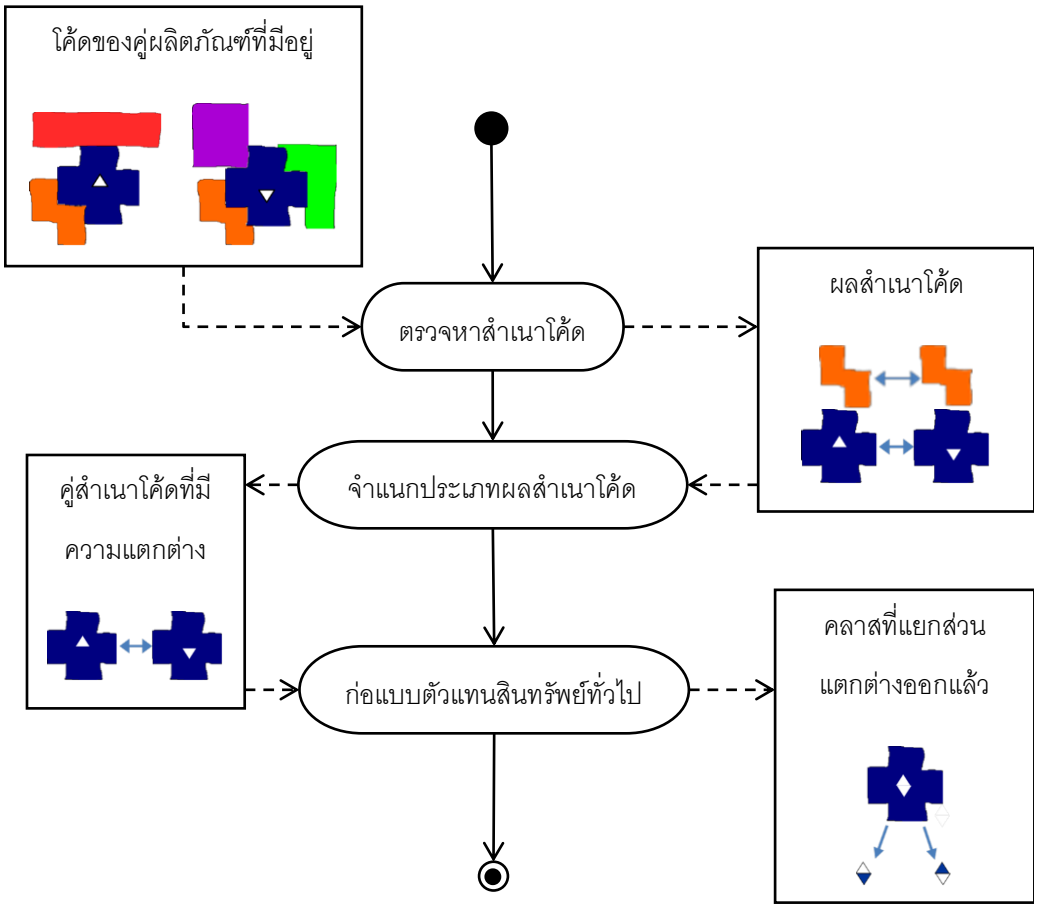
อย่างไรก็ตาม ค่าความคล้ายเป็นเพียงข้อมูลเพื่อตัดสินใจเบื้องต้นเท่านั้น การระบุตัวเลือกสินทรัพย์ร่วมจำเป็นต้องจัดการกับความแตกต่างที่พบในส่วนร่วม เพื่ออธิบายถึงแนวความคิดของปัญหา ในที่นี้ได้ยกตัวอย่างโดยสมมุติให้มีผลิตภัณฑ์ A ซึ่งประกอบด้วยชิ้นส่วน A1 A2 และA3 ดังแสดงในรูปที่ 12(ก) และผลิตภัณฑ์ B ซึ่งมีชิ้นส่วน B1 B2 B3และ B4 ดังแสดงในรูปที่ 12(ข) จะ

เห็นได้ว่ามีคู่สำเนาได้แก่ ชิ้นส่วน A2 กับ B2 และ A3 กับ B4 ซึ่งคล้ายกัน แต่มีความแตกต่างอยู่บ้างในคู่ A2 กับ B2 ซึ่งจำเป็นต้องแยกความแตกต่างนั้นออกเพื่อพัฒนาต่อเป็นสินทรัพย์ร่วม

งานวิจัยนี้มีจุดประสงค์เพื่อแก้ปัญหาดังกล่าว โดยแยกส่วนที่แตกต่างออกด้วยกลไกการโอเวอร์ไรด์ของแบบรูปการออกแบบเมท็อดแม่แบบ ซึ่งวิธีการระบุส่วนร่วมระหว่างผลิตภัณฑ์ด้วยการใช้เทคนิคการตรวจหาสำเนาโค้ดในระดับเมท็อด แล้วจึงจำแนกประเภทสำเนาโค้ดเพื่อแยกผลลัพธ์ที่ส่วนมีแตกต่างออกมา ซึ่งชิ้นส่วนโค้ดทั้งส่วนที่เหมือนกันและแตกต่างจะถูกแยกไว้ในเป็นเมท็อด ซึ่งเป็นหน่วยย่อยที่สุดที่จะสามารถนำไปใช้ซ้ำอย่างเป็นระบบได้ในสายผลิตภัณฑ์ซอฟต์แวร์ และสามารถรวมเป็นระดับที่ใหญ่ขึ้นได้ต่อไป

จากผลสำเนาโค้ดที่สามารถตรวจพบได้ ซึ่งได้แก่ สำเนาโค้ดประเภทที่ 1, 2 และ 3 งานวิจัยนี้มุ่งเน้นที่การจัดการกับสำเนาโค้ดที่มีความแตกต่าง ซึ่งสำเนาโค้ดประเภทที่ 1 นั้นไม่มีความแตกต่างใด ยกเว้นช่องว่างและหมายเหตุ และสำเนาโค้ดประเภทที่ 2 มีความแตกต่างกันเพียงชื่อตัวระบุ ซึ่งสามารถใช้การเปลี่ยนชื่อกลับมาให้เหมือนกันได้ ดังนั้นการจัดการกับสำเนาโค้ดประเภทที่ 1 และ 2 จึงถูกละไว้ และให้ความสำคัญกับสำเนาโค้ดประเภทที่ 3 ซึ่งมีการแก้ไขในบางข้อความสั่งของสำเนาโค้ดเป็นหลัก ส่วนสำเนาโค้ดประเภทที่ 4 นั้นไม่ถูกยกมารวมไว้ ณ ที่นี้ เนื่องจากเทคนิคการตรวจหาสำเนาโค้ดโดยส่วนใหญ่อาศัยความสัมพันธ์ของโครงสร้าง หรือวากยสัมพันธ์ของโค้ดในการตรวจหา จึงไม่สามารถตรวจหาสำเนาโค้ดในนิยามเชิงพฤติกรรมได้

ภาพรวมขั้นตอนการแก้ปัญหาของงานวิจัยนี้ ถูกแบ่งเป็น 3 ขั้นตอน ได้แก่ การตรวจหาสำเนาโค้ด การจำแนกประเภทผลสำเนาโค้ด และการก่อแบบตัวเลือกสินทรัพย์ร่วม ดังแสดงตามแผนภาพกิจกรรมพร้อมตัวอย่างในรูปที่ 13 เริ่มต้นจากการตรวจหาสำเนาโค้ดระหว่างคู่ผลิตภัณฑ์ที่มีอยู่ แล้วนำผลสำเนาโค้ดที่ได้มาจำแนกประเภทเพื่อแยกคู่สำเนาโค้ดที่มีความแตกต่าง จากนั้นจึงก่อแบบตัวเลือกสินทรัพย์ร่วมในรูปแบบของคลาสซึ่งประกอบด้วยส่วนร่วมที่สามารถปรับแต่งตามความแตกต่างของแต่ละผลิตภัณฑ์ รายละเอียดของทั้งสามขั้นตอนถูกอธิบายไว้ในหัวข้อ 3.1 3.2 และ 3.3 ตามลำดับ



รูปที่ 13 ภาพรวมขั้นตอนการแก้ปัญหา

3.1 การตรวจหาสำเนาโค้ด

งานวิจัยนี้ได้ใช้เทคนิคการตรวจหาสำเนาโค้ดเพื่อค้นหาส่วนร่วมในโค้ดระหว่างผลิตภัณฑ์ที่มีอยู่ โดยจำกัดขอบเขตของประเภทของสำเนาโค้ดที่ตรวจหา เฉพาะสำเนาโค้ดในประเภทที่ 1 2 และ 3 ซึ่งนิยามได้ชัดเจนในเชิงวากยสัมพันธ์ และเทคนิคการตรวจหาสำเนาโค้ดส่วนใหญ่สามารถตรวจหาได้ดี [11], [12]

การตัดสินใจเลือกเทคนิคและเครื่องมือที่นำมาใช้ในงานวิจัยนี้ ได้ใช้ข้อมูลที่รวบรวมจาก [12] ซึ่งได้จำลองสถานการณ์การสำเนาโค้ดด้วยชิ้นส่วนโค้ดที่มีการแก้ไขในรูปแบบต่างๆ ตามลักษณะสำเนาโค้ดประเภทที่ 1, 2, 3 และ 4 ผู้วิจัยได้พิจารณาความสามารถของแต่ละเครื่องมือในการตรวจสำเนาโค้ดในสามประเภทแรก ร่วมกับข้อจำกัดเรื่องใบอนุญาต และการเข้าถึงของเครื่องมือ ทำให้ได้ตัวเลือกซึ่งได้แก่นิแค็ด [29] CCFinderX [10] และ Deckard [31] อย่างไรก็ตามผู้วิจัยได้ตัดสินใจใช้เครื่องมือเพียงเครื่องมือเดียว เพื่อหลีกเลี่ยงความซับซ้อนอันเกิดจากการ

นิยามสำเนาโค้ดที่แตกต่างกันในแต่ละเทคนิคซึ่งส่งผลต่อการจำแนกประเภทสำเนาโค้ดในขั้นตอนถัดไป และพบว่านิแค็ดเหมาะสมที่สุดด้วยเหตุผลดังนี้

- จากผลการตรวจหาสำเนาโค้ดในสถานการณ์จำลองซึ่งมีตัวอย่างสำเนาโค้ดแต่ละประเภท [12] พบว่านิแค็ดมีความสามารถในการตรวจหาสำเนาโค้ดได้ดีทั้งประเภทที่ 1, 2 และ 3 เมื่อเทียบกับเครื่องมือ CCFinderX ซึ่งสามารถตรวจจับได้ดีเฉพาะประเภทที่ 1 และ 2
- ใช้หลักการเปรียบเทียบเชิงข้อความซึ่งใช้ทรัพยากรต่ำ
- จากการศึกษาหลักการทำงานของเครื่องมือ พบว่าสามารถปรับแต่งพารามิเตอร์เพื่อให้ได้ผลการตรวจหาที่แตกต่างตามประเภทของสำเนาโค้ด โดยใช้นิแค็ดเพียงเครื่องมือเดียว
- ข้อมูลนำออกเป็นชิ้นส่วนโค้ดซึ่งอยู่ในรูปแบบของไฟล์เอ็กซ์เอ็มแอล จึงนำไปใช้ได้โดยตรง เมื่อเทียบกับ CCFinderX ซึ่งอ้างอิงเป็นโทเค็นทำให้ต้องเชื่อมโยงกลับมาเป็นโค้ดอีก
- มีประเภทใบอนุญาตเป็นโอเพนซอร์ส และโครงการมีความเคลื่อนไหวสม่ำเสมอ

หน่วยย่อยที่สุดในการตรวจหาสำเนาโค้ดระหว่างผลิตภัณฑ์ ถูกกำหนดไว้ในระดับเมท็อด เนื่องจากเป็นหน่วยที่เล็กที่สุดรองจากระดับคลาสที่สามารถนำไปใช้ซ้ำได้ ส่วนการปรับแต่งเครื่องมือซึ่งมีผลสืบเนื่องต่อการจำแนกประเภทผลสำเนาโค้ดนั้น ถูกอธิบายไว้ในหัวข้อ 3.2

3.2 การจำแนกประเภทผลสำเนาโค้ด

ในการตรวจหาสำเนาโค้ดโดยทั่วไปนั้น เป็นการหาและเปรียบเทียบชิ้นส่วนโค้ดโดยสนใจว่าชิ้นส่วนโค้ดเหล่านั้นคล้ายกันมากน้อยแค่ไหน โดยละเว้นว่าชิ้นส่วนโค้ดมีความแตกต่างตรงไหนอย่างไร แต่ในการสกัดสำเนาโค้ดเพื่อสร้างเป็นตัวเลือกสินทรัพย์ร่วม ความแตกต่างถือเป็นอุปสรรค จำเป็นต้องระบุตำแหน่งของความแตกต่างระหว่างชิ้นส่วนโค้ด งานวิจัยนี้ใช้เกณฑ์การจำแนกประเภทของสำเนาโค้ดเป็น 4 ประเภท โดยมุ่งเน้นไปที่สำเนาโค้ดประเภทที่ 3 ซึ่งมีความแตกต่างเกิดขึ้นอย่างชัดเจน จากการเพิ่ม ลบ หรือแก้ไขคำสั่ง วิธีการคัดแยกผลลัพธ์การตรวจหาสำเนาโค้ดได้ใช้แนวทางตามบทความวิจัย [6] ซึ่งใช้การตรวจหาสำเนาโค้ดหลายวิธีซึ่งมีความสามารถในการตรวจหาสำเนาโค้ดแต่ละประเภทแตกต่างกัน ทำให้สามารถแยกประเภทสำเนาโค้ดได้จากการหักผลลัพธ์ส่วนที่ทับซ้อนกันเพื่อให้ได้ประเภทที่ต้องการ ด้วยวิธีดังกล่าวการ

จำแนกประเภทสำเนาโค้ดจึงสามารถกระทำได้โดยไม่กระทบถึงการทำงานภายในเทคนิคตรวจหาสำเนาโค้ด

อย่างไรก็ตามการใช้วิธีจำแนกประเภทสำเนาโค้ดด้วยวิธีดังกล่าว จำเป็นต้องมีการคัดเลือกเทคนิคการตรวจหาสำเนาโค้ดที่จะใช้เพื่อให้เกิดผลลัพธ์ที่แตกต่างและได้ประเภทของสำเนาโค้ดที่สนใจ ซึ่งในที่นี้ได้ใช้เพียงเทคนิคการเปรียบเทียบเชิงข้อความของเครื่องมือนิแค็ดเพียงอย่างเดียว เพื่อหลีกเลี่ยงผลการตรวจหาสำเนาโค้ดที่ผิดพลาด ซึ่งเกิดจากนิยามของสำเนาโค้ดที่อาจแตกต่างกันในแต่ละเทคนิค

การนิยามวิธีการตรวจหาสำเนาโค้ดโดยใช้เพียงเครื่องมือนิแค็ดนั้น ได้ใช้การปรับค่าพารามิเตอร์ ซึ่งได้แก่ ค่าขีดเริ่ม และการเปลี่ยนชื่อ ในการปรับแต่งให้ได้ผลลัพธ์ซึ่งตรวจพบได้สำเนาโค้ดตามประเภทที่ต้องการ ดังแสดงในตารางที่ 2 และเนื่องจากเทคนิคที่ใช้ไม่สามารถตรวจหาสำเนาโค้ดประเภทที่ 4 ได้ ดังอธิบายไว้ตอนต้นบทแล้วนั้น วิธีการตรวจหาสำเนาโค้ดที่แสดงในตารางที่ 2 จึงครอบคลุมเฉพาะสำเนาโค้ดเป็นประเภทที่ 1, 2 และ 3

วิธีการตรวจหาสำเนา A พารามิเตอร์ถูกปรับให้เคร่งครัดที่สุด โดยไม่ใช้การทำให้เป็นบรรทัดฐานใดๆ และกำหนดค่าขีดเริ่มเป็นศูนย์ ซึ่งทำให้ผลลัพธ์ที่ได้มีเฉพาะสำเนาตรงตัวเท่านั้น ส่วนวิธีการตรวจหาสำเนา B กำหนดพารามิเตอร์เพิ่มเติมจากวิธีการตรวจหาสำเนา A โดยใช้การเปลี่ยนชื่อ เพื่อให้ชิ้นส่วนโค้ดที่มีการแก้ไขชื่อตัวระบุ แต่ไม่มีการแก้ไขอื่นๆ ถูกตรวจพบได้ และวิธีการตรวจหา C ใช้ค่าขีดเริ่มเพื่อตรวจหาสำเนาโค้ดที่มีความแตกต่างได้ โดยยอมรับความแตกต่างที่ไม่เกินจากค่าขีดเริ่มที่กำหนดไว้

ตารางที่ 2 กิจกรรมในกระบวนการบูรณาการผลิตภัณฑ์จากมาตรฐานและแบบจำลองอ้างอิง

วิธีการตรวจหา	ชุดพารามิเตอร์		ประเภทสำเนาโค้ดที่ตรวจพบ		
	ค่าขีดเริ่ม	*การเปลี่ยนชื่อ*	*Type-1*	*Type-2*	*Type-3*
A	0.0		√		
B	0.0	√	√	√	
C	มากกว่า 0.0		√	√	√

จากตารางที่ 2 ถ้าให้ S_A, S_B และ S_C เป็นเซ็ตของสำเนาโค้ดที่เกิดจากวิธีการตรวจหา A, B และ C ตามลำดับ และ S_1, S_2 และ S_3 เป็นเซ็ตของสำเนาโค้ดประเภทที่ 1, 2 และ 3 ตามลำดับแล้ว การจำแนกประเภทสำเนาโค้ดทำได้ดังสมการ

$S_1 = S_A$

$S_2 = S_B - S_A$

$S_3 = S_C - S_B$

```
<clones>
<systeminfo  system="jabref-1.8" granularity="functions" threshold="0%" minlines="5"
maxlines="500"/>
<classinfo npcs="5996" nclones="1020" nfragments="2006" npairs="1062" nclasses="986"/>
<runinfo ncompares="117660" cputime="140000"/>

<class id="1" nlines="180" nfragments="2">
<source file="Jabrefproducts/jabref-2.0/src/java/net/sf/jabref/imports/CsaImporter.java"
startline="197" endline="449" pcid="3245">
</source>
<source file="Jabrefproducts/jabref-1.8/src/java/net/sf/jabref/imports/CsaImporter.java"
startline="189" endline="441" pcid="552">
</source>
</class>
…
</clones>
```

รูปที่ 14 ตัวอย่างผลสำเนาโค้ดในรูปแบบภาษาเอ็กซ์เอ็มแอล

จากผลการตรวจหาสำเนาโค้ดที่อธิบายไว้ในหัวข้อ 3.1 หน่วยย่อยที่ทำการตรวจหาเป็นระดับของเมท็อด ซึ่งข้อมูลนำออกถูกจัดเรียงในแบบคล้ายภาษาเอ็กซ์เอ็มแอล และมีโครงสร้างแบ่งออกเป็นกลุ่มสำเนาโค้ดของฟังก์ชันที่คล้ายกันดังตัวอย่างในรูปที่ 14 โดยข้อมูลในแต่ละป้ายระบุ ได้แก่

- *clones* เป็นป้ายระบุที่เป็นรากของไฟล์ผลลัพธ์ ประกอบด้วยป้ายระบุ *systeminfo*, *classinfo*, *runinfo* และ *class*
- *systeminfo* เป็นป้ายระบุที่บ่งบอกถึงรายละเอียดของเป้าหมายและพารามิเตอร์ที่ใช้ ได้แก่ ชื่อโฟลเดอร์เป้าหมาย ความละเอียดที่ใช้ตรวจหา ค่าขีดเริ่ม จำนวนบรรทัดน้อยที่สุด และมากที่สุดของเมท็อดที่ทำการตรวจหาสำเนา

- *classinfo* เป็นป้ายระบุที่บ่งบอกถึงจำนวนของชิ้นส่วนโค้ดและสำเนาโค้ดต่างๆ ซึ่งได้แก่ จำนวนชิ้นส่วนที่อาจเป็นสำเนา จำนวนชิ้นส่วนที่เป็นสำเนาโค้ด จำนวนคู่สำเนาโค้ดที่พบ จำนวนกลุ่มสำเนาโค้ดในกรณีที่คล้ายกันมากกว่าสองชิ้นส่วนโค้ด
- *runinfo* เป็นป้ายระบุที่มีข้อมูลในการเปรียบเทียบหาสำเนาโค้ด ซึ่งได้แก่ จำนวนครั้งการเปรียบเทียบทั้งหมดที่ใช้ และระยะเวลาในการค้นหา
- *class* เป็นป้ายระบุที่รวมกลุ่มของสำเนาโค้ดไว้ แยกเป็นชิ้นส่วนโค้ดในป้ายระบุ *source* และมีข้อมูลอื่นๆ ได้แก่ หมายเลขลำดับของกลุ่มสำเนา จำนวนบรรทัดของชิ้นส่วนโค้ดในกลุ่ม และจำนวนชิ้นส่วนโค้ดในกลุ่ม
- *source* เป็นป้ายระบุที่เก็บข้อมูลของชิ้นส่วนโค้ดที่เป็นสำเนาไว้ ได้แก่ ชื่อไฟล์ที่ชิ้นส่วนโค้ดอยู่ บรรทัดเริ่มต้นและสิ้นสุด หมายเลขลำดับของชิ้นส่วนโค้ด และชิ้นส่วนโค้ด(ในกรณีที่ให้ออกรายงานรวมชิ้นส่วนโค้ดด้วย) โดยป้ายระบุ *source* จะถูกบรรจุอยู่ในป้ายระบุ *class* ตามกลุ่มสำเนาที่ชิ้นส่วนโค้ดนั้นอยู่

จากผลลัพธ์ดังที่ได้อธิบายข้างต้น ข้อมูลจะถูกปรับให้อยู่ในรูปแบบที่ดี (Well-formed) ตามข้อกำหนดของภาษาเอ็กซ์เอ็มแอลเพื่อใช้เป็นข้อมูลนำเข้าสำหรับขั้นตอนต่อไป จากนั้นข้อมูลในป้ายระบุ class และ source ของแต่ละวิธีการตรวจหา จะถูกนำมาเปรียบเทียบและหักส่วนที่ซ้ำกันออกตามสมการของ S_1, S_2 และ S_3 ที่แสดงไว้ข้างต้น ทำให้ได้ชิ้นส่วนโค้ดที่แบ่งออกตามประเภทสำเนา

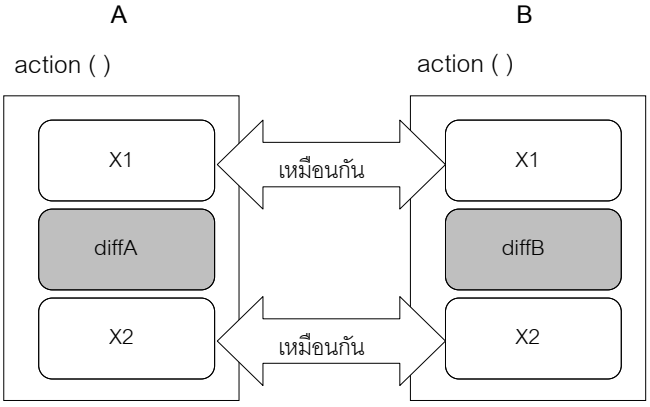
3.3 การก่อแบบตัวเลือกสินทรัพย์ร่วม

ในหัวข้อนี้จะอธิบายการนำผลลัพธ์สำเนาโค้ดที่ถูกจำแนกประเภทแล้วมาทำการก่อแบบเป็นตัวเลือกสินทรัพย์ร่วม โดยมุ่งเน้นไปที่เมท็อดสำเนาโค้ดในประเภทที่ 3 โดยละเว้นประเภทที่ 1 และ 2 ไว้ โดยมีเหตุผลดังนี้

- สำเนาโค้ดประเภทที่ 1 ไม่มีความแตกต่างใดๆ ยกเว้นผัง และหมายเหตุ จึงสามารถนำไปใช้ซ้ำได้เลย
- สำเนาโค้ดประเภทที่ 2 ซึ่งมีความแตกต่างในชื่อของตัวระบุ จำเป็นต้องมีการตรวจสอบในเชิงความหมายเพิ่มเติม นอกเหนือจากการตรวจสอบเชิงข้อความ เนื่องจากตัวระบุที่มีชื่อต่างกัน อาจเกิดจากการเปลี่ยนชื่อ หรือการอ้างอิงถึงตัวระบุตัวอื่นก็ได้

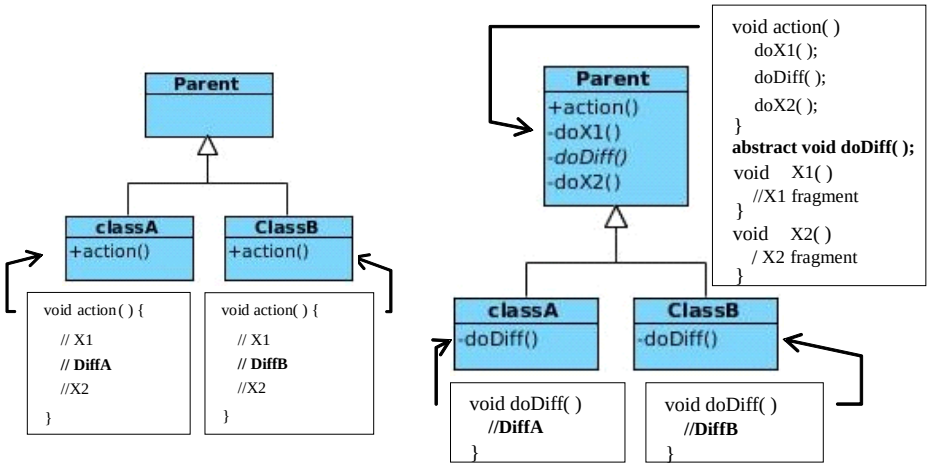
- สำเนาโค้ดประเภทที่ 3 มีการเพิ่ม ลด หรือแก้ไขในบางข้อความสั่ง ซึ่งถือเป็นความแตกต่างอย่างชัดเจน

แนวทางการแยกส่วนที่แตกต่างออกจากส่วนร่วมนั้น ใช้เมท็อดแม่แบบ ซึ่งเป็นแบบรูปการออกแบบซึ่งแก้ปัญหาความซ้ำซ้อนระหว่างคลาสสองคลาส ในระดับเมท็อดที่มีลำดับการทำงานในภาพรวมที่เหมือนกัน แต่มีความแตกต่างในบางขั้นตอน ทำให้ไม่สามารถรวมทั้งสองคลาสเข้าด้วยกันได้ทันที ในแบบรูปการออกแบบนี้จึงทำการแยกโค้ดส่วนที่เหมือนกันและแตกต่างกันออกจากกัน โดยย้ายโค้ดส่วนที่เหมือนกันไปไว้ในเมท็อดที่ถูกสร้างขึ้นใหม่ในคลาสแม่ของคลาสทั้งสอง ซึ่งถูกเรียกว่า เมท็อดแม่แบบ ในเมท็อดแม่แบบนี้จะมีการแทนที่จุดที่แตกต่างด้วยคำสั่งเรียกใช้เมท็อดซึ่งถูกเรียกว่า เมท็อดฮุค ซึ่งเป็นเมท็อดนามธรรม จากนั้นเมท็อดฮุคจะถูกเขียนขึ้นเป็นเมท็อดรูปธรรมในแต่ละคลาสลูก ด้วยโค้ดส่วนที่แตกต่างนั้นเอง เพื่อโอเวอร์ไรด์เมท็อดฮุคในคลาสแม่อีกทีหนึ่ง



รูปที่ 15 ตัวอย่างเมท็อดคล้ายกันที่มีส่วนแตกต่างอยู่ในเมท็อด

จากตัวอย่างในรูปที่ 15 ถ้ามีคลาส *A* และคลาส *B* ซึ่งมีลำดับในเมท็อด *action( )* ที่เหมือนกันในช่วงชิ้นส่วนโค้ด *X1* และ *X2* แต่มีชิ้นส่วนโค้ดที่แตกต่างตรงกลาง คือ ชิ้นส่วนโค้ด *diffA* ที่อยู่ในคลาส *A* และชิ้นส่วนโค้ด *diffB* ที่อยู่ในคลาส *B* ซึ่งเขียนเป็นแผนภาพคลาสได้ดังรูปที่ 16(ก) และแปลงเป็นเมท็อดแม่แบบโดยดึงเมท็อด *action( )*ขึ้นไปยังคลาสแม่ Parent และสร้างเมท็อดฮุค *doDiff( )* ซึ่งถูกโอเวอร์ไรด์ในคลาส *A* และคลาส *B* ด้วยชิ้นส่วนโค้ด *diffA* และ *diffB* ตามลำดับ ดังแสดงในรูปที่ 16(ข)



(ก) (ข)

รูปที่ 16 แผนภาพคลาสและตัวอย่างโค้ด (ก)ก่อน และ (ข)หลังการแปลงเป็นเมท็อดแม่แบบ

งานวิจัยนี้ได้ประยุกต์การสร้างเมท็อดแม่แบบในการก่อแบบตัวเลือกสินทรัพย์ร่วม เพื่อแยกโค้ดส่วนที่แตกต่างในส่วนร่วมของเมท็อดที่ต้องการ โดยรวมส่วนร่วมไว้ในคลาสแม่สำหรับนำไปใช้ซ้ำ ส่วนคลาสลูกนั้นเก็บเมท็อดฮุคซึ่งมีชุดคำสั่งสำหรับแต่ละผลิตภัณฑ์เพื่อโอเวอร์ไรด์ในจุดที่แตกต่าง

อย่างไรก็ตามการจะสร้างเมท็อดฮุคได้ จำเป็นต้องมีการระบุตำแหน่งจุดที่แตกต่าง เพื่อแทนที่ด้วยข้อความสั่งเรียกเมท็อดฮุคในคลาสแม่และย้ายชิ้นส่วนโค้ดที่แตกต่างไปไว้ในแต่ละคลาสลูก ดังนั้นงานวิจัยนี้จึงใช้ขั้นตอนวิธีหาลำดับเหมือนยาวที่สุด [32] เช่นเดียวกับเครื่องมือ Unix diff เพื่อหาตำแหน่งจุดที่แตกต่างระหว่างคู่สำเนาโค้ดประเภทที่ 3 ที่ได้ตรวจพบและจำแนกแล้ว และเพื่อให้แน่ใจว่าความแตกต่างเกิดจากการแก้ไขข้อความสั่งเท่านั้น สำเนาโค้ดจึงถูกทำให้อยู่ในรูปแบบผังเดียวกันและลบหมายเหตุออกก่อนจะทำการเปรียบเทียบหาจุดแตกต่าง

การย้ายชิ้นส่วนโค้ดที่แตกต่างมาเก็บไว้ในเมท็อดฮุคไม่สามารถทำได้กับทุกกรณี ยกตัวอย่างเช่น ข้อความสั่งควบคุมพิเศษ (Special control statement) ซึ่งได้แก่ ข้อความสั่ง return ข้อความสั่ง continue และข้อความสั่ง break เนื่องจากการย้ายข้อความสั่งประเภทนี้จะทำให้ขอบเขตการทำงานเปลี่ยนไป ไม่เหมือนเดิม งานวิจัยนี้จึงไม่รวมกรณีข้อความสั่งควบคุมพิเศษไว้ด้วย เพื่อหลีกเลี่ยงกรณีดังกล่าว รวมถึงกรณีที่เมท็อดสำเนาโค้ดมีรูปแบบอาร์กิวเมนต์ (argument) ของเมท็อดที่เปลี่ยนไป เช่น ถ้ามีเมท็อด loadElement ซึ่งอยู่ในผลิตภัณฑ์ A เป็น

public void loadElement(String name, String prefix) {

....

}

และเมท็อด loadElement ในผลิตภัณฑ์ B เป็น

public void loadElement(String name, String reference, String prefix) {

....

}

ซึ่งจะเห็นว่าในอาร์กิวเมนต์มี reference เพิ่มขึ้นมา ดังนั้นหากจะแก้ไข ต้องตัดสินใจว่าจะเปลี่ยนรูปแบบอย่างไร และค้นหาคำสั่งที่ใช้งานเมท็อดนี้ทั้งหมดเพื่อเปลี่ยนให้การเรียกใช้งานตรงกัน

นอกเหนือจากสำเนาโค้ดประเภทที่ 3 ซึ่งถูกแปลงให้อยู่ในรูปแบบเมท็อดแม่แบบแล้ว สำเนาโค้ดประเภทที่ 1 ซึ่งไม่มีความความแตกต่างนอกเหนือจากหมายเหตุและผังของโค้ด สามารถย้ายไปไว้ในคลาสแม่ได้โดยตรง ส่วนสำเนาโค้ดประเภทที่ 2 นั้นมีจุดแตกต่างของซึ่งเกิดจากตัวระบุที่มีชื่อไม่เหมือนกัน แต่มีรูปแบบของชุดคำสั่งที่เหมือนหรือคล้ายกัน ซึ่งอาจรวมให้เป็นส่วนร่วมได้โดยการเปลี่ยนชื่อตัวระบุให้เป็นชื่อเดียวกัน แต่ในบางกรณี ชื่อที่ต่างกันอาจเกิดจากอ้างอิงตัวระบุคนละชนิดกันก็ได้ ยกตัวอย่างเช่น หากมีชุดคำสั่ง

X = operation.do() +Y;

และ

X = equation.do() +Y;

ความแตกต่างของทั้งสองชุดคำสั่ง คือ การเรียกใช้เมท็อด do() ของวัตถุ operation และ equation ซึ่งรูปแบบของชุดคำสั่งมีความเหมือนกัน แต่วัตถุทั้งสองอาจเป็นวัตถุชนิดเดียวกันแล้วแต่มีชื่อที่ต่าง หรืออาจเป็นวัตถุคนละชนิดที่มีชื่อเมท็อดหรือลักษณะประจำ (Attribute) ที่เหมือนกัน ทำให้มีรูปแบบการใช้งานที่คล้ายกัน ด้วยเหตุผลดังกล่าว ในที่นี้จึงแยกสำเนาโค้ดประเภทที่ 2 ไว้ต่างหากในคลาสลูกของแต่ละผลิตภัณฑ์ เพื่อให้ผู้พัฒนาสินทรัพย์ทำการวิเคราะห์ และตัดสินใจเองต่อไป

บทที่ 4
การออกแบบและพัฒนาเครื่องมือ

ในบทนี้เป็นการอธิบายถึงการออกแบบและพัฒนาเครื่องมือตามแนวคิดที่ได้นำเสนอในบทที่ 3 ซึ่งใช้การตรวจหาสำเนาโค้ดเพื่อระบุส่วนร่วม โดยมุ่งเน้นการจำแนกสำเนาโค้ดที่มีความแตกต่าง แล้วจึงแยกจุดที่แตกต่างนั้นออกจากส่วนร่วมด้วยแนวทางการแบบเมท็อดแม่แบบ

ตารางที่ 3 ข้อกำหนดเบื้องต้นของระบบ

ข้อกำหนดของระบบเบื้องต้น	รายละเอียด
ผู้ใช้งานระบบ	- เป็นผู้มีหน้าที่วิเคราะห์โดเมน ซึ่งควรมีความรู้เกี่ยวกับสายผลิตภัณฑ์ซอฟต์แวร์ และข้อมูลที่เกี่ยวข้องกับโค้ดของผลิตภัณฑ์ที่เป็นข้อมูลนำเข้า
ข้อมูลนำเข้า	- โค้ดของผลิตภัณฑ์ทั้งสองระบบที่จะทำการเปรียบเทียบ
ข้อมูลนำออก	- โค้ดซึ่งถูกแยกเป็นคลาสแม่ซึ่งรวมส่วนร่วมของผลิตภัณฑ์ และคลาสลูกซึ่งรวมส่วนแปรผันของแต่ละผลิตภัณฑ์
สภาพแวดล้อมของระบบขณะทำการ	ระบบปฏิบัติการ ลินุกซ์ที่มีซอฟต์แวร์ติดตั้งไว้ ดังนี้ - ไพทอน รุ่น 2.6 ขึ้นไป - ทีเอ็กซ์แอล รุ่น 10.5 ขึ้นไป - นิแค็ด รุ่น 2.9

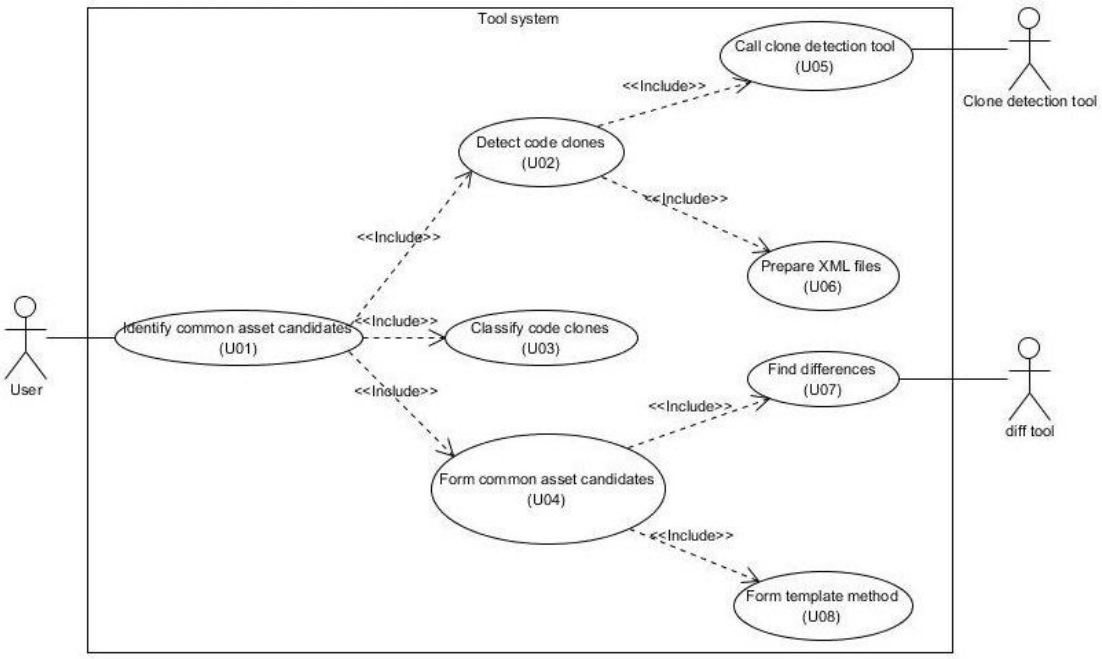
ผู้วิจัยได้กำหนดข้อกำหนดเบื้องต้นของระบบ (Prerequisites) ซึ่งได้แก่ ผู้ใช้งานระบบ (User) ข้อมูลนำเข้า (Input) ข้อมูลนำออก (Output) และสภาพแวดล้อมของระบบขณะทำการ (Execution environment) ดังแสดงในตารางที่ 3 ผู้ใช้งานระบบควรมีความรู้เกี่ยวกับสายผลิตภัณฑ์ และข้อมูลที่เกี่ยวข้องกับโค้ดของผลิตภัณฑ์ที่เป็นข้อมูลนำเข้าเพื่อตรวจสอบความถูกต้องของข้อมูลนำออก โดยข้อมูลนำเข้า คือ โค้ดของผลิตภัณฑ์ที่มีอยู่ทั้งสองระบบ และข้อมูลนำออก คือ โค้ดที่เป็นตัวเลือกสินทรัพย์ร่วมซึ่งถูกรวบรวมไว้ในรูปแบบของคลาสแม่ และแบ่งส่วนแปรผันไว้ในคลาสลูกสำหรับแต่ละผลิตภัณฑ์ ในส่วนสภาพแวดล้อมของระบบขณะทำการของเครื่องมือนั้น ได้ใช้ภาษาไพทอน (Python) ในการพัฒนาซึ่งสามารถใช้งานข้ามระบบปฏิบัติการได้ แต่เนื่องจากนิแค็ดรองรับเฉพาะการใช้งานบนระบบปฏิบัติการลินุกซ์ (Linux) และแมคโอเอส (Mac OS) เท่านั้น ในที่นี้จึงกำหนดให้สภาพแวดล้อมของระบบต้องใช้ระบบปฏิบัติการลินุกซ์ในการดำเนินการ

ข้อมูลการออกแบบระบบถูกแสดงด้วยแผนภาพการออกแบบ ได้แก่ แผนภาพยูสเคส แผนภาพคลาส แผนภาพกิจกรรม และแผนภาพลำดับ ซึ่งถูกแสดงต่อไปในบทนี้

4.1 แผนภาพยูสเคส (Use case diagram)

ภาพรวมเชิงฟังก์ชันของระบบ ถูกแสดงตามแผนภาพยูสเคสในรูปที่ 17 ซึ่งประกอบด้วยยูสเคสทั้งสิ้น 8 ยูสเคส ซึ่งได้แก่

- Identify common asset candidates (รหัส U01)
- Detect code clones (รหัส U02)
- Classify code clones (รหัส U03)
- Form common asset candidates (รหัส U04)
- Call clone detection tool (รหัส U05)
- Prepare XML files (รหัส U06)
- Find differences (รหัส U07)
- Form template method (รหัส U08)



รูปที่ 17 แผนภาพยูสเคสของระบบเครื่องมือสนับสนุน

ขั้นตอนการทำงานเริ่มต้นจาก ผู้ใช้งานนำเข้าข้อมูลโดยกำหนดโฟลเดอร์ของโค้ดในแต่ละผลิตภัณฑ์ที่จะทำการเปรียบเทียบผ่านยูสเคส U01 ซึ่งประกอบด้วยยูสเคสการทำงานหลัก คือ

U02, U03 และ U04 ซึ่งทำการตรวจหาสำเนาโค้ดระหว่างผลิตภัณฑ์ จำแนกผลสำเนาโค้ดที่ตรวจพบ และก่อแบบสำเนาโค้ดประเภทที่ 3 ให้เป็นตัวเลือกสินทรัพย์ร่วม ส่วนยูสเคส U05, U06, U07 และ U08 นั้นเป็นยูสเคสการทำงานย่อยๆ ซึ่งถูกแบ่งออกมาอีก อธิบายยูสเคสสำหรับแต่ละยูสเคสถูกแสดงไว้ในตารางที่ 4 ถึงตารางที่ 11 ตามลำดับ

ตารางที่ 4 คำอธิบายยูสเคส Identify common asset candidates

Use case Name: Identify common asset candidates	**ID:** U01
Primary Actor: User	
Description: ควบคุมการทำงานโดยรวมของระบบเพื่อระบุตัวเลือกสินทรัพย์ร่วม	
Pre condition: ผู้ใช้งานระบบนำเข้าโค้ดของผลิตภัณฑ์ทั้งสองผลิตภัณฑ์ที่จะนำมาเปรียบเทียบเพื่อระบุตัวเลือกสินทรัพย์ร่วม	
Relationships: **Association:** User **Include:** Detect code clones, Classify code clones, Form common asset candidates **Extend:** -	
Normal Flow of Event: 1.ผู้ใช้งานระบบกำหนดโฟลเดอร์ที่เก็บโค้ดของผลิตภัณฑ์ที่จะนำเข้าเพื่อเปรียบเทียบ 2.ผู้ใช้งานระบบกำหนดพารามิเตอร์ตัวเลือกที่จำเป็นสำหรับการตรวจหาสำเนาโค้ด 3.ระบบทำการตรวจหาสำเนาโค้ดตามพารามิเตอร์ที่กำหนด 4.ระบบจำแนกประเภทผลสำเนาโค้ดที่ตรวจพบ 5.ระบบก่อแบบตัวเลือกสินทรัพย์ร่วม	
Sub flow: -	
Alternative/Exception Flows: 1.a ผู้ใช้งานระบบกำหนดโฟลเดอร์นำเข้าไม่ถูกต้อง 1.a.1 มีการแจ้งข้อผิดพลาด 2.a ผู้ใช้งานระบบกำหนดพารามิเตอร์ไม่ถูกต้อง 2.a.1 มีการแจ้งข้อผิดพลาด	
Post Condition: ตัวเลือกสินทรัพย์ร่วม	

ตารางที่ 5 คำอธิบายยูสเคส Detect code clones

Use case Name: Detect code clones	**ID:** U02
Primary Actor: -	
Description: ตรวจหาสำเนาโค้ดระหว่างผลิตภัณฑ์สองผลิตภัณฑ์	

Pre condition: โฟลเดอร์เป้าหมายที่มีโค้ดของผลิตภัณฑ์ และพารามิเตอร์ตัวเลือกถูกกำหนด
Relationships: **Association:** - **Include:** Call detection tool, Prepare XML files **Extend:** -
Normal Flow of Event: 1.ระบบทำการเรียกเครื่องมือตรวจหาสำเนาด้วยพารามิเตอร์ที่กำหนดไว้ในแต่ละวิธีการ 2.ระบบเตรียมผลสำเนาโค้ดที่ได้ ให้อยู่รูปแบบที่ดีของภาษาเอ็กซ์เอ็มแอล
Sub flow: -
Alternative/Exception Flows: -
Post Condition: ผลสำเนาโค้ดที่อยู่ในรูปแบบแฟ้มเอ็กซ์เอ็มแอล

ตารางที่ 6 คำอธิบายยูสเคส Classify code clones

Use case Name: Classify code clones	**ID:** U03
Primary Actor: -	
Description: จำแนกประเภทสำเนาโค้ดโดยการเปรียบเทียบผลสำเนาโค้ดที่มาจากต่างวิธีการ	
Pre condition: ผลสำเนาโค้ดจากแต่ละวิธีการซึ่งอยู่ในรูปแบบแฟ้มเอ็กซ์เอ็มแอล	
Relationships: **Association:** - **Include:** - **Extend:** -	
Normal Flow of Event: 1.ระบบเปรียบเทียบผลสำเนาโค้ด และหักลบส่วนที่ซ้ำกันออก เพื่อจำแนกประเภทสำเนาโค้ด 2.ระบบจัดเก็บผลสำเนาโค้ดตามประเภทของสำเนาโค้ด	
Sub flow: -	
Alternative/Exception Flows: -	
Post Condition: ผลสำเนาโค้ดที่อยู่ในรูปแบบแฟ้มเอ็กซ์เอ็มแอล ซึ่งได้จำแนกตามประเภทสำเนาโค้ดแล้ว	

ตารางที่ 7 คำอธิบายยูสเคส Form common asset candidates

Use case Name: Form common asset candidates	**ID:** U04
Primary Actor: -	
Description: ก่อแบบตัวเลือกสินทรัพย์ร่วม	

Pre condition: ผลสำเนาโค้ดที่อยู่ในรูปแบบแฟ้มเอ็กซ์เอ็มแอล ซึ่งได้จำแนกตามประเภทสำเนาโค้ดแล้ว
Relationships: **Association:** - **Include:** Find differences, Form template method **Extend:** -
Normal Flow of Event: 1.ระบบเปิดแฟ้มสำเนาโค้ดประเภทที่ 3 เพื่อเปรียบเทียบหาจุดแตกต่างที่เกิดขึ้นในคู่สำเนาโค้ด 2.ระบบสร้างเมท็อดแม่แบบ และแทนที่เมท็อดฮุคในตำแหน่งจุดแตกต่างที่พบ
Sub flow: -
Alternative/Exception Flows: -
Post Condition: คลาสแม่ซึ่งเก็บเมท็อดแม่แบบ และคลาสลูกซึ่งเก็บเมท็อดฮุคของแต่ละผลิตภัณฑ์

ตารางที่ 8 คำอธิบายยูสเคส Call detection tool

Use case Name: Call clone detection tool	**ID:** U05
Primary Actor: clone detection tool	
Description: เรียกใช้งานเครื่องมือตรวจหาสำเนาโค้ด	
Pre condition: โค้ดของทั้งสองผลิตภัณฑ์ที่จะเปรียบเทียบตรวจหาสำเนาโค้ด และพารามิเตอร์ที่จำเป็น	
Relationships: **Association:** clone detection tool **Include:** - **Extend:** -	
Normal Flow of Event: 1.ระบบเรียกใช้งานเครื่องมือตรวจหาสำเนาโค้ด และกำหนดพารามิเตอร์ 2.ระบบจัดเก็บแฟ้มสำเนาโค้ด และแฟ้มชิ้นส่วนโค้ดที่ทำให้เป็นบรรทัดฐานเดียวกันแล้ว	
Sub flow: -	
Alternative/Exception Flows: -	
Post Condition: แฟ้มสำเนาโค้ด และแฟ้มชิ้นส่วนโค้ดที่ทำให้เป็นบรรทัดฐานเดียวกันแล้ว	

ตารางที่ 9 คำอธิบายยูสเคส Prepare XML files

Use case Name: Prepare XML files	**ID:** U06
Primary Actor: -	
Description: จัดเตรียมแฟ้มสำเนาโค้ด และแฟ้มชิ้นส่วนโค้ดที่ทำให้เป็นบรรทัดฐานเดียวกันแล้ว ซึ่งเป็นแฟ้ม	

เอ็กซ์เอ็มแอลให้อยู่ในรูปแบบที่ดีพร้อมนำไปใช้งานต่อ
Pre condition: แฟ้มสำเนาโค้ด และแฟ้มชิ้นส่วนโค้ดที่ทำให้เป็นบรรทัดฐานเดียวกันแล้วจากเครื่องมือตรวจหาสำเนาโค้ด
Relationships: Association: - Include: - Extend: -
Normal Flow of Event: 1.ระบบเพิ่มป้ายระบุกำหนดการเข้ารหัสภาษาของแฟ้มเอ็กซ์เอ็มแอล 2.ระบบเพิ่มป้ายระบุครอบชิ้นส่วนโค้ดในแฟ้มเอ็กซ์เอ็มแอล เพื่อละเว้นการแจงส่วน 3.ระบบจัดเก็บแฟ้มเอ็กซ์เอ็มแอลที่อยู่ในรูปแบบที่ดีแล้ว
Sub flow: -
Alternative/Exception Flows: -
Post Condition: แฟ้มเอ็กซ์เอ็มแอลที่อยู่ในรูปแบบที่ดีแล้ว

ตารางที่ 10 คำอธิบายยูสเคส Find differences

Use case Name: Find differences	ID: U07
Primary Actor: Diff tool	
Description: เรียกใช้งาน Diff tool เพื่อเปรียบเทียบหาจุดแตกต่างในคู่สำเนาโค้ดประเภทที่ 3	
Pre condition: สำเนาโค้ดประเภทที่ 3 ซึ่งเก็บเป็นแฟ้มเอ็กซ์เอ็มแอล	
Relationships: Association: Diff tool Include: - Extend: -	
Normal Flow of Event: 1.ระบบอ่านแฟ้มสำเนาโค้ดประเภทที่ 3 2.ระบบเรียกใช้งาน Diff tool เพื่อเปรียบเทียบหาจุดแตกต่างในคู่สำเนาโค้ดประเภทที่ 3 3.นำออกตำแหน่งของจุดแตกต่างในแต่ละคู่สำเนาโค้ด	
Sub flow: -	
Alternative/Exception Flows: -	
Post Condition: ตำแหน่งของจุดแตกต่างในแต่ละคู่สำเนาโค้ดประเภทที่ 3	

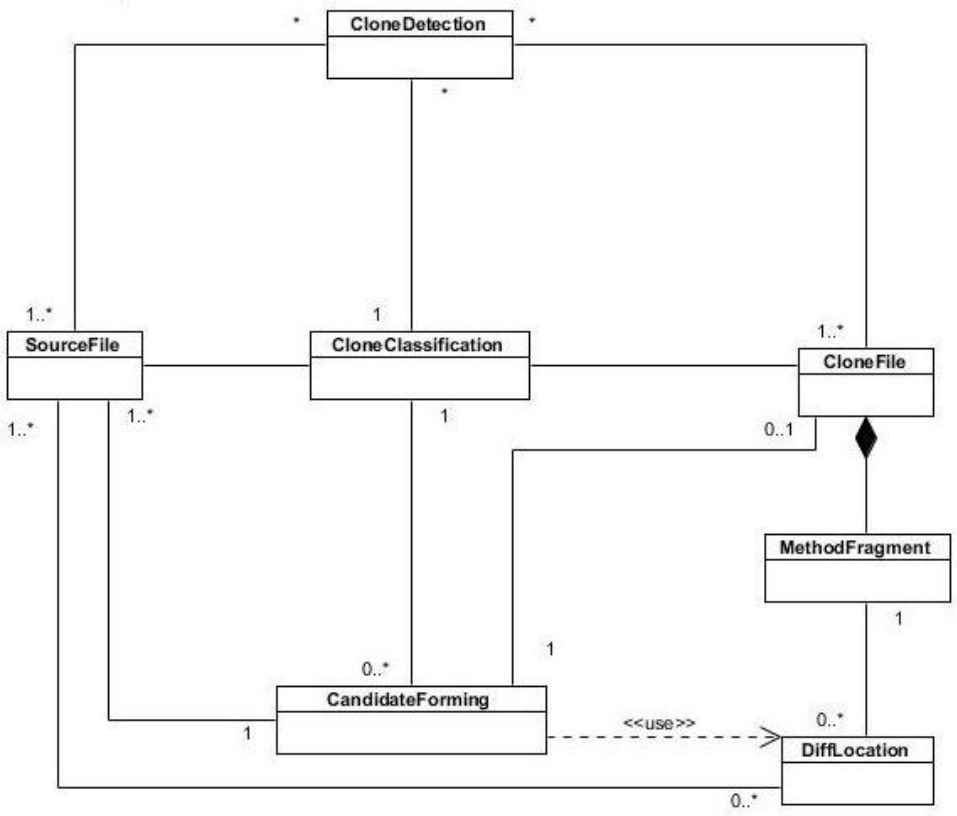
ตารางที่ 11 คำอธิบายยูสเคส Form template method

Use case Name: Form template method	**ID:** U08
Primary Actor: -	
Description: เรียกใช้งาน Diff tool เพื่อเปรียบเทียบหาจุดแตกต่างในคู่สำเนาโค้ดประเภทที่ 3	
Pre condition: สำเนาโค้ดประเภทที่ 3 ซึ่งอยู่ในรูปแบบแฟ้มเอ็กซ์เอ็มแอล และตำแหน่งของจุดแตกต่าง	
Relationships: **Association:** - **Include:** - **Extend:** -	
Normal Flow of Event: 1.ระบบอ่านแฟ้มสำเนาโค้ดประเภทที่ 3 2.ระบบสร้างคลาสแม่ ซึ่งคัดลอกเมท็อดซึ่งเป็นสำเนาโค้ดประเภทที่ 3 เพื่อเป็นเมท็อดแม่แบบ 3.ระบบแทนที่ชิ้นส่วนโค้ดในจุดที่แตกต่าง ด้วยคำสั่งเรียกใช้เมท็อดฮุค 4.ระบบสร้างคลาสลูกสำหรับแต่ละผลิตภัณฑ์ และสร้างเมท็อดฮุคโดยใช้ชิ้นส่วนโค้ดในจุดที่แตกต่าง	
Sub flow: -	
Alternative/Exception Flows: -	
Post Condition: คลาสแม่ซึ่งเก็บเมท็อดแม่แบบ และคลาสลูกซึ่งเก็บเมท็อดฮุคของแต่ละผลิตภัณฑ์	

4.2 **แผนภาพคลาส** (Class diagram)

แผนภาพคลาสถูกแสดงดังรูปที่ 18 ซึ่งประกอบด้วยคลาสดังต่อไปนี้

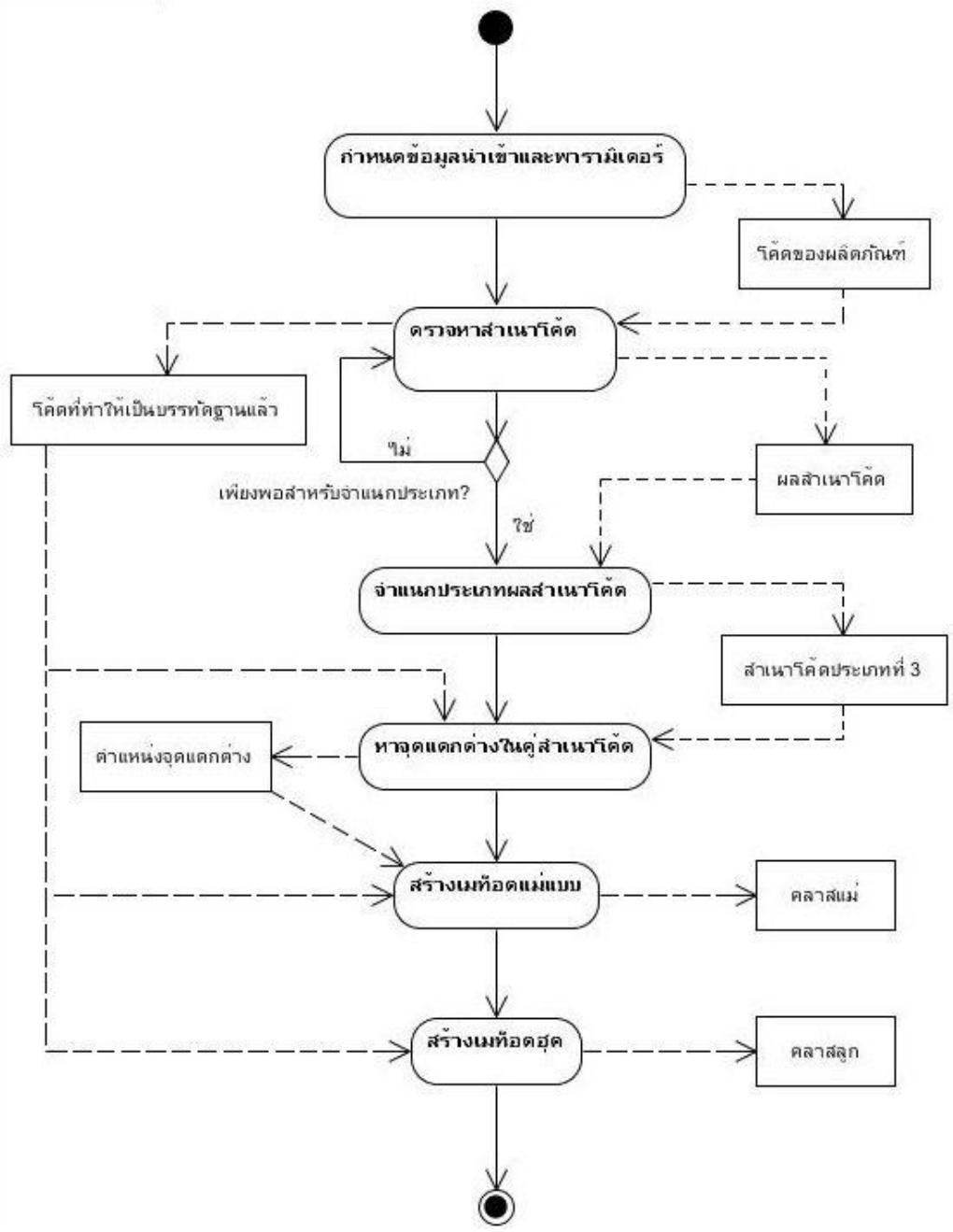
- SourceFile เป็นคลาสของแฟ้มโค้ดของผลิตภัณฑ์
- CloneFile เป็นคลาสของแฟ้มสำเนาโค้ ซึ่งอยู่ในรูปแบบเอ็กซ์เอ็มแอล
- MethodFragment เป็นคลาสของชิ้นส่วนโค้ดในระดับเมท็อด ซึ่งอยู่ใน CloneFile
- CloneDetection เป็นคลาสที่ควบคุมการติดต่อกับเครื่องมือตรวจหาสำเนาโค้ด
- CloneClassification เป็นคลาสที่ควบคุมการแยกประเภทของผลสำเนาโค้ด
- CandidateForming เป็นคลาสที่ควบคุมการก่อแบบตัวเลือกสินทรัพย์ร่วม
- DiffLocation เป็นคลาสที่จัดการการหาจุดแตกต่างระหว่างเมท็อดในคู่สำเนาโค้ด



รูปที่ 18 แผนภาพคลาสโดยสังเขปของระบบเครื่องมือสนับสนุน

4.3 **แผนภาพกิจกรรม** (Activity diagram)

ขั้นตอนการทำงานโดยสังเขปของเครื่องมือนั้น สามารถแบ่งออกเป็นขั้นตอนดังแสดงตามแผนภาพกิจกรรมในรูปที่ 19 ซึ่งขั้นตอนการทำงานเริ่มต้นจากการกำหนดข้อมูลนำเข้าและพารามิเตอร์ที่จำเป็น แล้วจึงตรวจหาสำเนาโค้ดและจำแนกประเภทจนได้สำเนาโค้ดประเภทที่ 3 ซึ่งจะถูกนำไปหาจุดแตกต่างในเมท็อดเพื่อสร้างเป็นตัวเลือกสินทรัพย์ร่วม



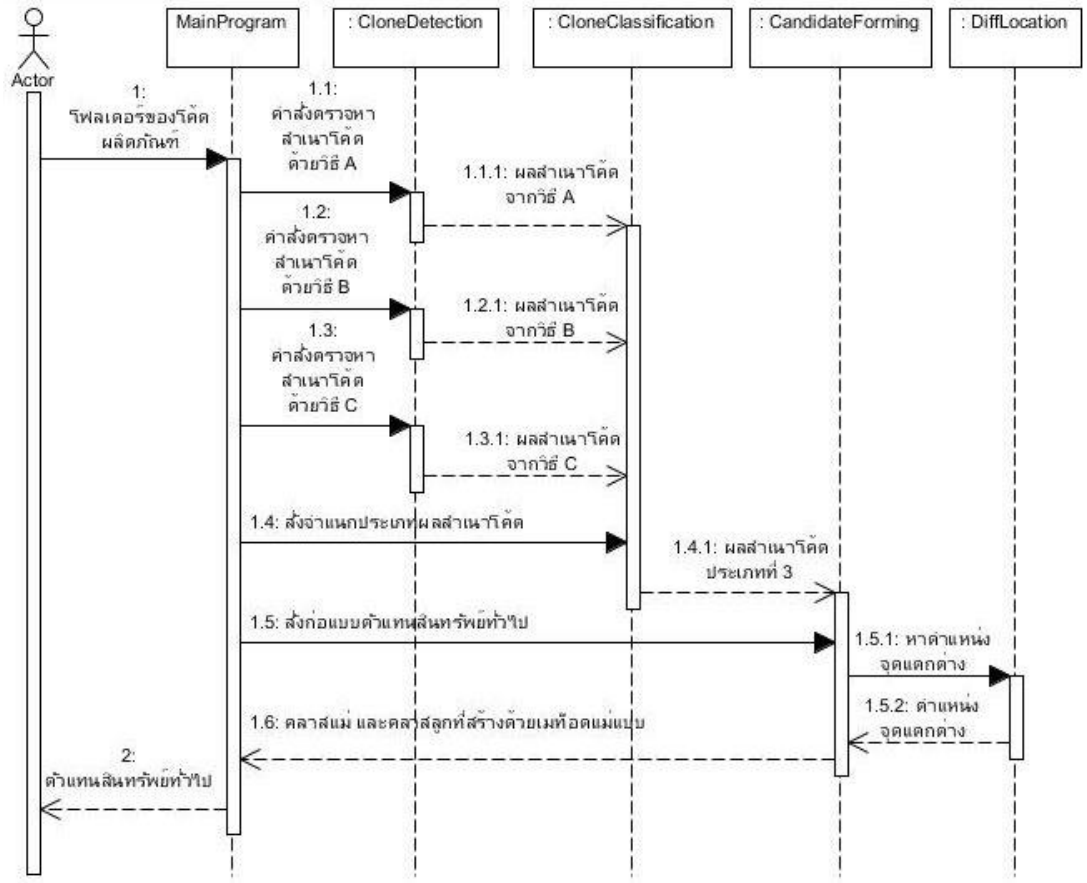
รูปที่ 19 แผนภาพกิจกรรมแสดงขั้นตอนการทำงานของระบบเครื่องมือสนับสนุน

4.4 แผนภาพลำดับ (Sequence diagram)

เพื่อให้เข้าใจลำดับการทำงานในภาพรวม แผนภาพลำดับของยูสเคส Identify common asset candidates จึงถูกแสดงไว้ในรูปที่ 20 โดยเริ่มต้นจากการที่ผู้ใช้กำหนดโฟลเดอร์ที่บรรจุโค้ดของผลิตภัณฑ์ที่จะเปรียบเทียบระบุตัวเลือกสินทรัพย์ร่วม จากนั้นโปรแกรมหลักจะสั่งให้วัตถุ

CloneDetection ทำการตรวจหาสำเนาโค้ดด้วยวิธี A, B และ C ตามลำดับดังที่ได้อธิบายไว้ในบทที่ 3 เพื่อใช้ในการจำแนกประเภทต่อไปโดยวัตถุ CloneClassification และท้ายที่สุดก็ติดต่อกับวัตถุ CandidateForming เพื่อสร้างตัวเลือกสินทรัพย์ร่วม โดยใช้วัตถุ DiffLocation ในการหาจุดแตกต่างระหว่างเมท็อดที่เป็นคู่สำเนาโค้ด

ตัวเลือกสินทรัพย์ร่วมที่ได้ จะอยู่ในรูปแบบคลาสแม่ซึ่งรวบรวมส่วนร่วมที่แยกความแตกต่างไว้ในคลาสลูก แล้วใช้การโอเวอร์ไรด์เพื่อใช้งานต่อไป



รูปที่ 20 แผนภาพลำดับแสดงขั้นตอนของยูสเคส Identify common asset candidates

บทที่ 5
การประเมินผลวิธีการ

ในบทนี้จะกล่าวถึงรายละเอียดการประเมินผลวิธีการ โดยการทดลองเพื่อหาพารามิเตอร์ที่เหมาะสม และตรวจสอบความถูกต้องของผลลัพธ์ของวิธีการที่ได้นำเสนอ ซึ่งเนื้อหาในบทนี้ถูกแบ่งออกเป็น 4 ส่วน ได้แก่ โปรแกรมที่นำมาใช้ทดลอง วิธีการประเมินผล ผลการทดลองในการก่อแบบตัวเลือกสินทรัพย์ร่วม และบทวิเคราะห์ผลการทดลอง

ตารางที่ 12 คุณลักษณะทั่วไปของโปรแกรม JabRef ในแต่ละเวอร์ชันหลัก

เวอร์ชันของ JabRef	**คุณลักษณะ**		
	จำนวนบรรทัดโค้ด	**จำนวนแฟ้มโค้ด**	**จำนวนเมท็อด**
1.0	11352	91	855
1.1	13450	101	978
1.2	16827	149	1215
1.3.1	18750	155	1311
1.4	57517	402	3918
1.5	22041	178	1542
1.6	29016	224	1953
1.7	34320	294	2438
1.8	38680	311	2686
2.0	44444	360	3310
2.1	46263	377	3424
2.2	58152	470	4039
2.3	64730	511	4453
2.4	68953	537	4773
2.5	72512	562	4976
2.6	74264	578	5112
2.7	77814	597	5492

5.1 โปรแกรมที่นำมาใช้ในการทดลอง

ในที่นี้ได้ใช้โปรแกรมที่มีใบอนุญาตเป็นแบบโอเพนซอร์ส (Open source) มาใช้จำลองแทนสายผลิตภัณฑ์ซอฟต์แวร์ โดยมองเวอร์ชันที่แตกต่างของโปรแกรมเสมือนเป็นแต่ละผลิตภัณฑ์ โดยมีเหตุผลดังนี้

- การหาโค้ดของผลิตภัณฑ์ซอฟต์แวร์เพื่อมาใช้ทดลองในงานวิจัยเป็นไปได้ยาก เนื่องจากมักเป็นซอฟต์แวร์เชิงพาณิชย์ซึ่งปกปิดข้อมูล
- โครงการที่แยกสาขา (Forked project) อาจนำมาใช้ทดลองโดยสมมุติเป็นผลิตภัณฑ์ได้ อย่างไรก็ตามการแยกสาขาของโครงการนั้นเป็นแนวทางที่ไม่นิยมปฏิบัติ และโครงการเหล่านั้นมักจะไม่ใช้ภาษาเชิงวัตถุในการพัฒนา จึงไม่สามารถนำมาใช้กับวิธีการในงานวิจัยนี้ได้

โปรแกรมที่นำมาใช้ทดลองนั้น คือ JabRef [33] ซึ่งเป็นโปรแกรมจัดการฐานข้อมูลบรรณานุกรมที่ใช้ภาษาจาวาในการพัฒนา ซึ่งถูกคัดเลือกมาใช้ทดลองเฉพาะเวอร์ชันหลักตั้งแต่ 1.0 ถึง 2.7 จำนวนทั้งสิ้น 17 เวอร์ชัน ซึ่งรายละเอียดของซอฟต์แวร์ในแต่ละเวอร์ชัน ทั้งจำนวนบรรทัดโค้ด (SLOC) จำนวนแฟ้มโค้ด และจำนวนเมท็อด ถูกแสดงไว้ในตารางที่ 12

5.2 วิธีการประเมินผล

การประเมินผล ถูกแบ่งออกเป็นสองขั้นตอน ได้แก่ 1) การทดลองตรวจหาและจำแนกสำเนาโค้ด และ 2) การทดลองก่อแบบตัวเลือกสินทรัพย์ร่วม โดยในขั้นตอนแรกผู้วิจัยได้ตรวจหาและจำแนกประเภทสำเนาโค้ดระหว่าง JabRef ทั้ง 17 เวอร์ชัน โดยใช้ค่าขีดเริ่มที่แตกต่างกันสำหรับวิธีการตรวจหาสำเนาโค้ด C เพื่อหาค่าขีดเริ่มที่เหมาะสม กล่าวคือ ทำให้จำนวนสำเนาโค้ดที่ตรวจพบมากที่สุด โดยใช้ทรัพยากรให้น้อยที่สุดเท่าที่เป็นไปได้ ซึ่งจากหลักการทำงานของเครื่องมือตรวจหาสำเนาโค้ด NiCad นั้น ก่อนจะทำการเปรียบเทียบหาชิ้นส่วนที่เป็นสำเนาโค้ด ได้ใช้การรวมกลุ่มข้อมูลชิ้นส่วนโค้ดโดยใช้ค่าขีดเริ่มเป็นเกณฑ์ เพื่อลดจำนวนครั้งในการเปรียบเทียบด้วยเหตุผลดังกล่าว การตั้งค่าขีดเริ่มที่มากขึ้นจะทำให้ภาระในการตรวจหาสำเนาโค้ดเพิ่มขึ้นด้วย

ในการทดลองขั้นตอนสุดท้าย ผู้วิจัยได้ใช้ผลสำเนาโค้ดจากค่าขีดเริ่มที่เหมาะสมซึ่งได้มาจากขั้นตอนแรก ทำการจำแนกประเภทสำเนาโค้ด แล้วจึงก่อแบบตัวเลือกสินทรัพย์ร่วมตามขั้นตอนดังที่ได้อธิบายไว้ในบทที่ 3

ตารางที่ 13 ผลการตรวจหาสำเนาโค้ดระหว่างเวอร์ชัน 1.0 และ 1.1 ด้วยค่าขีดเริ่มต่างกัน

ค่าขีดเริ่มของวิธีการ C	ผลการตรวจหาสำเนาโค้ด	
	จำนวนสำเนาประเภทที่ 3	จำนวนครั้งการเปรียบเทียบของวิธีการ C
0.1	97	16907
0.2	118	35085
0.3	134	48464
0.4	171	65868
0.5	201	82406
0.6	236	94160
0.7	309	110531

5.3 ผลการทดลองในการก่อแบบตัวเลือกสินทรัพย์ร่วม

จากการทดลองตรวจหาและจำแนกประเภทสำเนาโค้ดด้วยค่าขีดเริ่มต่างๆ ตั้งแต่ 0.1 ถึง 0.7 พบว่าจำนวนเมท็อดสำเนาโค้ดประเภทที่ 3 ที่ตรวจพบมีจำนวนเพิ่มขึ้น ดังข้อมูลในภาคผนวก ก. ซึ่งแสดงในตารางที่ 19 ถึง 25 และใช้เวลาในการตรวจหาสำเนาโค้ดเพิ่มขึ้นตามจำนวนครั้งที่ใช้ในการเปรียบเทียบ ดังแสดงในตารางที่ 13 ซึ่งสรุปข้อมูลจากการเปรียบเทียบระหว่าง JabRef เวอร์ชัน 1.0 และ 1.1 ซึ่งแสดงถึงแนวโน้มดังกล่าว อย่างไรก็ตามการใช้ค่าขีดเริ่มที่มากเกินไป ทำให้ตรวจพบเมท็อดที่มีการทำงานคล้ายกันแต่ไม่ใช่เมท็อดเดียวกัน เนื่องจากยอมรับความแตกต่างมากเกินไป ในขณะที่ค่าขีดเริ่มที่น้อยเกินไป ทำให้พลาดตรวจหาสำเนาโค้ดบางส่วนไม่พบ ด้วยเหตุผลดังกล่าวในที่นี้จึงได้เลือกใช้ค่าขีดเริ่ม 0.3

ผลการทดลองที่นำมาแสดงในที่นี้ เป็นการระบุตัวเลือกสินทรัพย์ร่วมโดยการทดลองระหว่าง JabRef เวอร์ชัน 1.8 และ 2.0 ซึ่งจากเอกสารช่วยเหลือ พบว่ามีฟังก์ชันการทำงานที่แตกต่างกันบางส่วน ดังแสดงในช่องแรเงาของตารางที่ 14 ผลลัพธ์ถูกออกเป็นสองส่วนตามขั้นตอน ได้แก่ การตรวจหาและจำแนกประเภทสำเนาโค้ด และการก่อแบบตัวเลือกสินทรัพย์ร่วม

5.3.1 การตรวจหาและจำแนกประเภทสำเนาโค้ด

จากการตรวจหาสำเนาโค้ดด้วยวิธี A, B และC ดังที่ได้อธิบายไว้ในบทที่ 3 นั้น จำนวนกลุ่มสำเนาโค้ดที่ตรวจพบโดยแต่ละวิธีถูกแสดงดังตารางที่ 15 จากนั้นจึงหักผลลัพธ์ที่ทับซ้อนกันในแต่ละวิธีเพื่อจำแนกประเภทสำเนาโค้ดตามประเภทที่ 1, 2 และ 3 ตามลำดับ ซึ่งจำนวนของเมท็อดสำเนาโค้ดในแต่ละประเภทถูกแสดงดังตารางที่ 16

ตารางที่ 14 ฟังก์ชันการทำงานของเวอร์ชัน 1.8 และ 2.0 ที่บันทึกไว้ในเอกสารช่วยเหลือ

ฟังก์ชันการทำงานเวอร์ชัน 1.8	**ฟังก์ชันการทำงานเวอร์ชัน 2.0**
Command line	Command line
Custom exports	Custom exports
	Custom exports
Label patterns	Label patterns
Custom entries	Custom entries
Custom general fields	Custom general fields
EndNote filters	EndNote filters
Export to OpenOffice	Export to OpenOffice
Entry Editor	Entry Editor
Fetching entries from Citeseer	Fetching entries from Citeseer
Fetching entries from Medline	Fetching entries from Medline
Content selector	Content selector
	Journal abbreviations
Using Groups	Using Groups
Base frame	Base frame
Marking entries	Marking entries
“Owner” field	“Owner” field
Link to external files	Link to external files
Searching	Searching
String editor	String editor
	Entry time stamps
Import inspection dialog	Import inspection dialog

ตารางที่ 15 ผลการตรวจหาสำเนาโค้ดระหว่างเวอร์ชัน 1.8 และ 2.0 ด้วยวิธี A, B และ C

วิธีการตรวจหาสำเนาโค้ด	**จำนวนกลุ่มสำเนาโค้ดที่ตรวจพบ**
วิธี A	986
วิธี B	1015
วิธี C	1206

ตารางที่ 16 ผลการจำแนกประเภทสำเนาโค้ดระหว่างเวอร์ชัน 1.8 และ 2.0

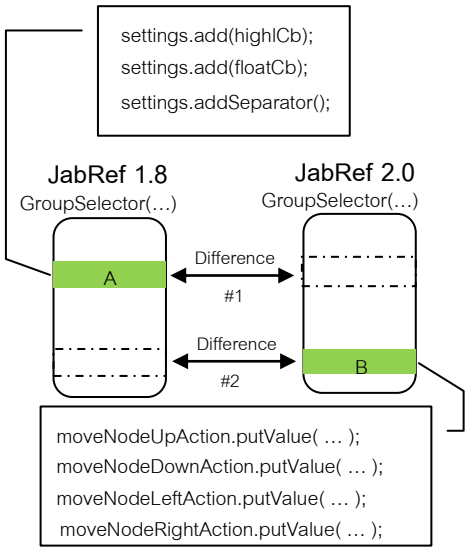
ประเภทสำเนาโค้ด	จำนวนเมท็อดสำเนาโค้ดที่ตรวจพบ
ประเภทที่ 1	986
ประเภทที่ 2	117
ประเภทที่ 3	378

5.3.2 การก่อร่างตัวเลือกสินทรัพย์ร่วม

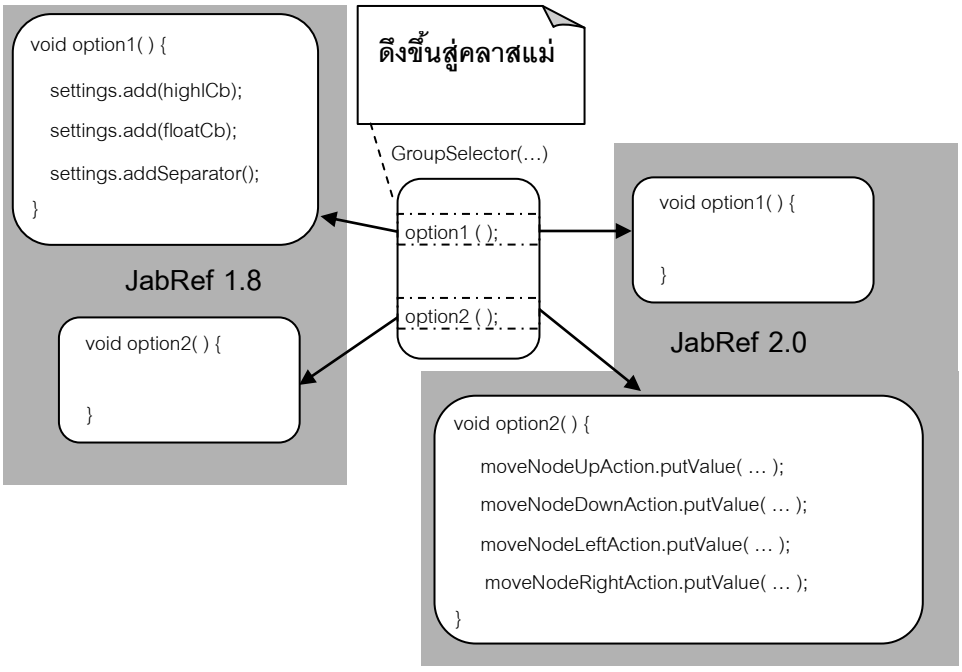
เมื่อได้จำแนกประเภทสำเนาโค้ดแล้ว สำเนาโค้ดประเภทที่ 3 จะถูกนำมาก่อแบบเป็นตัวเลือกสินทรัพย์ร่วม ดังเช่นตัวอย่างคู่เมท็อดสำเนาโค้ดที่แสดงอยู่ในรูปที่ 21 ซึ่งเป็นคู่สำเนาโค้ดจาก JabRef ระหว่างเวอร์ชัน 1.8 และ 2.0 ซึ่งเป็นเมท็อดตัวสร้าง (Constructor method) ของคลาส GroupSelector ซึ่งอยู่ในฟังก์ชันการทำงาน Using Groups ซึ่งเป็นส่วนร่วมของทั้งสองเวอร์ชัน โดยในเมท็อดนี้มีจุดแตกต่างสองจุดในส่วนกลางของเมท็อด GroupSelector() ซึ่งได้แก่ จุดแตกต่างที่ 1 ซึ่งอยู่ส่วนบนของเมท็อดนั้น ในเวอร์ชัน 1.8 มีชิ้นส่วนโค้ด A ซึ่งเพิ่มเติมขึ้นมาจากเวอร์ชัน 2.0 และจุดแตกต่างที่ 2 ชิ้นส่วนโค้ด B ในเวอร์ชัน 2.0 ขาดหายไปจากเวอร์ชัน 1.8

จุดที่แตกต่างทั้งสองจะถูกระบุตำแหน่ง เพื่อแยกโค้ดในจุดแตกต่างไปไว้ในเมท็อดฮุค ซึ่งแสดงดังในตัวอย่างในรูปที่ 22 เมท็อดฮุคของจุดแตกต่างที่ 1 และ 2 คือ เมท็อด option1() และ option2() ตามลำดับ โดยในคลาสแม่นั้นจะบรรจุเมท็อดแม่แบบ GroupSelector เอาไว้ และทำการประกาศเมท็อด option1() และ option2() เป็นเพียงเมท็อดนามธรรมเท่านั้น

ในส่วนที่เดิมไม่มีชิ้นส่วนโค้ดอยู่ สามารถใช้การประกาศเป็นเมท็อดว่างได้ ดังเช่นเมท็อด option1() ของเวอร์ชัน 2.0 และเมท็อด option2() ของเวอร์ชัน 1.8 เป็นต้น



รูปที่ 21 ตัวอย่างเมท็อดสำเนาโค้ด GroupSelector ระหว่าง JabRef เวอร์ชัน 1.8 และ 2.0 ก่อนทำการก่อแบบตัวเลือกสินทรัพย์ร่วม



รูปที่ 22 ตัวอย่างเมท็อดสำเนาโค้ด GroupSelector ระหว่าง JabRef เวอร์ชัน 1.8 และ 2.0 ซึ่งทำการแยกชิ้นโค้ดตามแบบอย่างเมท็อดแม่แบบเพื่อก่อแบบเป็นตัวเลือกสินทรัพย์ร่วม

5.3 บทวิเคราะห์ผลการทดลอง

ด้วยแนวทางการก่อแบบตัวเลือกสินทรัพย์ร่วมโดยใช้แนวทางการออกแบบเมท็อดแม่แบบ ทำให้สามารถแยกชิ้นส่วนโค้ดที่แตกต่างกันระหว่างผลิตภัณฑ์ ซึ่งแทรกอยู่กับโค้ดส่วนอื่นที่เหมือนกัน อย่างไรก็ตามเนื่องจากวิธีนี้ขึ้นกับการค้นหาสำเนาโค้ดเป็นหลัก ข้อจำกัดจึงอยู่ที่การกำหนดค่าขีดเริ่มที่เหมาะสม ซึ่งในการทดลองนี้ได้เลือกใช้ค่าขีดเริ่ม 0.3

สำหรับวิธีการก่อแบบตัวเลือกสินทรัพย์ร่วมที่นำเสนอนั้น ใช้การระบุส่วนแตกต่างระหว่างคู่สำเนาเมท็อดด้วยความละเอียดในระดับบรรทัดคำสั่ง จึงเหมาะสำหรับกรณีที่ความแตกต่างระหว่างเมท็อดที่ตรวจพบอยู่ในระดับชุดคำสั่ง ทำให้ความแตกต่างในระดับนิพจน์ถูกระบุรวมเป็นคำสั่งเดียว และมีข้อจำกัดในกรณีที่เมท็อดที่ตรวจพบมีอาร์กิวเมนต์ที่ต่างกัน ซึ่งผู้พัฒนาสินทรัพย์จำเป็นต้องพิจารณาและตัดสินใจเพิ่มเติม

บทที่ 6
บทสรุปและข้อเสนอแนะ

6.1 สรุปผลการวิจัย

งานวิจัยนี้นำเสนอวิธีการระบุตัวเลือกสินทรัพย์ร่วมจากผลิตภัณฑ์ที่มีอยู่ ซึ่งเป็นแนวทางการพัฒนาสินทรัพย์ในเชิงปฏิกิริยา โดยการประยุกต์ใช้เทคนิคการตรวจหาสำเนาโค้ดในการหาส่วนร่วม และใช้การก่อแบบเมท็อดแม่แบบในการแยกส่วนแปรผันออกจากส่วนร่วมที่ตรวจพบในกรณีที่สำเนาโค้ดมีความแตกต่างอยู่ ซึ่งขั้นตอนของวิธีการการระบุตัวเลือกสินทรัพย์ร่วมในงานวิจัยนี้ ถูกแบ่งออกเป็น 3 ขั้นตอนหลัก ได้แก่ ตรวจหาสำเนาโค้ด จำแนกประเภทผลสำเนาโค้ด และก่อแบบตัวเลือกสินทรัพย์ร่วม โดยในการตรวจหาสำเนาโค้ดได้ใช้เทคนิคการเปรียบเทียบเชิงข้อความด้วยเครื่องมือนิแค็ด เพื่อตรวจสอบระหว่างผลิตภัณฑ์ในระดับเมท็อด และได้ทำซ้ำโดยกำหนดตัวแปรทำให้ได้วิธีการตรวจหาสำเนาโค้ดหลายวิธี ทำให้ได้ผลสำเนาโค้ดที่เพียงพอสำหรับขั้นตอนการจำแนกประเภทผลสำเนาโค้ด เพื่อแยกเอาเฉพาะสำเนาโค้ดที่มีความแตกต่างออกมาและก่อแบบเป็นตัวเลือกสินทรัพย์ร่วมในขั้นตอนสุดท้าย

การจำแนกประเภทผลสำเนาโค้ดใช้พารามิเตอร์ในการตรวจหาสำเนาโค้ดของเครื่องมือนิแค็ด ซึ่งได้แก่ พารามิเตอร์ค่าขีดเริ่ม และการทำให้โค้ดเป็นบรรทัดฐานเดียวกัน โดยการเปลี่ยนชื่อตัวระบุ เพื่อทำให้ผลสำเนาโค้ดที่ตรวจพบในแต่ละวิธีการมีความแตกต่างกัน แล้วจึงหักลบผลลัพธ์ส่วนที่ทับซ้อนกันระหว่างวิธีการ ทำให้สามารถจำแนกประเภทผลสำเนาโค้ด จนได้สำเนาโค้ดประเภทที่ 3 ซึ่งมีการเพิ่ม ลบ หรือแก้ไขคำสั่งในชิ้นส่วนโค้ด ทำให้เกิดจุดแตกต่างระหว่างคู่สำเนาอย่างชัดเจน

ในขั้นตอนการก่อแบบตัวเลือกสินทรัพย์ร่วมซึ่งเป็นขั้นตอนสุดท้าย ด้วยแนวทางการออกแบบเมท็อดแม่แบบ คลาสแม่จะถูกสร้างขึ้นใหม่เพื่อแยกโค้ดส่วนที่เหมือนกันไว้ และเก็บส่วนที่แตกต่างไว้ในคลาสลูกสำหรับแต่ละผลิตภัณฑ์ โดยจุดที่แตกต่างในเมท็อดจะถูกระบุตำแหน่งด้วยขั้นตอนวิธีหาลำดับย่อยร่วมกันที่ยาวที่สุด เช่นเดียวกับโปรแกรมอรรถประโยชน์ diff แต่หน่วยย่อยที่สุดของความแตกต่างในที่นี้จะถูกระบุเป็นบรรทัดโค้ด เพื่อที่จะแทนที่จุดแตกต่างเหล่านั้นด้วยคำสั่งเรียกเมท็อดฮุคในการสร้างเมท็อดแม่แบบไว้ในคลาสแม่ ส่วนเมท็อดฮุคจะถูกเขียนด้วยชิ้นส่วนโค้ดที่แตกต่างสำหรับแต่ละผลิตภัณฑ์ซึ่งเก็บไว้ที่คลาสลูกของผลิตภัณฑ์นั้นๆ เพื่อโอเวอร์ไรด์เมท็อดฮุคในคลาสแม่อีกทีหนึ่ง

เนื่องจากการหาซอร์สโค้ดของสายผลิตภัณฑ์ซอฟต์แวร์เพื่อมาใช้ในการทดลองเป็นไปได้ยาก เพราะเป็นแนวคิดที่มักจะพบในโครงการเชิงพาณิชย์ซึ่งไม่เปิดเผยข้อมูล งานวิจัยนี้จึงได้ใช้ซอฟต์แวร์โอเพนซอร์สในการทดลองแนวคิดวิธีการ และประเมินผลเครื่องมือ โดยมองว่าซอฟต์แวร์ในแต่ละเวอร์ชันมีความคล้ายกัน เสมือนเป็นผลิตภัณฑ์ในสายผลิตภัณฑ์เดียวกัน และในขณะเดียวกันก็มีความแตกต่างที่เกิดจากพัฒนาซอฟต์แวร์ด้วย ซึ่งซอฟต์แวร์ที่นำซอร์สโค้ดมาใช้เป็นตัวอย่างทดลองนั้น คือ JabRef ซึ่งเป็นโปรแกรมสำหรับจัดการฐานข้อมูลบรรณานุกรม และได้เลือกมาเฉพาะเวอร์ชันหลัก ซึ่งมีขนาดของซอฟต์แวร์ประมาณ 10,000 ถึง 60,000 บรรทัดโค้ด

จากผลการทดลองดังกล่าวพบว่าวิธีการที่นำเสนอสามารถแยกชิ้นส่วนโค้ดที่แตกต่างกันระหว่างผลิตภัณฑ์ ซึ่งแทรกอยู่กับโค้ดส่วนอื่นที่เหมือนกันได้ อย่างไรก็ตาม ในการใช้งานจริงจำเป็นต้องปรับปรุงในหลายส่วน เช่น การทำให้รองรับการนำไปใช้ซ้ำในระดับที่สูงขึ้นไปกว่าระดับคลาส การเพิ่มความยืดหยุ่นในการใช้ซ้ำโดยวิเคราะห์จุดแตกต่างด้วยการแจงส่วนชุดคำสั่งให้ละเอียดลงไป

6.2 ข้อจำกัด

1) โค้ดของผลิตภัณฑ์ที่นำมาเปรียบเทียบ ใช้ได้เฉพาะกับภาษาจาวาเท่านั้น

2) เนื่องจากงานวิจัยนี้มุ่งเน้นไปที่การจัดการกับสำเนาโค้ดประเภทที่ 3 สำหรับสำเนาโค้ดประเภทที่ 2 จึงจำเป็นต้องมีการวิเคราะห์ต่อไปว่าเป็นเพียงการเปลี่ยนชื่อตัวระบุ หรือเป็นการอ้างอิงตัวระบุคนละตัวกัน

3) ไม่รองรับในกรณีที่จุดแตกต่างที่พบในสำเนาโค้ดเป็นข้อความสั่งควบคุมพิเศษ

4) ไม่รองรับในกรณีที่เมท็อดที่พบมีอาร์กิวเมนต์ที่แตกต่างกัน

6.3 แนวทางการวิจัยต่อ

1) การระบุตัวเลือกสินทรัพย์ร่วมสามารถทำในระดับที่ใหญ่ขึ้นไปกว่าระดับคลาส ซึ่งอาจต้องใช้ข้อมูลการออกแบบในการพิจารณาร่วมด้วย

2) การแทนที่จุดแตกต่างในสำเนาโค้ดเดียวกันด้วยคำสั่งเรียกเมท็อดฮุค ไม่จำเป็นต้องแทนที่ทั้งบรรทัด แต่ใช้การวิเคราะห์รูปแบบของโค้ดที่แตกต่าง เพื่อกำหนดการคืนค่าของเมท็อดฮุคที่เหมาะสมในการแทนที่โค้ดเพียงบางส่วน ทำให้เป็นอิสระต่อกันมากขึ้น

3) เมื่อทำการเปรียบเทียบระหว่างผลิตภัณฑ์ที่มากกว่า 1 คู่ จุดแตกต่างอาจจะมีมากขึ้น จนทำให้เมท็อดแม่แบบที่ถูกสร้างขึ้นอาจต้องเรียกเมท็อดฮุคเป็นจำนวนมากซึ่งอาจเป็นค่าใช้จ่ายที่มากจนเกินไป การวิเคราะห์ในระดับที่สูงขึ้นกว่าโค้ดจะทำให้ตัดจุดแตกต่างที่ไม่จำเป็นออกไปได้

รายการอ้างอิง

[1] Clements, P., and Northrop, L. Software product lines: practices and patterns, Addison-Wesley Longman Publishing, 2001.

[2] Mende, T., Beckwermert, F., Koschke, R., and Meier, G. Supporting the Grow-and-Prune Model in Software Product Lines Evolution Using Clone Detection, Proceedings of the 2008 12th European Conference on Software Maintenance and Reengineering, pp. 163-172, IEEE Computer Society, 2008.

[3] Faust, D., and Verhoef, C. Software product line migration and deployment, Software: Practice and Experience, vol. 33, no. 10, pp. 933-955, Aug. 2003.

[4] Roy, C.K., and Cordy, J.R. A Survey on Software Clone Detection Research, SCHOOL OF COMPUTING TR 2007-541, vol. 115, QUEEN'S UNIVERSITY, 2007.

[5] Koschke, R. Survey of Research on Software Clones, Duplication, Redundancy, and Similarity in Software, Koschke, R., Merlo, E., and Walenstein, A. Eds., Dagstuhl, Germany: Internationales Begegnungsund Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2007.

[6] Higo, Y., Sawa, K., and Kusumoto, S. Problematic Code Clones Identification Using Multiple Detection Results, 2009 16th Asia-Pacific Software Engineering Conference, pp. 365-372, Batu Ferringhi, Penang, Malaysia: 2009.

[7] Matinlassi, M. Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA, Proceedings of the 26th International Conference on Software Engineering, pp. 127-136, IEEE Computer Society, 2004.

[8] Lisboa, L.B., Garcia, V.C., Lucrédio, D., de Almeida E.S., de Lemos Meira, S.R., and de Mattos Fortes, R.P. A systematic review of domain analysis tools, Information and Software Technology, pp. 1-13, vol. 52, 2010.

[9] Baxter, I.D., Yahin, A., Moura, L., Sant'Anna, M., and Bier, L. Clone Detection Using Abstract Syntax Trees, Proceedings of the International Conference on Software Maintenance, p. 368, IEEE Computer Society, 1998.

[10] Kamiya, T., Kusumoto, S., and Inoue, K. CCFinder: a multilinguistic token-based code clone detection system for large scale source code, IEEE Transactions on. Software Engineering, vol. 28, 2002: pp. 654-670.

[11] Bellon, S., Koschke, R., Antoniol, G., Krinke, J., and Merlo, E. Comparison and Evaluation of Clone Detection Tools, IEEE Transactions on Software Engineering, vol. 33, 2007: pp. 577-591.

[12] Roy, C.K., Cordy, J.R., and Koschke, R. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach, Science of Computer Programming, vol. 74, 2009: pp. 470-495.

[13] Ducasse, S., Rieger, M., and Demeyer, S. A Language Independent Approach for Detecting Duplicated Code, Proceedings of the IEEE International Conference on Software Maintenance, pp. 109, IEEE Computer Society, 1999.

[14] Johnson, J.H. Identifying redundancy in source code using fingerprints, Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering, Vol. 1, pp. 171-183, Toronto, Ontario, Canada: IBM Press, 1993.

[15] Marcus, A., and Maletic, J.I. Identification of High-Level Concept Clones in Source Code, Proceedings of the 16th IEEE international conference on Automated software engineering, IEEE Computer Society, 2001, p. 107.

[16] Baker, B., On Finding Duplication and Near-Duplication in Large Software Systems, Second Proceeding of Working Conference on Reverse Engineering (WCRE95) , pp. 86-95, 1995.

[17] Basit, H.A., and Jarzabek, S. Efficient token based clone detection with flexible tokenization, Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering, pp. 513-516, Dubrovnik, Croatia: ACM, 2007.

[18] Lague, B., Proulx, D., Mayrand, J., Merlo, E.M., and Hudepohl, J. Assessing the Benefits of Incorporating Function Clone Detection in a Development Process,

Proceedings of the International Conference on Software Maintenance, p. 314, IEEE Computer Society, 1997.

[19] Kontogiannis, K.A., Demori, R., Merlo, E., Galler, M., and Bernstein, M. Pattern matching for clone and concept detection, Reverse engineering, pp. 77-108, Kluwer Academic Publishers, 1996.

[20] Mayrand, J., Leblanc, C., and Merlo, E. Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics, Proceedings of the 1996 International Conference on Software Maintenance, p. 244, IEEE Computer Society, 1996.

[21] Kontogiannis, K., DeMori, R., Bernstein, M., Galler, M., and Merlo, E. Pattern matching for design concept localization, Proceedings of 2nd Working Conference on Reverse Engineering, pp. 96-103, Toronto, Ont., Canada, 1995.

[22] Lucca, G.A.D., Penta, M.D., and Fasolino, A.R. An Approach to Identify Duplicated Web Pages, Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment, pp. 481-486, IEEE Computer Society, 2002.

[23] Lanubile, F., and Mallardo, T. Finding Function Clones in Web Applications, Proceedings of the Seventh European Conference on Software Maintenance and Reengineering, pp. 379, IEEE Computer Society, 2003.

[24] Koschke, R., Falke, R., and Frenzel, P. Clone Detection Using Abstract Syntax Suffix Trees, Proceedings of the 13th Working Conference on Reverse Engineering, pp. 253-262, IEEE Computer Society, 2006.

[25] Leitão, A.M. Detection of Redundant Code Using R2D2, Software Quality Control, vol. 12, pp. 361-382, 2004.

[26] Wahler, V., Seipel, D., Gudenberg, J.W.V., and Fischer, G. Clone Detection in Source Code by Frequent Itemset Techniques, Proceedings of the Source Code Analysis and Manipulation, pp. 128-135, Fourth IEEE International Workshop, IEEE Computer Society, 2004.

[27] Li, Z., Lu, S., Myagmar, S., and Zhou, Y. CP-Miner: a tool for finding copy-paste and related bugs in operating system code, Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation, Vol. 6, pp. 176-192, San Francisco, CA: USENIX Association, 2004.

[28] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. Design patterns: elements of reusable object-oriented software. Addison-Wesley: 1995.

[29] Roy, C. K., and Cordy, J.R., NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization, The 16th IEEE International Conference on Program Comprehension, 2008 (ICPC 2008), pp. 172–181, 2008.

[30] Cordy, J.R., The TXL Source Transformation Language, Science of Computer Programming, pp. 190-210, August 2006.

[31] Jiang, L., Misherghi, G., Su, Z., and Glondu, S. DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones, 29th International Conference on Software Engineering (ICSE'07), pp. 96-105, May 2007.

[32] Myers, E. W. An O(ND) difference algorithm and its variations, Algorithmica, vol. 1, no. 1–4, pp. 251-266, Nov. 1986.

[33] Open source software [Computer software]. Available from : http://sourceforge.net [2012, April 18]

ภาคผนวก

ภาคผนวก ก.
จำนวนเมท็อดสำเนาโค้ดที่ตรวจพบใน JabRef แต่ละเวอร์ชัน

เนื้อหาในส่วนนี้ประกอบด้วยตารางซึ่งแสดงถึงจำนวนเมท็อดสำเนาโค้ดที่ตรวจพบระหว่าง JabRef แต่ละเวอร์ชัน โดย

- จำนวนสำเนาโค้ดประเภทที่ 1 ที่ตรวจพบถูกแสดงไว้ในตารางที่ 17
- จำนวนสำเนาโค้ดประเภทที่ 2 ที่ตรวจพบถูกแสดงไว้ในตารางที่ 18
- จำนวนสำเนาโค้ดประเภทที่ 3 ที่ตรวจพบถูกแสดงในตารางที่ 19 ถึงตารางที่ 25 โดยแยกตามค่าขีดเริ่มที่ใช้สำหรับวิธีการ C ตั้งแต่ 0.1 ถึง 0.7 ตามลำดับ

ตารางที่ 17 จำนวนเมท็อดสำเนาโค้ดประเภทที่ 1 ที่ตรวจพบ

Ver	1.0	1.1	1.2	1.3.1	1.4	1.5	1.6	1.7	1.8	2.0	2.1	2.2	2.3	2.4	2.5	2.6	2.7
1.0																	
1.1	319																
1.2	217	306															
1.3.1	182	257	457														
1.4	172	245	422	541													
1.5	140	206	324	411	558												
1.6	112	168	257	332	455	522											
1.7	89	127	206	264	381	428	677										
1.8	73	106	179	231	345	381	603	880									
2.0	66	97	162	213	318	346	534	751	986								
2.1	62	90	147	188	286	305	464	647	831	1227							
2.2	59	86	142	182	274	287	437	606	761	1105	1378						
2.3	57	84	137	176	268	281	424	585	734	1058	1285	1731					
2.4	39	61	99	122	184	193	296	410	508	720	853	1129	1346				
2.5	39	60	98	119	180	191	291	405	503	707	820	1083	1276	2125			
2.6	39	57	95	116	176	186	276	389	487	675	782	1042	1221	2005	2217		
2.7	39	57	95	111	171	181	269	380	478	661	763	1021	1190	1960	2155	2359	

ตารางที่ 18 จำนวนเมท็อดสำเนาโค้ดประเภทที่ 2 ที่ตรวจพบ

Ver	1.0	1.1	1.2	1.3.1	1.4	1.5	1.6	1.7	1.8	2.0	2.1	2.2	2.3	2.4	2.5	2.6	2.7
1.0																	
1.1	63																
1.2	42	69															
1.3.1	42	61	70														
1.4	40	59	75	51													
1.5	39	57	56	64	65												
1.6	30	49	52	62	70	79											
1.7	30	38	50	60	58	70	80										
1.8	27	35	39	59	55	64	83	93									
2.0	29	35	41	48	65	70	96	106	117								
2.1	27	37	43	45	72	71	102	116	131	152							
2.2	25	35	47	52	61	68	86	102	122	153	117						
2.3	29	39	45	54	67	69	87	115	127	148	124	146					
2.4	22	28	35	32	52	57	67	82	100	101	121	110	137				
2.5	22	29	27	41	48	48	71	95	95	104	119	123	153	167			
2.6	20	27	36	34	56	61	77	81	98	108	133	133	149	172	149		
2.7	23	31	39	38	46	67	79	84	91	107	129	137	149	168	164	177	

ตารางที่ 19 จำนวนเมท็อดสำเนาโค้ดประเภทที่ 3 ที่ตรวจพบ โดยวิธีการ C ใช้ค่าขีดเริ่ม 0.1

Ver	1.0	1.1	1.2	1.3.1	1.4	1.5	1.6	1.7	1.8	2.0	2.1	2.2	2.3	2.4	2.5	2.6	2.7
1.0																	
1.1	97																
1.2	77	102															
1.3.1	78	99	108														
1.4	68	86	115	83													
1.5	59	74	83	86	77												
1.6	45	60	73	77	85	91											
1.7	32	38	62	70	69	81	108										
1.8	26	33	47	66	66	71	109	146									
2.0	31	36	56	56	73	76	125	162	183								
2.1	31	39	54	50	79	72	122	173	196	220							
2.2	27	37	56	57	61	68	99	147	170	220	175						
2.3	29	36	51	54	68	62	99	161	185	218	207	245					
2.4	24	31	50	50	79	81	104	142	192	245	265	320	399				
2.5	22	30	40	56	73	73	111	157	188	243	248	310	391	236			
2.6	20	27	47	49	81	85	116	138	183	238	250	310	383	259	214		
2.7	20	28	47	49	68	88	118	140	174	236	243	312	379	262	251	245	

ตารางที่ 20 จำนวนเมท็อดสำเนาโค้ดประเภทที่ 3 ที่ตรวจพบ โดยวิธีการ C ใช้ค่าขีดเริ่ม 0.2

Ver	1.0	1.1	1.2	1.3.1	1.4	1.5	1.6	1.7	1.8	2.0	2.1	2.2	2.3	2.4	2.5	2.6	2.7
1.0																	
1.1	118																
1.2	129	134															
1.3.1	127	143	142														
1.4	119	128	161	130													
1.5	98	94	123	139	129												
1.6	87	84	125	136	149	151											
1.7	61	67	115	134	139	144	168										
1.8	53	63	110	123	136	141	183	228									
2.0	55	63	133	138	164	157	208	276	294								
2.1	52	63	129	131	164	152	216	299	326	319							
2.2	49	62	130	138	157	151	203	276	313	338	268						
2.3	51	62	123	129	155	141	198	278	319	343	316	346					
2.4	49	60	125	136	181	178	235	321	391	471	490	628	710				
2.5	47	59	120	136	174	171	233	317	378	463	479	626	721	337			
2.6	45	57	123	137	176	177	237	308	370	451	477	625	729	390	335		
2.7	45	58	122	130	168	175	236	309	363	450	474	622	719	412	374	348	

ตารางที่ 21 จำนวนเมท็อดสำเนาโค้ดประเภทที่ 3 ที่ตรวจพบ โดยวิธีการ C ใช้ค่าขีดเริ่ม 0.3

Ver	1.0	1.1	1.2	1.3.1	1.4	1.5	1.6	1.7	1.8	2.0	2.1	2.2	2.3	2.4	2.5	2.6	2.7
1.0																	
1.1	134																
1.2	154	156															
1.3.1	156	158	173														
1.4	152	150	190	158													
1.5	133	130	154	165	166												
1.6	124	120	161	179	202	203											
1.7	89	102	152	180	201	207	226										
1.8	86	102	149	172	200	204	247	302									
2.0	89	103	176	198	235	227	286	367	378								
2.1	79	95	168	183	230	220	290	391	410	404							
2.2	76	94	167	186	217	213	273	359	404	432	351						
2.3	77	92	159	178	212	199	267	367	417	450	402	440					
2.4	71	90	164	185	245	240	314	420	504	606	636	788	870				
2.5	68	90	160	187	240	234	315	419	495	605	635	807	897	450			
2.6	66	87	162	186	240	238	321	412	494	597	637	805	914	517	442		
2.7	66	88	161	179	235	239	322	413	491	598	641	812	913	551	501	463	

ตารางที่ 22 จำนวนเมท็อดสำเนาโค้ดประเภทที่ 3 ที่ตรวจพบ โดยวิธีการ C ใช้ค่าขีดเริ่ม 0.4

Ver	1.0	1.1	1.2	1.3.1	1.4	1.5	1.6	1.7	1.8	2.0	2.1	2.2	2.3	2.4	2.5	2.6	2.7
1.0																	
1.1	171																
1.2	201	205															
1.3.1	193	209	221														
1.4	190	204	242	208													
1.5	181	186	211	220	235												
1.6	179	178	232	246	285	287											
1.7	145	161	227	255	289	307	316										
1.8	139	157	215	241	280	292	332	403									
2.0	150	168	246	276	328	317	382	485	481								
2.1	138	155	233	260	319	310	387	506	529	526							
2.2	131	151	230	261	306	297	366	485	523	560	456						
2.3	133	149	221	251	302	286	366	486	533	580	525	583					
2.4	126	147	225	255	326	322	416	555	654	767	788	971	1058				
2.5	125	149	223	258	326	319	416	555	643	771	785	988	1094	616			
2.6	120	144	223	255	324	321	423	550	641	761	793	1000	1117	697	625		
2.7	118	143	221	243	314	315	422	547	632	766	795	1014	1117	745	683	639	

ตารางที่ 23 จำนวนเมท็อดสำเนาโค้ดประเภทที่ 3 ที่ตรวจพบ โดยวิธีการ C ใช้ค่าขีดเริ่ม 0.5

Ver	1.0	1.1	1.2	1.3.1	1.4	1.5	1.6	1.7	1.8	2.0	2.1	2.2	2.3	2.4	2.5	2.6	2.7
1.0																	
1.1	201																
1.2	243	266															
1.3.1	257	280	281														
1.4	258	285	309	271													
1.5	234	261	274	279	303												
1.6	235	258	312	321	369	363											
1.7	210	235	295	321	373	389	413										
1.8	201	231	299	317	373	385	436	516									
2.0	227	257	340	362	432	422	493	601	615								
2.1	219	244	325	347	426	412	505	633	660	674							
2.2	213	242	327	346	412	400	490	629	669	718	618						
2.3	216	241	317	337	407	381	495	632	683	752	707	809					
2.4	205	229	314	339	437	421	544	708	806	933	964	1186	1268				
2.5	205	235	314	347	432	420	547	708	798	946	967	1203	1312	850			
2.6	201	228	308	340	428	420	549	706	801	946	986	1225	1342	930	866		
2.7	199	227	310	330	425	415	556	705	792	958	991	1241	1345	990	931	898	

ตารางที่ 24 จำนวนเมท็อดสำเนาโค้ดประเภทที่ 3 ที่ตรวจพบ โดยวิธีการ C ใช้ค่าขีดเริ่ม 0.6

Ver	1.0	1.1	1.2	1.3.1	1.4	1.5	1.6	1.7	1.8	2.0	2.1	2.2	2.3	2.4	2.5	2.6	2.7
1.0																	
1.1	236																
1.2	280	295															
1.3.1	302	317	332														
1.4	332	356	389	355													
1.5	288	314	335	335	358												
1.6	300	327	382	388	438	435											
1.7	282	310	374	397	444	457	497										
1.8	275	308	372	393	450	459	525	604									
2.0	318	350	428	456	526	507	597	706	715								
2.1	311	341	416	444	524	509	613	744	762	786							
2.2	311	346	418	447	524	509	608	745	781	834	742						
2.3	317	352	417	446	526	504	618	763	809	892	850	973					
2.4	320	354	419	451	555	537	673	842	937	1067	1101	1329	1402				
2.5	321	360	423	457	554	539	682	851	945	1086	1107	1351	1451	1051			
2.6	316	356	423	460	556	543	689	854	951	1094	1131	1380	1485	1133	1073		
2.7	324	361	430	459	571	554	702	867	962	1120	1157	1418	1516	1210	1157	1119	

ตารางที่ 25 จำนวนเมท็อดสำเนาโค้ดประเภทที่ 3 ที่ตรวจพบ โดยวิธีการ C ใช้ค่าขีดเริ่ม 0.7

Ver	1.0	1.1	1.2	1.3.1	1.4	1.5	1.6	1.7	1.8	2.0	2.1	2.2	2.3	2.4	2.5	2.6	2.7
1.0																	
1.1	309																
1.2	364	384															
1.3.1	411	427	457														
1.4	620	636	700	684													
1.5	406	422	462	468	504												
1.6	449	475	533	546	605	584											
1.7	489	516	582	606	665	655	719										
1.8	518	544	611	630	696	689	783	870									
2.0	615	648	721	745	837	788	901	1026	1023								
2.1	613	642	718	736	832	794	915	1044	1056	1099							
2.2	651	683	757	775	878	834	962	1103	1128	1200	1130						
2.3	692	724	788	812	923	865	1018	1162	1199	1294	1267	1424					
2.4	712	738	819	839	977	917	1090	1237	1323	1444	1468	1708	1742				
2.5	731	765	845	861	999	941	1120	1269	1352	1487	1511	1766	1822	1563			
2.6	749	788	873	892	1029	970	1156	1302	1385	1528	1556	1820	1888	1664	1622		
2.7	780	819	901	922	1067	1003	1196	1347	1429	1580	1611	1879	1945	1760	1732	1693	

ประวัติผู้เขียนวิทยานิพนธ์

นายธนิต เหรียญตระกูล เกิดเมื่อวันที่ 18 กรกฎาคม พ.ศ. 2526 ที่จังหวัดกรุงเทพมหานคร สำเร็จการศึกษาหลักสูตรวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมอิเล็กทรอนิกส์ จากภาควิชาอิเล็กทรอนิกส์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ในปีการศึกษา 2547 และเข้าศึกษาต่อในหลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมซอฟต์แวร์ ที่ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2551