การตรวจหาร่องรอยที่ผิดพลาดของโปรแกรมที่เก็บอยู่ในฐานข้อมูล

นางสาวสุทธิกานต์ เนาวรัตน์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2559

Detection of Stored Procedure Bad Smells

Miss Sutthikan Naowarat

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 2016

หัวข้อวิทยานิพนธ์	การตรวจหาร่องรอยที่ผิดพลาดของโปรแกรมที่เก็บอยู่ในฐานข้อมูล
โดย	นางสาวสุทธิกานต์ เนาวรัตน์
สาขาวิชา	วิศวกรรมซอฟต์แวร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร. พรศิริ หมื่นไชยศรี

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

.....คณบดีคณะวิศวกรรมศาสตร์

(รองศาสตราจารย์ ดร. สุพจน์ เตชวรสินสกุล)

คณะกรรมการสอบวิทยานิพนธ์

.....ประธานกรรมการ

(รองศาสตราจารย์ ดร. วิวัฒน์ วัฒนาวุฒิ)

.....อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

(รองศาสตราจารย์ ดร. พรศิริ หมื่นไชยศรี)

.....กรรมการ

(รองศาสตราจารย์ ดร. ทวิตีย์ เสนีวงศ์ ณ อยุธยา)

.....กรรมการภายนอกมหาวิทยาลัย

(ผู้ช่วยศาสตราจารย์ ดร. มฑุปายาส ทองมาก)

สุทธิกานต์ เนาวรัตน์ : การตรวจหาร่องรอยที่ผิดพลาดของโปรแกรมที่เก็บอยู่ในฐานข้อมูล (Detection of Stored Procedure Bad Smells) อ.ที่ปรึกษาวิทยานิพนธ์หลัก: รศ. ดร. พรศิริ หมื่นไชยศรี, 85 หน้า.

โปรแกรมที่เก็บอยู่ในฐานข้อมูลจะใช้เพื่อการเข้าถึงข้อมูลและการจัดการข้อมูลในระบบที่มีขนาดใหญ่เพื่อเพิ่มประสิทธิภาพให้กับการสืบค้นข้อมูล ลดปริมาณงานระหว่างโปรแกรมประยุกต์และฐานข้อมูล และลดปัญหาปริมาณการเชื่อมต่อที่มากระหว่างโปรแกรมประยุกต์และฐานข้อมูล ดังนั้น ถ้าซอร์ซโค้ดของโปรแกรมที่เก็บอยู่ในฐานข้อมูลมีร่องรอยที่ผิดพลาดเกิดขึ้น จะส่งผลกระทบเมื่อต้องมีการเปลี่ยนแปลงหรือแก้ไขซอร์ซโค้ด และในที่สุดก็มีผลเสียต่อคุณภาพของระบบและความสามารถในการบำรุงรักษาของโปรแกรมที่เก็บอยู่ในฐานข้อมูล

งานวิจัยนี้นำเสนอวิธีการในการตรวจหาร่องรอยที่ผิดพลาดของโปรแกรมที่เก็บอยู่ในฐานข้อมูลด้วยการใช้แผนภาพต้นไม้และการวิเคราะห์บริบท สำหรับร่องรอยที่ผิดพลาด 6 ประเภท โดยวิธีการใช้แผนภาพต้นไม้นั้นจะเป็นการเปรียบเทียบโครงสร้างของแผนภาพต้นไม้ของร่องรอยที่ผิดพลาดกับแผนภาพต้นไม้ของซอร์ซโค้ด ซึ่งซอร์ซโค้ดที่ใช้จะพัฒนาด้วยภาษา PL/SQL ส่วนการใช้วิธีการวิเคราะห์บริบท คือการสร้างข้อกำหนดและคุณสมบัติให้กับร่องรอยที่ผิดพลาด เพื่อเพิ่มความแม่นยำในการตรวจหาร่องรอยที่ผิดพลาด นอกจากนี้งานวิจัยนี้ได้มีการอธิบายกระบวนการของภาพรวมของระบบ กระบวนการอัลกอริทึม และใช้ซอร์ซโค้ดสำหรับการทดลอง โดยการประเมินความสามารถของวิธีการที่นำเสนอนี้จะใช้ค่าดัชนีความสามารถในการบำรุงรักษาเพื่อแสดงให้เห็นว่าวิธีการที่นำเสนอนี้มีประสิทธิผล

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต .....

สาขาวิชา วิศวกรรมซอฟต์แวร์ ลายมือชื่อ อ.ที่ปรึกษาหลัก .....

ปีการศึกษา 2559

5670980421 : MAJOR SOFTWARE ENGINEERING
KEYWORDS: STORED PROCEDURES / BAD SMELLS DETECTION / TREE DIAGRAM / CONTEXT ANALYSIS / MAINTAINABILITY INDEX

SUTTHIKAN NAOWARAT: Detection of Stored Procedure Bad Smells. ADVISOR: ASSOC. PROF. PORNSIRI MUENCHAISRI, Ph.D., 85 pp.

Stored procedures are commonly used for accessing and manipulating data in large-scale system development to optimize the database query, reduce the application workloads and reduce the traffic problems between the database and the application. If the source code of stored procedures has bad smells, it will have the impact on modification and eventually, have a negative impact on their quality and maintainability.

This research proposes Tree Diagram and Context Analysis approach in detecting six different bad smells of stored procedures. The tree diagram approach is the comparison tree diagram of bad smells and source code which is written in PL/SQL. The context analysis approach is the creation of rules and qualifications of bad smells for increasing the accuracy in detection. In addition, this research explains the overview process, the algorithm process, and uses example source code. The research uses MI (Maintainability Index) to evaluate the effectiveness of the approach.

Department:	Computer Engineering	Student's Signature	.....
Field of Study:	Software Engineering	Advisor's Signature	.....
Academic Year:	2016		

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้ สำเร็จลุล่วงไปได้ด้วยความช่วยเหลือและเมตตาจาก รศ.ดร.พรศิริ หมื่นไชยศรี อาจารย์ที่ปรึกษาวิทยานิพนธ์ ขอกราบขอบพระคุณอาจารย์ที่สละเวลาให้คำปรึกษา ให้ความรู้และให้คำแนะนำต่างๆ เกี่ยวกับงานวิจัย ตลอดช่วงระยะเวลาการทำวิจัยนี้

ขอกราบขอบพระคุณ รศ.ดร.วิวัฒน์ วัฒนาวุฒิ ประธานกรรมการสอบ รศ.ดร.ทวิตีย์ เสนีวงศ์ ณ อยุธยา และ ผศ.ดร.มฑุปายาส ทองมาก กรรมการสอบวิทยานิพนธ์ ที่ได้สละเวลาในการตรวจสอบความถูกต้องสมบูรณ์ของวิทยานิพนธ์และให้คำแนะนำต่างๆ ในการสอบวิทยานิพนธ์

ขอขอบคุณคณาจารย์ทุกท่านในภาควิศวกรรมคอมพิวเตอร์ที่ให้ความรู้ซึ่งมีส่วนสำคัญที่สามารถนำมาใช้ประโยชน์ในการทำงานวิจัยนี้

ขอบคุณเพื่อนๆ พี่ๆ น้องๆ สำหรับกำลังใจและความช่วยเหลือที่มีให้กันตลอดระยะเวลาที่ศึกษาและทำงานวิจัยนี้

ท้ายที่สุดนี้ ขอกราบขอบพระคุณ คุณพ่อ คุณแม่ และคนในครอบครัวที่คอยเป็นกำลังใจและห่วงใยกันมาตลอดจนงานวิจัยได้สำเร็จลุล่วงได้ด้วยดี

สารบัญ

หน้า

หน้า

สารบัญรูปภาพ

หน้า

สารบัญตาราง

หน้า

บทที่ 1
บทนำ

1.1 ที่มาและความสำคัญของปัญหา

โปรแกรมที่เก็บอยู่ในฐานข้อมูล หรือ สโตร์โพรซีเยอร์ (Stored Procedure) เป็นที่นิยมใช้เพื่อการเข้าถึงข้อมูลและการจัดการข้อมูลในฐานข้อมูล โดยโปรแกรมเหล่านี้มีผลกับความถูกต้องของข้อมูลที่ต้องนำไปใช้งาน ดังนั้นถ้าหากมีลักษณะของการเขียนโค้ดที่ไม่ดีหรือมีลักษณะที่อาจจะส่งผลให้เกิดความผิดพลาดในอนาคตได้นั้น จะส่งผลให้การจัดการเกี่ยวกับข้อมูลมีความผิดพลาด ทำให้ได้รับข้อมูลจากการสืบค้นที่ผิดพลาดตามมาด้วยเช่นกัน ทำให้เกิดความเสียหายจากการนำข้อมูลที่ผิดพลาดนั้นไปใช้งาน

ลักษณะการเขียนโค้ดสำหรับสโตร์โพรซีเยอร์ที่ไม่ดีหรือไม่มีประสิทธิภาพของผู้พัฒนาซอฟต์แวร์ จะทำให้เกิดปัญหาร่องรอยที่ผิดพลาด (Bad Smells) ตัวอย่างของซอร์ซโค้ดที่มีลักษณะร่องรอยที่ผิดพลาด เช่น โค้ดซ้ำซ้อนกันหลายตำแหน่งในโปรแกรม การนำส่วนของซอร์ซโค้ดที่เกี่ยวข้องกับตรรกะทางธุระกิจหรือมีความสำคัญไปไว้ในส่วนของข้อยกเว้น (Exception) เป็นต้น ซอร์ซโค้ดที่มีลักษณะร่องรอยที่ผิดพลาดนั้นถึงแม้ว่าจะสามารถทำงานได้ถูกต้องตามความต้องการของผู้ใช้งาน แต่จะพบปัญหาจากซอร์ซโค้ดลักษณะนี้เมื่อเราต้องการปรับแก้ไขซอร์ซโค้ด หรือเพิ่มเติมการทำงานบางอย่างเข้าไปในซอร์ซโค้ด เนื่องจากการเขียนโค้ดที่ไม่ดีหรือไม่มีประสิทธิภาพนี้จึงเป็นสาเหตุของปัญหาต่างๆ ตามมา เช่น ความยากในการเข้าใจซอร์ซโค้ดว่าทำงานอย่างไร ความยากในการปรับแก้ไขซอร์ซโค้ดเมื่อพบข้อผิดพลาด ความยากในการเพิ่มเติมส่วนขยายจากซอร์ซโค้ดเดิม ความยากเมื่อต้องการนำซอร์ซโค้ดเดิมกลับมาใช้อีกครั้ง เป็นต้น จากปัญหาต่างๆ ของซอร์ซโค้ดลักษณะนี้ ส่งผลให้สโตร์โพรซีเยอร์มีความยากต่อการบำรุงรักษา (Maintainability) [1] และเพื่อต้องการให้สโตร์โพรซีเยอร์มีลักษณะง่ายต่อการบำรุงรักษาจึงควรมีการค้นหาและกำจัดร่องรอยที่ผิดพลาดที่เกิดขึ้นจากการเขียนโค้ดลักษณะนั้น โดยภายหลังจากที่กำจัดร่องรอยที่ผิดพลาดออกแล้วการทำงานของสโตร์โพรซีเยอร์ยังคงเหมือนเดิมก่อนการกำจัดร่องรอยที่ผิดพลาด ซึ่งวิธีการที่นำมาช่วยในการกำจัดร่องรอยที่ผิดพลาด คือ รีแฟคทอริง (Refactoring) [2]

ในปัจจุบันมีงานวิจัยจำนวนมากที่ศึกษาวิธีในการค้นหาร่องรอยที่ผิดพลาดในระดับโค้ด[3] และได้พัฒนาออกมาเป็นเครื่องมือ[4] เพื่อให้ความสะดวกแก่ผู้พัฒนาระบบสำหรับค้นหาร่องรอยที่ผิดพลาด แต่งานวิจัยและเครื่องมือดังกล่าวส่วนใหญ่ออกแบบและพัฒนาสำหรับโปรแกรมเชิงวัตถุ (Object Oriented Programming) แต่การค้นหาร่องรอยที่ผิดพลาดนั้นก็มีความจำเป็นสำหรับโปรแกรมเชิงคำสั่ง (Procedural Language) อย่างเช่น ภาษาพีแอล/เอสคิวแอล (PL/SQL) ด้วย

เช่นกัน โดยภาษา พีแอล/เอสคิวแอล นั้น เป็นภาษาที่ออราเคิล (Oracle) พัฒนาขึ้นเพื่อเพิ่มประสิทธิภาพในการจัดการและการสืบค้นข้อมูลในฐานข้อมูล และเป็นภาษาที่ใช้ในการเขียนสโตร์โพรซีเยอร์สำหรับฐานข้อมูลนี้ และเนื่องจากสโตร์โพรซีเยอร์ที่พัฒนาขึ้นมานั้นมีวัตถุประสงค์ส่วนใหญ่คือการจัดการข้อมูลหรือการสืบค้นข้อมูล จึงมีความจำเป็นที่ควรจะต้องมีลักษณะของโค้ดที่ดี เพื่อให้การจัดการข้อมูลและการสืบค้นข้อมูลเป็นไปอย่างมีประสิทธิภาพ ส่งผลให้ได้ข้อมูลที่มีคุณภาพเพื่อนำไปใช้งานในด้านต่างๆ โดยปัจจุบันมีงานวิจัยที่เกี่ยวกับการแนะนำการปรับปรุงสโตร์โพรซีเยอร์[5] [6] หรืองานวิจัยที่เกี่ยวกับการค้นหาร่องรอยที่ผิดพลาดมากมาย และการพัฒนาออกมาเป็นเครื่องมือในการค้นหาร่องรอยที่ผิดพลาด แต่ส่วนใหญ่เพื่อโปรแกรมเชิงวัตถุ

งานวิจัยนี้มีจุดมุ่งหมายเพื่อตรวจหาร่องรอยที่ผิดพลาดของโปรแกรมที่เก็บอยู่ในฐานข้อมูลที่พัฒนาด้วยภาษา พีแอล/เอสคิวแอล เพื่อเพิ่มความสามารถสำหรับการตรวจหาร่องรอยที่ผิดพลาดสำหรับโปรแกรมเชิงคำสั่ง เพื่อส่งเสริมให้ซอร์ซโค้ดมีลักษณะง่ายต่อการบำรุงรักษา สามารถเข้าใจได้ง่าย และทำการปรับแก้ไขหรือเพิ่มเติมซอร์ซโค้ดได้ง่าย

1.2 วัตถุประสงค์

เพื่อนำเสนอวิธีการสำหรับการตรวจหาร่องรอยที่ผิดพลาดและพัฒนาเครื่องมือที่ใช้ในการตรวจหาร่องรอยที่ผิดพลาดสำหรับโปรแกรมที่เก็บอยู่ในฐานข้อมูล

1.3 ขอบเขตงานวิจัย

1. วิธีการที่นำเสนอเพื่อตรวจหาร่องรอยที่ผิดพลาด คือ
   1) แผนภาพต้นไม้ (Tree Diagram)
   2) การวิเคราะห์บริบท (Context Analysis)
2. งานวิจัยนี้จะไม่ตรวจสอบความถูกต้องของวากยสัมพันธ์ (Syntax) หรือ อรรถศาสตร์ (Semantic) ของซอร์ซโค้ด โดยซอร์ซโค้ดที่นำมาใช้สำหรับงานวิจัยนี้จะต้องมีความถูกต้องทางวากยสัมพันธ์หรือผ่านการคอมไพล์ (Compile) แล้ว
3. ร่องรอยที่ผิดพลาดที่ใช้ในงานวิจัยมีทั้งหมด 6 ประเภท คือ
   1) Business Logic in Exception Sections
   2) Using cursor loops with DML
   3) Initialize and Cleanup Logic in Execution Section
   4) Use Literals
   5) No Range Values in Numeric FOR Loop

6) Not Using SUBTYPES and Anchored Types

4. ลักษณะของซอร์ซโค้ดที่จะใช้ทำการทดลองนั้น จะอยู่ในรูปแบบของโพรซีเยอร์ หรือฟังก์ชันเท่านั้น
5. ซอร์ซโค้ดที่จะใช้ในการทดลองเขียนด้วยภาษา พีแอล/เอสคิวแอล เท่านั้น
6. พัฒนาเครื่องมือและทดสอบบนระบบปฏิบัติการวินโดวส์ (Windows)

1.4 ขั้นตอนและวิธีดำเนินการวิจัย

1. ศึกษางานวิจัยเกี่ยวกับวิธีการของแผนภาพต้นไม้และการวิเคราะห์บริบท เพื่อนำไปประยุกต์ใช้ในการตรวจหาร่องรอยที่ผิดพลาดสำหรับโปรแกรมที่เก็บอยู่ในฐานข้อมูล
2. วิเคราะห์คุณสมบัติของร่องรอยที่ผิดพลาดทั้ง 6 ประเภท เพื่อหาคุณลักษณะเฉพาะสำหรับการสร้างแผนภาพต้นไม้และการวิเคราะห์บริบท
3. ออกแบบและพัฒนาเครื่องมือสำหรับตรวจหาร่องรอยที่ผิดพลาดสำหรับโปรแกรมที่เก็บอยู่ในฐานข้อมูล
4. ทำการทดลองใช้เครื่องมือ และเก็บรวบรวมข้อมูลที่ได้จากการทดลอง
5. วิเคราะห์ข้อมูลและสรุปผลการทดลอง
6. จัดทำรายงานวิทยานิพนธ์

1.5 ประโยชน์ที่คาดว่าจะได้รับ

สามารถนำวิธีการที่นำเสนอในการตรวจหาร่องรอยที่ผิดพลาดและเครื่องมือที่ได้พัฒนานี้ตรวจหาร่องรอยที่ผิดพลาดเพื่อเพิ่มประสิทธิภาพของโปรแกรมที่เก็บอยู่ในฐานข้อมูล

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ทฤษฎีที่เกี่ยวข้อง

งานวิจัยนี้ใช้แผนภาพต้นไม้และการวิเคราะห์บริบทในการตรวจหาร่องรอยที่ผิดพลาดของโปรแกรมที่เก็บอยู่ในฐานข้อมูล โดยมีทฤษฎีที่เกี่ยวข้องดังนี้

2.1.1 ร่องรอยที่ผิดพลาด (Bad Smells) [2]

ร่องรอยที่ผิดพลาด หมายถึง ลักษณะของซอร์ซโค้ดที่อาจก่อให้เกิดข้อผิดพลาดขึ้นได้ โดยอาจจะเกิดจากความผิดพลาดในกระบวนการออกแบบหรือในขั้นตอนของการพัฒนาซอฟต์แวร์ ซึ่งทำให้ระบบมีความยากต่อการบำรุงรักษา และอาจส่งผลทำให้ระบบนั้นทำงานไม่ถูกต้องในภายหลัง โดยการแก้ปัญหาเกี่ยวกับร่องรอยที่ผิดพลาดนั้นทำได้หลายกระบวนการ เช่น การออกแบบโครงสร้างใหม่หรือการใช้วิธีรีแฟคทอริงในการแก้ไขปัญหา โดยมาร์ติน ฟาวเลอร์ได้นำเสนอชนิดของร่องรอยที่ผิดพลาดทั้งหมด 22 ประเภท มีรายละเอียดดังนี้

1) Duplicated Code
   มีซอร์ซโค้ดเขียนซ้ำกันหลายๆจุด
2) Long Method
   เมทธอดที่มีขนาดใหญ่ เมธอดมีความยาวมาก
3) Large Class
   คลาสมีขนาดที่ใหญ่สามารถทำงานได้หลายๆอย่าง อยู่ในคลาสเดียว
4) Long parameter list
   การรับหรือส่งพารามิเตอร์ที่มีขนาดยาวเกินไป ทำให้เข้าใจได้ยาก
5) Divergent Change
   คลาสหนึ่งรองรับการเปลี่ยนแปลงหลายๆ อย่างที่เกิดขึ้น
6) Shotgun Surgery
   เมื่อมีการเพิ่มฟังก์ชันหนึ่ง ต้องทำการแก้ไขในหลายๆ คลาส ดังนั้นความสัมพันธ์ระหว่างคลาสและการเปลี่ยนแปลงใดๆ ควรเป็นหนึ่งต่อหนึ่ง
7) Feature Envy
   คลาสที่มีการเรียกใช้เมทธอดหรือตัวแปรของคลาสอีกคลาสหนึ่งบ่อยครั้ง ดังนั้นควรจะนำเมทธอดหรือตัวแปรของคลาสนั้นมาเก็บไว้ที่คลาสตัวเอง

8) Data Clumps

เป็นลักษณะของข้อมูลตัวเดียวกันที่กระจัดกระจายอยู่หลายคลาส ซึ่งควรจะรวมข้อมูลเหล่านี้เข้าด้วยกันและสร้างเป็นคลาสฯ หนึ่ง

9) Primitive Obsession

เป็นลักษณะการใช้ตัวแปรพื้นฐาน(Primitive Data) ให้ถูกวิธี โดยคำนึงถึงว่าใช้เพื่อสิ่งใดหรือควรสร้างเป็นคลาสแทนหรือไม่

10) Switch Statements

การปรากฏของสวิทช์สเตทเมนส์ (Switch Statements) เหมือนกันในโค้ดหลายๆ ที่ ทำให้เกิดโค้ดที่ซ้ำกันได้

11) Parallel Inheritance Hierarchical

เมื่อมีการสร้างคลาสลูกของคลาสหนึ่ง จะต้องตามเพิ่มคลาสลูกนั้นให้อีกคลาสหนึ่งด้วย

12) Lazy Class

คลาสที่ไม่ได้ใช้งาน ซึ่งควรลบทิ้ง

13) Speculative Generality

เมทธอดที่ถูกออกแบบมาให้ใช้งานอย่างใดอย่างหนึ่งแต่ในความเป็นจริงไม่มีการใช้งานเลย เช่น มีการสร้างคลาสหรือเมทธอดขึ้นมาเป็นกรณีทดสอบ

14) Temporary field

การใช้ตัวแปรชั่วคราวมากเกินไป อาจทำให้เกิดความสับสนได้

15) Message Chains

การที่คลาสหนึ่งเรียกอ็อบเจ็คแล้วอ็อบเจ็คนั้นก็ไปเรียกอ็อบเจ็คถัดไปเรื่อยๆ เป็นลูกโซ่

16) Middle man

มีการเรียกใช้คลาสตัวแทน (Delegation) บ่อยครั้ง ซึ่งไม่มีความจำเป็น

17) Inappropriate Intimacy

การที่คลาสฯ หนึ่งสามารถเข้าถึงในส่วนที่ไม่สามารถเข้าถึงได้ของอีกคลาส

18) Alternative classes that with different interface

คลาสที่มีการทำงานเหมือนกันหลายๆ คลาสแต่มีชื่อต่างกันก็ควรจะนำมารวมกันเพื่อใช้ลักษณะช่องทางเดียวกัน

19) Incomplete library class

คลาสไลบารี่ไม่สมบูรณ์ จนต้องมีการแก้ไขข้อมูลเพื่อให้การทำงานสมบูรณ์

20) Data Class

คลาสที่มีฟิลด์ข้อมูลและเมทธอด สำหรับทำงานที่สามารถเชื่อมต่อได้ แต่ยังไม่มีพฤติกรรมที่แท้จริง จะต้องเปลี่ยนเมทธอดเหล่านั้นให้ระดับการเข้าถึงเป็นแบบส่วนตัว(Private)

21) Refused Bequest

คุณสมบัติที่คลาสลูกไม่ต้องการใช้ แต่ได้รับจากการสืบทอดคุณสมบัติจากคลาสแม่มาทั้งหมด

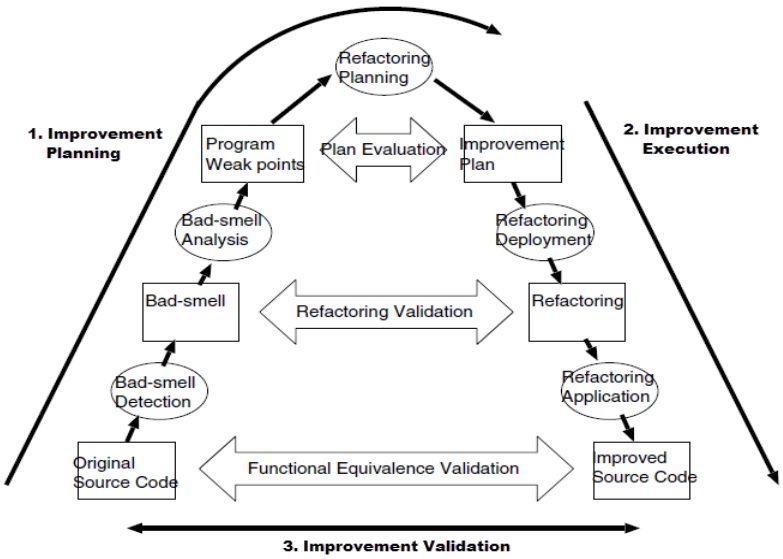
22) Comments

การที่ต้องเขียนคำอธิบาย (Comments) มากๆ นั้นแสดงว่าโปรแกรมเข้าใจยาก ถ้าโปรแกรมเข้าใจง่ายอยู่แล้ว ก็ไม่จำเป็นต้องเขียนคำอธิบาย

2.1.2 รีแฟคทอริง (Refactoring) [2]

รีแฟคทอริง หมายถึง กระบวนการเปลี่ยนแปลงโครงสร้างของซอฟต์แวร์ โดยการเปลี่ยนแปลงนั้นจะไม่กระทบกับพฤติกรรมภายนอก (External Behavior) หรือฟังก์ชันการทำงาน เพื่อทำให้ซอฟต์แวร์นั้นง่ายต่อการบำรุงรักษา การทำความเข้าใจการทำงานของซอฟต์แวร์ และมีความง่ายต่อการปรับปรุงหรือนำซอฟต์แวร์นั้นมาพัฒนาเพื่อเพิ่มฟังก์ชันการทำงานใหม่ๆ เข้าไปในซอฟต์แวร์นั้น การทำรีแฟคทอริงมีกระบวนการทำงานตามลำดับ[3] ดังด้านล่างนี้ และกระบวนการดังกล่าว สามารถแสดงรายละเอียดดังรูปที่ 2.1

1. การวางแผนปรับปรุงซอฟต์แวร์ (Improvement Planning) แบ่งเป็น 3 ขั้นตอน คือ
   - การตรวจหาร่องรอยที่ผิดพลาด (Bad-smells Detection)
   - การวิเคราะห์ร่องรอยที่ผิดพลาด (Bad-smells Analysis)
   - การวางแผนการทำรีแฟคทอริง (Refactoring Planning)
2. การตรวจสอบความเหมาะสมของการปรับปรุงซอฟต์แวร์ (Improvement Validation) แบ่งเป็น 3 ขั้นตอน คือ
   - การประเมินการทำรีแฟคทอริง (Plan Evaluation)
   - การตรวจสอบความเหมาะสมของการทำรีแฟคทอริง (Refactoring Validation)
   - การตรวจสอบความเท่าเทียมกันของฟังก์ชัน (Function Evaluation Validation)
3. การปรับปรุงซอฟต์แวร์ (Improvement Execution) แบ่งเป็น 2 ขั้นตอน คือ
   - การเตรียมการรีแฟคทอริง (Refactoring Deployment)
   - การประยุกต์ใช้รีแฟคทอริง (Refactoring Application)



รูปที่ 2.1 กระบวนการรีแฟคทอริง [1]

รายละเอียดวิธีการรีแฟคทอริงของโปรแกรมเชิงวัตถุนั้น สรุปและรวบรวมโดย Martin Fowler มีทั้งหมด 72 วิธี และร่องรอยที่ผิดพลาดในหัวข้อ 2.1 ต้องรีแฟคทอริงด้วยวิธีการอย่างน้อยหนึ่งวิธีใน 72 วิธีของ Martin Fowler [2]

วิธีการรีแฟคทอริงสำหรับร่องรอยที่ผิดพลาดที่เขียนด้วยภาษา พีแอล/เอสคิวแอล ที่วิเคราะห์และรวบรวมโดย PL/SQL Knowledge Xpert มีทั้งหมด 20 ประเภท มีรายละเอียดดังนี้ [7]

1) Business Logic in Exception Sections

   คุณลักษณะ คือ การวางซอร์ซโค้ดส่วนที่มีความสำคัญ เช่น ตรรกะทางธุรกิจ ไว้ในส่วนของข้อยกเว้น (Exception) เป็นลักษณะที่ไม่สมควรกระทำ

   วิธีการรีแฟคอทิง คือ โดยการย้ายซอร์ซโค้ดในส่วนของข้อยกเว้น นำไปสร้างเป็นโพรซีเยอร์ใหม่ ไว้ในส่วนของ Declaration Section

2) Calling DBMS.OUTPUT.PUT_LINE Via Wrapper

   คุณลักษณะ คือ การเรียกใช้ DBMS.OUTPUT.PUT_LINE โดยตรง

   วิธีการรีแฟคทอริง คือ เรียกใช้คำสั่ง DBMS.OUTPUT.PUT_LINE ผ่านตัวแปร หรือ ฟังก์ชัน

3) Hide Business Rules Inside Functions

   คุณลักษณะ คือ การแสดงซอร์ซโค้ดส่วนของกฎหรือข้อบังคับทางธุรกิจไว้ในส่วนการทำงานหลักโดยไม่ซ่อนไว้ในฟังก์ชัน

   วิธีการรีแฟคทอริง คือ ซ่อนซอร์ซโค้ดส่วนของกฎหรือข้อบังคับทางธุรกิจไว้ในฟังก์ชัน

4) Hide References to Package Variables

คุณลักษณะ คือ ไม่ทำการซ่อนการอ้างอิงถึงตัวแปรในระดับโกลบอล (Global) เนื่องจากตัวแปรที่ประกาศไว้ในระดับโกลบอล แล้วมีการเรียกใช้ในฟังก์ชัน หรือ โพรซีเยอร์ และมีการถูกเปลี่ยนแปลงค่าของตัวแปรแล้ว ซึ่งทำให้ตัวแปรนั้น เมื่อนำไปใช้ต่อในฟังก์ชัน หรือ โพรซีเยอร์ถัดไป อาจจะมีค่าตัวแปรที่เปลี่ยนแปลงไปแล้ว ซึ่งส่งผลให้ได้ค่าตัวแปรที่ไม่ถูกต้องเมื่อต้องการเรียกใช้ค่าตัวแปรในระดับโกลบอล

วิธีการรีแฟคทอริง คือ ซ่อนการอ้างอิงถึงตัวแปรในระดับโกลบอล (Global)

5) Complex Logic Into Separate Modules

คุณลักษณะ คือ ไม่มีการแยกส่วนของซอร์ซโค้ดที่มีความซับซ้อนสูงเป็นอีกโมดุล

วิธีการรีแฟคทอริง คือ นำส่วนของซอร์ซโค้ดที่มีความซับซ้อนสูงแยกเป็นอีกโมดุล

6) Initialize and Cleanup Logic in Execution Section

คุณลักษณะ คือ การวางซอร์ซโค้ด ส่วนกำหนดค่า กับ ส่วนการลบล้างค่า ไว้ในส่วนการทำงานหลักโดยไม่แยกไว้ในโพรซีเยอร์ต่างหากให้ชัดเจน

วิธีการรีแฟคทอริง คือ แยก ส่วนกำหนดค่า กับ ส่วนการลบล้างค่า ออกไปไว้ในโพรซีเยอร์แยกต่างหากให้ชัดเจน แล้วเรียกใช้งานผ่านโพรซีเยอร์แทนการนำซอร์ซโค้ดเหล่านั้นมาไว้ปนกับส่วนของการทำงานหลัก

7) Move Repeated Logic

คุณลักษณะ คือ การวางส่วนของโค้ดที่ซ้ำกันไว้ในโปรแกรมหลายๆ ที่ โดยไม่มีการย้ายส่วนของซอร์ซโค้ดที่ซ้ำกันนั้นไปสร้างเป็นฟังก์ชัน หรือ โพรซีเยอร์ แล้วเรียกใช้งานแทน

วิธีการรีแฟคทอริง คือ ย้ายส่วนของซอร์ซโค้ดที่ซ้ำกัน โดยนำไปสร้างเป็นฟังก์ชัน หรือ โพรซีเยอร์ แล้วเรียกใช้งานแทนการเขียนโค้ดซ้ำๆ กันไว้หลายตำแหน่ง

8) Reduce Declaration Sections

คุณลักษณะ คือ ขนาดในส่วนของการประการศตัวแปรมีขนาดใหญ่ แต่ไม่มีการปรับลดขนาดให้เล็กลง

วิธีการรีแฟคทอริง คือ ปรับลดขนาดส่วนการประการศตัวแปรที่มีขนาดใหญ่ให้เล็กลง

9) Reorganize Large Packages

คุณลักษณะ คือ แพ็คเก็จมีขนาดใหญ่

วิธีการรีแฟคทอริง คือ แยกแพ็คเก็จที่มีขนาดใหญ่ออกมาสร้างเป็นแพ็คเก็จย่อย

10) Use Literals

คุณลักษณะ คือ การระบุค่าตัวอักษรให้โดยตรงในการใช้คำสั่งต่างๆ โดยไม่เรียกใช้งานผ่านตัวแปรประเภทค่าคงที่ หรือ ฟังก์ชัน

วิธีการรีแฟคทอริง คือ เรียกใช้งานผ่านตัวแปรประเภทค่าคงที่ หรือ ฟังก์ชันแทนเมื่อต้องใช้งานค่าตัวอักษร

11) Replace Single Row Queries

คุณลักษณะ คือ ไม่มีการสร้างฟังก์ชันสำหรับซอร์ซโค้ดส่วนที่ให้ผลลัพธ์จากการสืบค้นข้อมูลเพียง 1 เรคคอร์ด ซึ่งควรแทนที่ซอร์ซโค้ดส่วนที่ให้ผลลัพธ์จากการสืบค้นข้อมูลเพียง 1 เรคคอร์ด ด้วยฟังก์ชัน

วิธีการรีแฟคทอริง คือ นำซอร์ซโค้ดส่วนที่ให้ผลลัพธ์จากการสืบค้นเพียง 1 เรคคอร์ดไปสร้างเป็นฟังก์ชัน แล้วเรียกใช้งานฟังก์ชันแทน

12) Replace Status Codes

คุณลักษณะ คือ ส่วนของซอร์ซโค้ดที่เขียนขึ้นเพื่อแสดงสถานะบางอย่าง เช่น ตรวจสอบสถานะการทำงานที่ไม่สมบูรณ์ แต่ไม่นำไปไว้ในส่วนของข้อยกเว้น

วิธีการรีแฟคทอริง คือ นำส่วนของซอร์ซโค้ดที่เขียนขึ้นเพื่อแสดงสถานะบางอย่าง เช่น ตรวจสอบสถานะการทำงานที่ไม่สมบูรณ์ นำไปไว้ในส่วนของข้อยกเว้น

13) Using FORMAT_ERROR_STACK

คุณลักษณะ คือ ใช้งาน SQLERRM ซึ่ง SQLERRM มีขนาดเพียงแค่ 255 Bytes เท่านั้น

วิธีการรีแฟคทอริง คือ ใช้คำสั่ง FORMAT_ERROR_STACK แทน SQLERRM

14) Not Using SUBTYPES and Anchored Types

คุณลักษณะ คือ การประกาศตัวแปรประเภท VARCHAR2 โดยไม่ใช้การประกาศแบบอ้างอิงชื่อคอลัมน์ในตารางข้อมูล

วิธีการรีแฟคทอริง คือ ใช้การประกาศแบบอ้างอิงชื่อคอลัมน์ในตารางข้อมูล คือการใช้ <field_name>.%TYPE สำหรับตัวแปรประเภท VARCHAR2 แต่ถ้าไม่มีตารางข้อมูล ก็ใช้คำสั่ง SUBTYPE เข้ามาช่วยในการประกาศตัวแปร โดยวิธีการนี้ เมื่อมีการเปลี่ยนแปลงความยาวของตัวแปรประเภท VARCHAR2 ในตารางข้อมูล จะไม่กระทบกับซอร์ซโค้ดส่วนนี้

15) Using Collections for Multiple Accesses

คุณลักษณะ คือ การเข้าถึงข้อมูล Static Data หลายๆ ครั้ง แต่ไม่มีการใช้โครงสร้างข้อมูลแบบชั่วคราว (Collections)

วิธีการรีแฟคทอริง คือ การเข้าถึงข้อมูลแบบคงที่ (Static Data) หลายๆ ครั้ง ควรใช้โครงสร้างข้อมูลแบบชั่วคราว (Collections) เข้ามาช่วยเพื่อเป็นการลดการประมวลผลโดยตรงจากฐานข้อมูล

16) Using Explaining Variables

คุณลักษณะ คือ เมื่อมีส่วนของซอร์ซโค้ดที่ซับซ้อนและเกี่ยวข้องกับสูตรต่างๆ แต่ไม่มีการกำหนดตัวแปรขึ้นมารองรับค่าการคำนวณนั้นๆ ก่อน แล้วจึงเรียกใช้งานผ่านตัวแปร

วิธีการรีแฟคทอริง คือ เมื่อมีส่วนของซอร์ซโค้ดที่ซับซ้อนและเกี่ยวข้องกับสูตรต่างๆ ควรมีการกำหนดตัวแปรขึ้นมารองรับค่าการคำนวณนั้นๆ ก่อน แล้วจึงเรียกใช้งานผ่านตัวแปรนั้น จะลดความซับซ้อน เพราะการนำสูตรไว้ในโค้ดนั้นเลยจะทำให้เข้าใจยาก โดยเฉพาะเมื่อสูตรมีความยาวมาก

17) Writing Handlers

คุณลักษณะ คือ ไม่มีการเขียนส่วนจัดการ (handlers) ให้กับความผิดพลาด (errors) ที่สามารถคาดเดาได้ว่าจะเกิดขึ้น

วิธีการรีแฟคทอริง คือ เขียนส่วนจัดการ (handlers) ให้กับความผิดพลาด (errors) ที่สามารถคาดเดาได้ว่าจะเกิดขึ้น เช่น ในการค้นหาข้อมูล มีความเป็นไปได้ที่จะไม่พบข้อมูลเลย ก็ควรจะมี EXCEPTION WHEN NO_DATA_FOUND

18) No Range Values in Numeric FOR Loop

คุณลักษณะ คือ ไม่มีการป้องกันโดยการตรวจสอบเงื่อนไขก่อนการใช้ข้อมูลแบบชั่วคราว กับ FOR Loop ซึ่งถ้าค่าช่วงตัวเลขสำหรับการทำงานแบบวนซ้ำ เป็นค่าว่าง (Null) จะทำให้เกิดความผิดพลาด

วิธีการรีแฟคทอริง คือ โดยการตรวจสอบเงื่อนไขก่อนการเรียกใช้คำสั่งที่มีการทำงานแบบวนซ้ำ หรือ ควรใช้ WHILE Loop แทน เนื่องจากมีการตรวจสอบก่อนการทำงานแบบวนซ้ำ

19) Using cursor loops with DML

คุณลักษณะ คือ การใช้การทำงานแบบวนซ้ำ ด้วยภาษาสำหรับจัดการข้อมูลที่จัดเก็บอยู่ในตารางข้อมูล คือ DML (Data Manipulation Language) ประกอบด้วยคำสั่ง ดังนี้ SELECT, INSERT, UPDATE, DELETE ซึ่งมีผลกระทบกับความเร็วในการสืบค้นข้อมูล

วิธีการรีแฟคทอริง คือ การเปลี่ยนการทำงานแบบวนซ้ำ ที่มีคำสั่งที่ใช้จัดการข้อมูล ประกอบด้วยคำสั่ง ดังนี้ SELECT, INSERT, UPDATE, DELETE ให้อยู่ในรูปของ การประมวลผลแบบปริมาณมาก (Bulk Processing) ก็คือ การเปลี่ยนจาก Cursor Loops ไปเป็น Bulk Processing

20) Convert cursor loops with DML and conditional logic

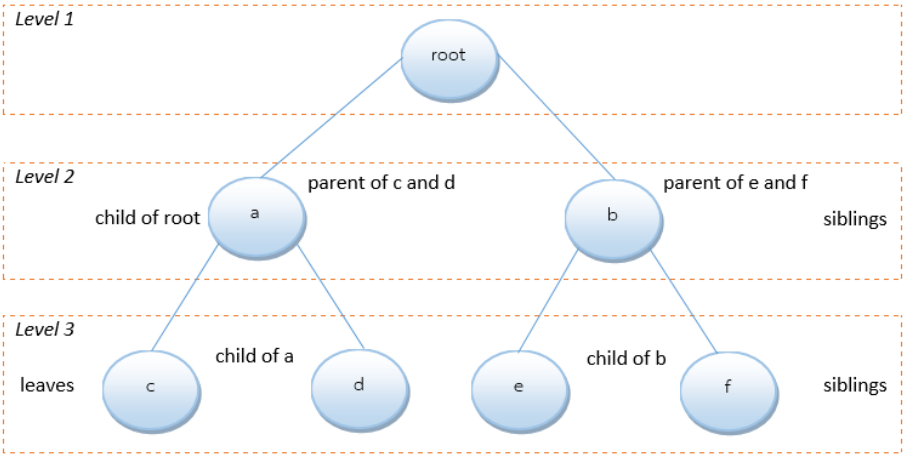
คุณลักษณะ คือ ไม่มีการเปลี่ยนการทำงานแบบวนซ้ำที่ทำงานกับชุดคำสั่งของ DML (Data Manipulation Language) และเงื่อนไขทางตรรกะต่างๆ ให้อยู่ในรูปของ การประมวลผลแบบปริมาณมาก (Bulk Processing)

วิธีการรีแฟคทอริง คือ เปลี่ยนการทำงานแบบวนซ้ำ ด้วยภาษาสำหรับจัดการข้อมูลที่จัดเก็บอยู่ในตารางข้อมูล คือ DML (Data Manipulation Language) ประกอบด้วยคำสั่ง ดังนี้ SELECT, INSERT, UPDATE, DELETE และเงื่อนไขทางตรรกะต่างๆ ให้อยู่ในรูปของ การประมวลผลแบบปริมาณมาก (Bulk Processing) แทน โดยสรุปคือ การเปลี่ยนจาก Cursor Loops และเงื่อนไขทางตรรกะต่างๆ ไปเป็น Bulk Processing

2.1.3 แผนภาพต้นไม้ (Tree Diagram) [8]

แผนภาพต้นไม้เป็นแผนภาพการจัดเก็บข้อมูลที่มีลักษณะเป็นโครงสร้างโดยไม่มีการวนซ้ำ โดยข้อมูลแต่ละหน่วยจะถูกจัดเก็บไว้ที่โหนด (node) ซึ่งจะมีความสัมพันธ์ของแต่ละหน่วยในลักษณะของพ่อแม่กับลูก (parent-child relationship) กับโหนดในระดับที่อยู่ติดกัน แผนภาพต้นไม้จะเริ่มจากส่วนของต้นไม้ซึ่งเรียกว่าราก (Root) และมีส่วนประกอบดังแสดงในรูปที่ 2.2 ดังนี้

1. ราก (Root) : คือข้อมูลที่เป็นรากต้นไม้
2. โหนด (Node) : ข้อมูลที่จัดเก็บอยู่ในแผนภาพต้นไม้มีความสัมพันธ์กันในลักษณะของพ่อกับลูกกับโหนดที่อยู่ในระดับที่ติดกัน
3. โหนดพ่อแม่ (Parent Node) : โหนดที่อยู่ในระดับบนที่ติดกัน
4. โหนดลูก (Child Node) : โหนดที่อยู่ในระดับล่างที่ติดกัน
5. โหนดพี่น้อง (Siblings Node) : โหนดที่อยู่ในระดับเดียวกัน และมี Parent Node เป็นโหนดเดียวกัน
6. โหนดใบ (Leaf Node) : โหนดที่ไม่มีโหนดลูก
7. ระดับ (Level) : ระยะทางตามแนวดิ่งจากโหนดรากไปจนถึงโหนดนั้นๆ



รูปที่ 2.2 ภาพแสดงแผนภาพต้นไม้

2.1.3.1 การท่องแผนภาพต้นไม้ (Tree Traversal)

การท่องผ่านไปตามโหนดหรือการค้นหาข้อมูลในแผนภาพต้นไม้ สามารถแบ่งลำดับการท่องผ่านหรือการค้นหา ได้ดังนี้

1. Depth-first Traversal เป็นการท่องผ่านโหนดหรือการค้นหาข้อมูล มีรูปแบบดังนี้
   - การท่องแบบก่อนลำดับ (Preorder Traversal) : การท่องโดยมีลำดับจากราก แล้วท่องไปยังโหนดลูกที่อยู่ทางซ้ายและขวา ตามลำดับ
     ราก -> โหนดทางซ้าย -> โหนดทางขวา
   - การท่องแบบตามลำดับ (Inorder Traversal) : การท่องโดยให้ความสำคัญกับโหนดลูกที่อยู่ทางซ้ายก่อน แล้วท่องไปที่รากและลูกที่อยู่ทางขวา ตามลำดับ
     โหนดทางซ้าย -> ราก -> โหนดทางขวา
   - การท่องแบบหลังลำดับ (Postorder Traversal) : การท่องโดยให้ความสำคัญกับโหนดลูกที่อยู่ทางซ้ายก่อน แล้วท่องไปยังโหนดลูกที่อยู่ทางขวาและราก ตามลำดับ
     โหนดทางซ้าย -> โหนดทางขวา -> ราก
2. Breadth-first Traversal
   เป็นการท่องผ่านโหนดหรือการค้นหาแนวกว้างลงไปทีละระดับของแผนภาพต้นไม้ตามลำดับ
   ราก -> ระดับ

2.1.4 การวัดซอฟต์แวร์ (Software Measurement)

การวัดเป็นพื้นฐานสำคัญสำหรับทุกศาสตร์ทางวิศวกรรม ซึ่งเป็นกระบวนการกำหนดค่าตัวเลข หรือสัญลักษณ์ ให้กับคุณลักษณะ (Attributes) ของเอนทิตี (Entities) หรือสิ่งที่อยู่ในโลกของความเป็นจริงที่เราสนใจ เพื่อบรรยายคุณลักษณะนั้น ให้อยู่ในรูปของกฎที่นิยามไว้ชัดเจน สำหรับการวัดในทางวิศวกรรมซอฟต์แวร์ (Software Engineering) นั้น การวัดสามารถแบ่งออกได้เป็น 2 วิธี คือ

1. การวัดทางตรง (Direct Measurement) เป็นการวัดเฉพาะคุณลักษณะภายใน (Internal Attribute) ของสิ่งที่เราสนใจโดยไม่นำคุณลักษณะหรือเอนทิตี อื่นมาเกี่ยวข้อง ผลจากการวัดทางตรงทำให้ทราบลักษณะในด้านโครงสร้างของซอฟต์แวร์ ลักษณะของมาตรวัด (Metric) ซึ่งเป็นค่าพื้นฐาน เช่น การวัดความยาวของซอร์ซโค้ดสามารถวัดจากการนับจำนวนบรรทัดทั้งหมดของโปรแกรมเป็นต้น

2. การวัดทางอ้อม (Indirect Measurement) เป็นการวัดโดยการนำการวัดทางตรงมาประกอบการวัด เนื่องจากคุณลักษณะบางอย่าง ผลของการวัดทางอ้อม คือคุณภาพของซอฟต์แวร์ (Software Quality) ในด้านต่างๆ เช่น การวัดความสามารถในการใช้งาน (Usability) ความสามารถในการทดสอบ (Testability) ความสามารถในการบำรุงรักษา (Maintainability) ความสามารถใน

การทำความเข้าใจระบบ (Understandability) เป็นต้น การที่จะวัดเพื่อทราบถึงคุณภาพของซอฟต์แวร์ ในด้านใดด้านหนึ่งนั้น จะต้องมีกำหนดเกณฑ์ (Criteria) ในการวัด หรือ ปัจจัย (Factor) และ แต่ละเกณฑ์ จะมีการกำหนดมาตรวัดที่เกี่ยวข้อง ที่วัดได้จากการวัดทางตรงหรือจากมาตรวัดทางอ้อม

การวัดทั้ง 2 วิธี จำเป็นต้องใช้มาตรวัด เพื่อวัดคุณลักษณะของสิ่งที่เราสนใจ ในปัจจุบันได้มีนักวิจัยหลายท่านได้นำเสนอมาตรวัดใหม่ๆ ที่นิยมใช้กันอย่างแพร่หลาย โดยมาตรวัดเหล่านี้สามารถแบ่งได้เป็น 2 ประเภท ได้แก่มาตรวัดแบบดังเดิม (Traditional Metrics) เช่น จำนวนบรรทัดของโปรแกรม (Lines Of Code - LOC) ค่าวัดไซโครเมติกคอมเพลกซิตีของแมคเคบ (Cyclomatic Complexity - CC) ใช้สำหรับวัดค่าความซับซ้อนของโปรแกรม และมาตรวัดเชิงวัตถุ ได้แก่ ระดับความลึกของการสืบทอดคุณสมบัติของคลาส (Depth of Inheritance Tree - DIT) จำนวนคลาสลูก (Number Of Children - NOC) และขาดความสัมพันธ์ระหว่างวัตถุ (Coupling Between Object - CBO) หลังจากหาค่ามาตรวัดต่างๆ ได้แล้วจะต้องนำมาตรวัดเหล่านี้มาทำการตรวจสอบ (Validation) ว่าค่ามาตรวัดที่ได้มีความน่าเชื่อถือ (Reliability) สามารถนำไปใช้วัดคุณภาพซอฟต์แวร์ต่อไปได้

2.2 งานวิจัยที่เกี่ยวข้อง

สำหรับงานวิจัยที่เกี่ยวข้องสามารถแบ่งออกเป็น 3 กลุ่มหลักๆ คือ งานวิจัยที่เกี่ยวข้องกับโปรแกรมที่เก็บอยู่ในฐานข้อมูล[5] [9] งานวิจัยที่เกี่ยวข้องกับแผนภาพต้นไม้[8] [10] [3] [11] [12] และงานวิจัยที่เกี่ยวข้องกับการวิเคราะห์บริบท[13] [14]

2.2.1 งานวิจัย “PL/SQL Advisor: a Static Analysis-based Tool to Suggest Improvements for Stored Procedures” [5]

งานวิจัยนี้นำเสนอเครื่องมือที่ชื่อว่า "PL/SQL Advisor" ซึ่งเป็นเครื่องมือที่ให้คำแนะนำเกี่ยวกับสโตร์โพรซีเยอร์ ที่พัฒนาด้วยภาษา พีแอล/เอสคิวแอล เนื่องจากพบว่าบางครั้งสโตร์โพรซีเยอร์นี้ เมื่อมีการทำงานก็จะใช้ทรัพยากรของเครื่องมาก และมีปัญหาอื่นๆ อีก เช่น การส่งข้อมูลล่าช้าให้กับเครื่องลูกข่าย (Clients) โดยเครื่องมือที่นำเสนอในงานวิจัยนี้จะทำการตรวจหาประสิทธิภาพและคุณภาพแบบอัตโนมัติสำหรับสโตร์โพรซีเยอร์ เพื่อให้คำแนะนำ และสามารถนำไปปรับปรุงสโตร์โพรซีเยอร์ให้มีคุณภาพดีขึ้น งานวิจัยนี้จะแบ่งกลุ่มของการให้คำแนะนำออกเป็น 2 กลุ่มหลักๆ คือ กลุ่มของการปรับปรุงด้านประสิทธิภาพ (Efficiency Improvements) และกลุ่มการปรับปรุงด้านคุณภาพของโค้ด (Code Quality Improvements) โดยการปรับปรุงทั้ง 2 แบบ มีตัวบ่งชี้ (Indicate) ดังตารางที่ 2.1

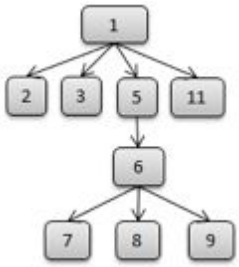
ตารางที่ 2.1 แสดงตัวบ่งชี้การปรับปรุงความมีประสิทธิภาพและด้านคุณภาพของโค้ด

Efficiency Improvements
1. Data type changes (DTC)
2. Changes in the expression evaluation order (EO)
3. Reduction of parameter copying (PC)
4. Optimization of database-related operations (DB-OP)
5. Utilization of native functions (NF)
6. Removal of useless declarations (UD)
Code Quality Improvements
1. Simplify code control flow (CCF)
2. Avoid dodgy code (DC)
3. Avoid collateral effects (CE)
4. Improve the meaning of identifiers (MI)
5. Usage of programming styles (PS)

ในงานวิจัยนี้ใช้ต้นไม้พึ่งพาการควบคุม (Control Dependence Tree (CDT)) ซึ่งเกิดจากกราฟการไหลของการควบคุม (Control Flow Graph(CFG)) โดยรูปที่ 2.3 เป็นตัวอย่าง CDT ของ Code 1

Code 1. Inefficient procedure which persists the employees of an enterprise that are eligible to retire.

```
1. CREATE PROCEDURE ELIGIBLE_EMPLOYEES (emp_limit IN INTEGER) AS
2.          count_employees INTEGER := 0;
3.          emp employee_table%ROWTYPE;
4. BEGIN
5.          FOR emp IN (SELECT ID, NAME, YEARS_WORKED FROM EMPLOYEE_TABLE) LOOP
6.                    IF (employee_score(emp.id) > 5 AND emp.years_worked > 30) THEN
7.                      INSERT INTO ELIGIBLE_EMPLOYEES VALUES (emp.id, emp.name, emp.years_worked);
8.                      count := count + 1;
9.                      COMMIT;
10                                . . . .
11.         DBMS_OUTPUT.PUT_LINE('# Eligible Employees:' || count_employees);
12. END ELIGIBLE_EMPLOYEES;
```



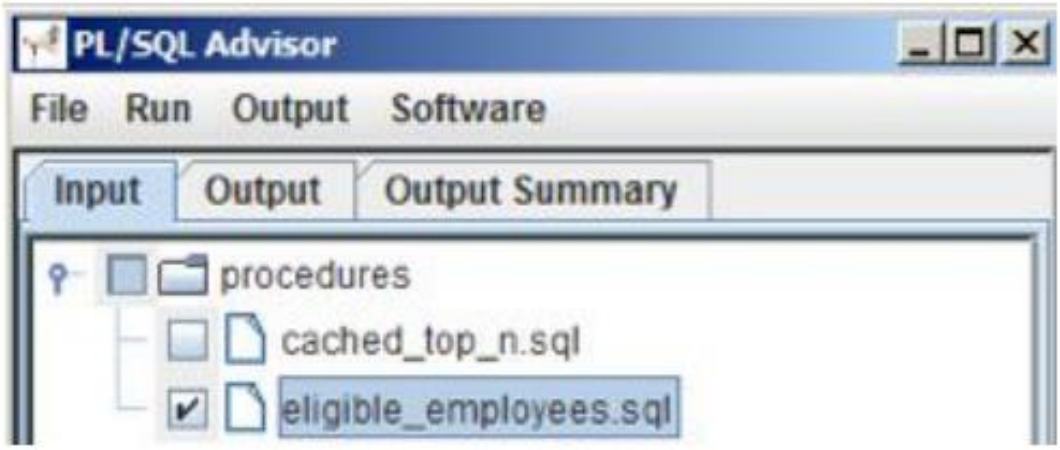
รูปที่ 2.3 ภาพแสดง CDT ของ Code 1

การปรับปรุงด้านความมีประสิทธิภาพ งานวิจัยนี้จะสร้างตัววิเคราะห์ความมีประสิทธิภาพ (Efficiency Analyzer) สำหรับวิเคราะห์โครงสร้างของ CDT เพื่อที่จะตรวจจับ (Detect) แพทเทิร์น (Patterns) ของโค้ดเพื่อระบุว่ามีส่วนใดที่ต้องมีการปรับปรุงด้านความมีประสิทธิภาพตามหัวข้อ ของตัวบ่งชี้ การปรับปรุงด้านคุณภาพของโค้ด งานวิจัยนี้จะสร้างตัววิเคราะห์ความมีคุณภาพ (Quality Analyzer) สำหรับตรวจจับร่องรอยที่ผิดพลาด (Code Smells) ผ่านโครงสร้างทั้งหมด ของ CDT

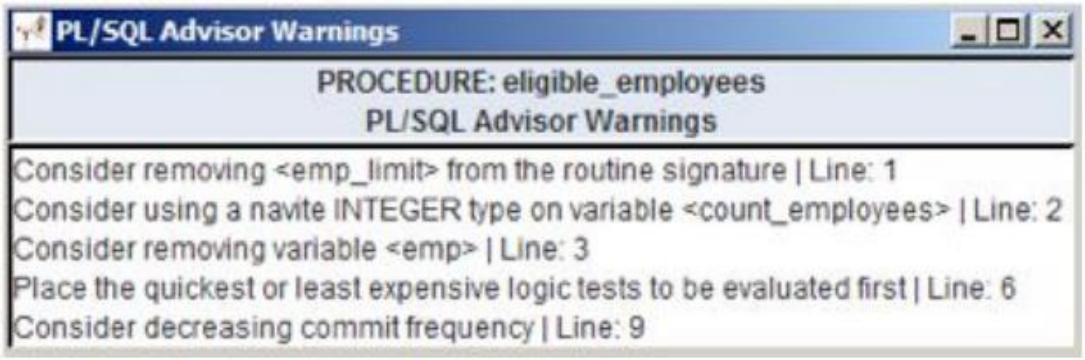
ตัวอย่างรายงานมีรูปแบบ ดังรูปที่ 2.4 โดยจะแบ่ง เป็น 2 ส่วนหลักๆ คือ การปรับปรุงด้านความมีประสิทธิภาพ และการปรับปรุงด้านคุณภาพของโค้ด โดยรูปที่ 2.5 และ 2.6 คือ ภาพแสดงหน้าจอการใช้งาน และ ภาพแสดงหน้าจอผลลัพธ์ของ PL/SQL Advisor ตามลำดับ

Efficiency Improvements	
Short Description	*Subclass*
E1: Use native INTEGER types	DTC
E2: Use native NUMBER types	DTC
E3: Avoid using constrained datatypes (e.g. POSITIVE, POSITIVEN, NATURAL)	DTC
E4: Optimize the order of logic expressions	EO
E5: Optimize the order of conditional commands	EO
E6: Pass parameters by reference	PC
E7: Optimize the length of variables	DTC
E8: Reduce context switches [Berkovic et al. 2010; Feuerstein 2007]	DB-OP
E9: Decrease commit frequency	DB-OP
E10: Remove declarations inside loops	UD
E11: Use built-in string functions	NF
E12: Iterate over rows implicitly	DB-OP
E13: Remove unused variables	UD
E14: Remove unused parameters	UD
Code Quality Improvements	
Short Description	*Subclass*
Q1: Avoid using GOTO commands	CCF
Q2: Avoid declaring a variable using the same identifier of a loop for counter	DC
Q3: Avoid using escape commands (e.g. EXIT, CONTINUE) inside loops	CCF

รูปที่ 2.4 แสดงรายงานของ PL/SQL Advisor



รูปที่ 2.5 ภาพแสดงหน้าจอการใช้งานของ PL/SQL Advisor



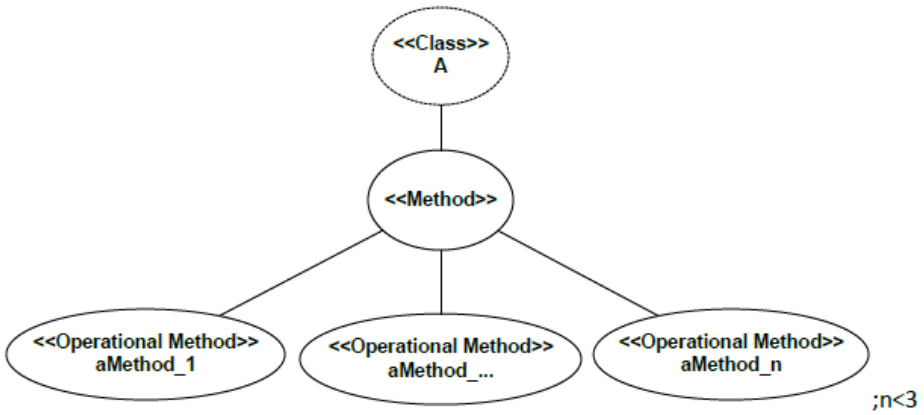
รูปที่ 2.6 ภาพแสดงหน้าจอผลลัพธ์ของ PL/SQL Advisor

งานวิจัยนี้วิเคราะห์โครงสร้าง CDT โดยใช้อัลกอริทึมการค้นหาในแนวลึก (Depth First Search(DFS)) และ อัลกอริทึมการค้นหาในแนวกว้าง (Breadth First Search(BFS)) และได้ทำการประเมินผลงานวิจัยโดยการตั้ง เป็นข้อคำถาม 3 ข้อ และหาคำตอบด้วยการทำกรณีศึกษา (Case Study) จากโปรเจคโอเพ่นซอร์ซ (Open Source Projects) ที่เขียนขึ้นด้วยภาษา พีแอล/เอสคิวแอล และสรุปได้ว่าเครื่องมือนี้มีความสามารถที่จะแสดงรายงานการเตือน (Report Warning) การปรับปรุงภาษาในการเขียนโปรแกรมทั่วไปได้ รวมไปถึงการปรับปรุงที่เฉพาะเจาะจงที่อยู่ในบริบทของฐานข้อมูล ซึ่งก็คือสโตร์โพรซีเยอร์

2.2.2 งานวิจัย "Design and Development of an Approach for detecting Flaws in Software Design Model Using Graph Diagram and Tree Diagram" [8]

งานวิจัยนี้นำเสนอวิธีการและเครื่องมือในการค้นหาข้อบกพร่องในโมเดลการออกแบบซอฟต์แวร์ โดยใช้แผนภาพต้นไม้และแผนภาพกราฟสำหรับการค้นหาข้อบกพร่อง ซึ่งจะทำการวิเคราะห์คุณสมบัติของข้อบกพร่องแต่ละประเภทและนำคุณสมบัติที่ได้มาออกแบบแม่แบบของข้อบกพร่อง และสร้างเป็นแม่แบบของข้อบกพร่อง 10 ประเภท คือ Refused Bequest , Feature Envy , Middle Man , Message Chain , Inappropriate Intimacy , Lazy Class , Data Class , Long Parameter Lists , Large Class , Parallel Inheritance Hierarchy โดยจะค้นหาข้อบกพร่องด้วยการใช้แม่แบบของข้อบกพร่องนำมาเทียบกับแผนภาพแสดงโมเดลการออกแบบ รูปที่ 2.7 คือตัวอย่างของแม่แบบในการค้นหาข้อบกพร่องประเภท Lazy Class

ในการค้นหาข้อบกพร่อง แม่แบบจะถูกนำมาตรวจสอบกับแผนภาพแสดงโมเดลการออกแบบเพื่อตรวจหาส่วนของการออกแบบที่เป็นข้อบกพร่อง โดยการแปลงโมเดลการออกแบบเป็นแผนภาพและนำแม่แบบที่ได้มาเปรียบเทียบกับแผนภาพที่แสดงโมเดลการออกแบบ ในการเปรียบเทียบจะท่องไปตามโหนดในแผนภาพโมเดลการออกแบบ เพื่อนำแม่แบบเปรียบเทียบกับแต่ละโหนด หรือเวอร์เท็กซ์ เพื่อตรวจสอบคุณลักษณะ โดยจะถือว่าตรวจพบข้อบกพร่อง เมื่อคุณลักษณะของโหนด หรือเวอร์เท็กซ์นั้นตรงกับแม่แบบ



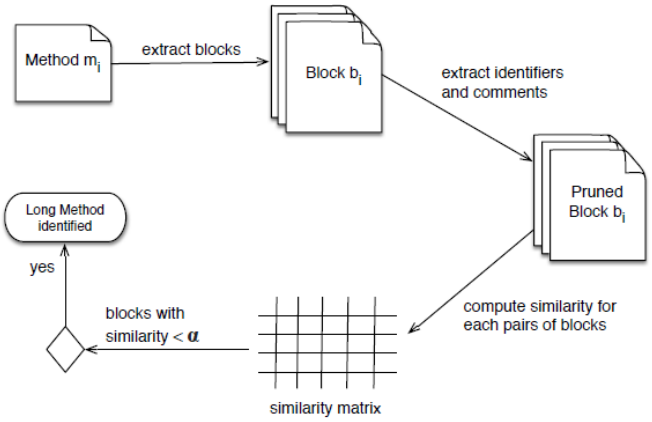
รูปที่ 2.7 ภาพแสดงแม่แบบในการค้นหาข้อบกพร่องประเภท Lazy Class

งานวิจัยนี้ได้นำวิธีการที่นำเสนอไปพัฒนาเป็นเครื่องมือและทำการทดสอบกับโมเดลการออกแบบซอฟต์แวร์ขนาดเล็ก จำนวน 3 ระบบ คือ ระบบฝากและถอนเงินอัตโนมัติ ระบบบันทึกข้อมูลผู้ติดต่อ และ ระบบควบคุมการทำงานของลิฟต์ ซึ่งค่าที่ได้จากการใช้เครื่องมือจะถูกนำมา

เปรียบเทียบกับค่าจริงของข้อบกพร่องที่มีอยู่ในโมเดลทดสอบเพื่อประเมินประสิทธิภาพของวิธีการ ผลการทดสอบพบว่าวิธีการนี้สามารถค้นหาข้อบกพร่องทั้ง 10 ประเภทได้

2.2.3 งานวิจัย “Textual Analysis for Code Smell Detection” [13]

งานวิจัยนี้นำเสนอเทคนิคการวิเคราะห์เกี่ยวกับใจความ เรียกว่า TACO (Textual Analysis for Code smell detectiOn) ซึ่งใช้ในการค้นหาข้อบกพร่องประเภท Long Method โดยผลการวิเคราะห์ที่ได้จากนิยามที่ฟาวเลอร์ได้กล่าวไว้[2] คือ Long Method จะมีการประกอบไปด้วยกลุ่มของโค้ดที่ไม่มีความสัมพันธ์กันในแนวคิด ซึ่งควรจะมีการจัดการแยกออกไป รูปภาพที่ 2.8 คือภาพรวมของขั้นตอนในการค้นหาข้อบกพร่อง



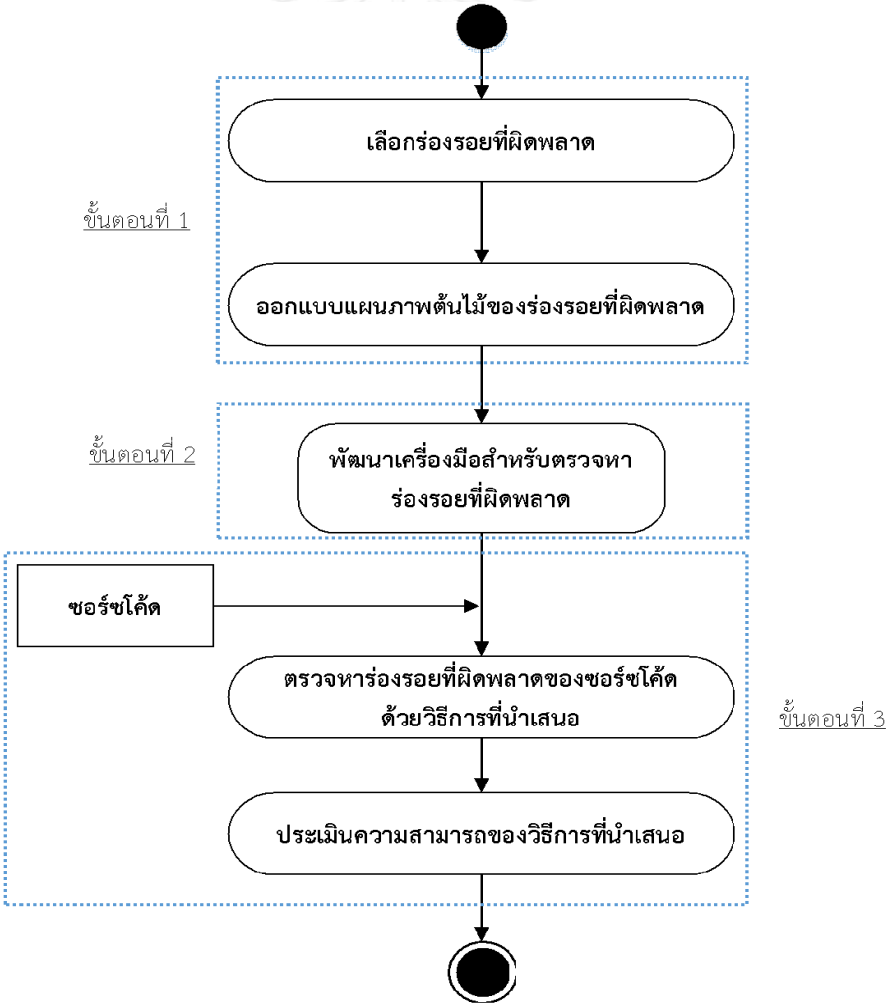
รูปที่ 2.8 ภาพแสดงภาพรวมของการค้นหาร่องรอยที่ผิดพลาดด้วยวิธี TACO

เริ่มจากการสะกัดแยกโค้ดออกมาเป็นบล็อก (Block) แล้วทำการสะกัดคำที่เป็นตัวแปร (Identifiers) และคอมเม้นท์ (Comments) ออกไป รวมถึงพวกที่เป็นคำเฉพาะสำหรับโปรแกรม (Language Keywords) ด้วย หลังจากขั้นตอนนี้ก็จะได้บล็อกของโค้ดแต่ละอันซึ่งปราศจากคำที่สะกัดออกไปเหล่านั้น (Pruned Block) แล้วจึงทำการเปรียบเทียบบล็อกของโค้ดเหล่านั้นเพื่อหาความใกล้เคียงกัน โดยใช้ Latent Semantic Indexing (LSI) ค่าความคล้ายกันที่ได้จากการเปรียบเทียบของบล็อกแต่ละคู่จะถูกเก็บไว้ในเมทริกซ์ความคล้ายกัน (Similarity Matrix) ถ้าค่าที่ได้จากเมทริกซ์ความคล้ายกันน้อยกว่า α (ซึ่งในงานวิจัยนี้ได้ประเมินให้ใช้ค่า α = 0.4) ก็จะถือว่าพบข้อบกพร่องประเภท Long Method

การประเมินผลของวิธีที่นำเสนอในงานวิจัยนี้ ใช้การทดสอบกับ 3 โปรเจค ที่เป็นจาวาโอเพ่นซอร์ซ และได้ผลคือ TACO สามารถค้นหาข้อบกพร่องได้ 50% ถึง 77%

บทที่ 3
วิธีการตรวจหาร่องรอยที่ผิดพลาดของโปรแกรมที่เก็บอยู่ในฐานข้อมูล

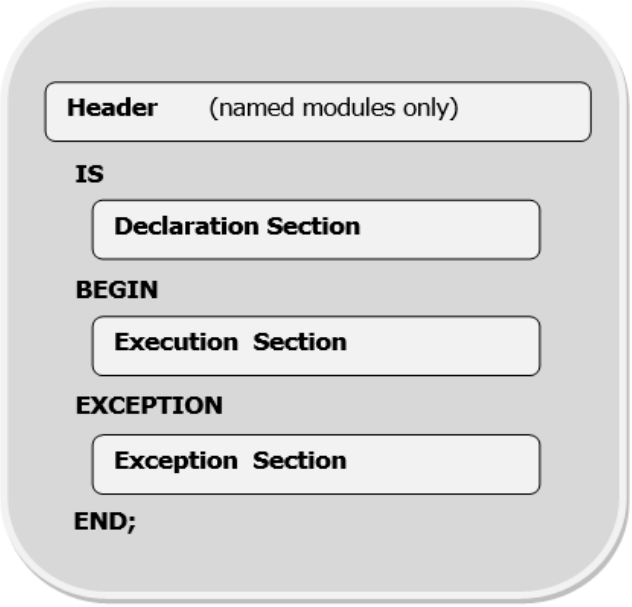
งานวิจัยนี้นำเสนอวิธีการในการตรวจหาร่องรอยที่ผิดพลาดของโปรแกรมที่เก็บอยู่ในฐานข้อมูลด้วยการใช้แผนภาพต้นไม้ (Tree Diagram) และการวิเคราะห์บริบท (Context Analysis) โดยแบ่งเป็น 3 ขั้นตอนหลัก ดังแสดงด้วยแผนภาพแอคทิวิตี (Activity Diagram) ในรูป 3.1 ประกอบด้วย ขั้นตอนที่ 1 คือ เลือกร่องรอยที่ผิดพลาด และ ออกแบบแผนภาพต้นไม้ของร่องรอยที่ผิดพลาด ขั้นตอนที่ 2 คือ พัฒนาเครื่องมือสำหรับตรวจหาร่องรอยที่ผิดพลาด ขั้นตอนที่ 3 คือ ตรวจหาร่องรอยที่ผิดพลาดของซอร์ซโค้ดด้วยวิธีการที่นำเสนอ และ ประเมินความสามารถของวิธีการที่นำเสนอ โดยขั้นตอนที่ 1 จะนำเสนอในบทที่ 3 และขั้นตอนที่ 2 จะนำเสนอในบทที่ 4 และขั้นตอนที่ 3 จะนำเสนอในบทที่ 5



รูปที่ 3.1 ภาพแสดงขั้นตอนงานวิจัย

3.1 เลือกร่องรอยที่ผิดพลาด

จากการศึกษาโครงสร้างของภาษา PL/SQL ซึ่งมีโครงสร้างเป็นลักษณะบล็อก (Block) [15] ดังรูปที่ 3.2 ประกอบด้วย 3 บล็อก คือ Declaration Section , Execution Section และ Exception Section และผู้วิจัยได้ทำการเลือกร่องรอยที่ผิดพลาด 6 ประเภท เพื่อใช้ในงานวิจัยนี้



รูปที่ 3.2 ภาพแสดงโครงสร้างของภาษา PL/SQL

ที่มาของการเลือกร่องรอยที่ผิดพลาดที่ใช้สำหรับการทดลองในงานวิจัยนี้ มีที่มาจากปัญหาที่พบในการทำงานซึ่งประกอบด้วยร่องรอยที่ผิดพลาดประเภท Using cursor loops with DML, Initialize and Cleanup Logic in Execution Section, Use Literals, No Range Values in Numeric FOR Loop, Not Using SUBTYPES and Anchored Types และในบางร่องรอยที่ผิดพลาดมักจะไม่พบในการทำงานแต่เป็นความประสงค์ของผู้วิจัยที่อยากจะทราบถึงผลของค่าดัชนีความสามารถในการบำรุงรักษาที่เกิดจากร่องรอยที่ผิดพลาดประเภท Business Logic in Exception Sections ว่าจะมีผลเป็นอย่างไร

โดยมีรายละเอียดของร่องรอยที่ผิดพลาดและแนวทางสำหรับการแก้ไขที่ได้จาก PL/SQL Knowledge Xpert และตัวอย่างของซอร์ซโค้ดก่อนการรีแฟคทอริง และหลังการรีแฟคทอริง ดังนี้

1. Business Logic in Exception Sections

   <u>แนวทางในการแก้ไข</u> คือ การย้ายซอร์ซโค้ดในส่วนของข้อยกเว้น นำไปสร้างเป็นโพรซีเยอร์ใหม่ ไว้ในส่วนของ Declaration Section

```
PROCEDURE snc (NAME_IN  IN VARCHAR2,) IS
  Part1   VARCHAR2(100);
BEGIN
  IF NAME_IN = 'A' THEN
    Do_Process1;
  END IF;
EXCEPTION
  WHEN OTHERS  THEN
    IF Part1 IS NOT NULL THEN
      Do_Process2;
    END IF;
END snc;
```

รูปที่ 3.3 ภาพแสดงก่อนรีแฟคทอริง BLES

```
PROCEDURE snc (NAME_IN  IN VARCHAR2,) IS
  Part1   VARCHAR2(100);

  PROCEDURE Show_Exception IS
  BEGIN
    IF Part1 IS NOT NULL THEN
      Do_Process2;
    END IF;
  END;

BEGIN
  IF NAME_IN = 'A' THEN
    Do_Process1;
  END IF;
EXCEPTION
  WHEN OTHERS  THEN
    Show_Exception;
END snc;
```

รูปที่ 3.4 ภาพแสดงหลังรีแฟคทอริง BLES

2. Using cursor loops with DML

แนวทางในการแก้ไข คือ เปลี่ยนการทำงานแบบวนซ้ำ (Cursor Loops) สำหรับคำสั่งประเภท DML (Data Manipulation Language) ซึ่งประกอบด้วย INSERT, DELETE, UPDATE, SELECT ให้อยู่ในรูปของการประมวลผลแบบ Bulk Processing

```
CREATE OR REPLACE PROCEDURE upd_for_dept1 (
   dept_in IN employee.department_id%TYPE
  ,newsal IN employee.salary%TYPE) IS
   CURSOR emp_cur IS
     SELECT employee_id, salary, hire_date
      FROM employee
     WHERE department_id = dept_in;
BEGIN
   FOR rec IN emp_cur
   LOOP
     INSERT INTO employee_history (employee_id, salary, hire_date)
        VALUES (rec.employee_id, rec.salary, rec.hiredate);
     UPDATE employee
        SET salary = newsal,
        hire_date = rec.hiredate
        WHERE employee_id = rec.employee_id;
   END LOOP;
END upd_for_dept1;
```

รูปที่ 3.5 ภาพแสดงก่อนรีแฟคทอริง UCL

```
CREATE OR REPLACE PROCEDURE upd_for_dept2 (
   dept_in   IN   employee.department_id%TYPE,
   newsal    IN   employee.salary%TYPE) IS
   TYPE employee_tt IS TABLE OF employee.employee_id%TYPE
     INDEX BY BINARY_INTEGER;
   employees    employee_tt;
   TYPE salary_tt IS TABLE OF employee.salary%TYPE
     INDEX BY BINARY_INTEGER;
   salaries     salary_tt;
   TYPE hire_date_tt IS TABLE OF employee.hire_date%TYPE
     INDEX BY BINARY_INTEGER;
   hire_dates   hire_date_tt;
BEGIN
   SELECT employee_id, salary, hire_date
   BULK COLLECT INTO employees, salaries, hire_dates
   FROM employee
   WHERE department_id = dept_in;
   FORALL indx IN employees.FIRST .. employees.LAST
     INSERT INTO employee_history (employee_id, salary, hire_date)
        VALUES (employees (indx), salaries (indx), hire_dates (indx));
   FORALL indx IN employees.FIRST .. employees.LAST
     UPDATE employee
       SET salary = newsal,
          hire_date = hire_dates (indx)
       WHERE employee_id = employees (indx);
END upd_for_dept2;
```

รูปที่ 3.6 ภาพแสดงหลังรีแฟคทอริง UCL

3. Initialize and Cleanup Logic in Execution Section

<u>แนวทางในการแก้ไข</u> คือ การย้ายซอร์ซโค้ดในส่วนของการกำหนดค่า (Initialize) และส่วนการลบล้างค่า (Clean Up) นำไปสร้างเป็นโพรซีเยอร์ใหม่

```
FUNCTION eqfiles (
   check_this_in          IN   VARCHAR2
  ,check_this_dir_in      IN   VARCHAR2
) RETURN BOOLEAN IS
   checkid        UTL_FILE.file_type;
   checkline      VARCHAR2 (32767);
   retval         BOOLEAN;
   check_eof      BOOLEAN;
   against_eof    BOOLEAN;
BEGIN
   checkid := UTL_FILE.fopen (check_this_dir_in
              ,check_this_in
              ,'R'
              ,max_linesize       => 32767);
   LOOP
      BEGIN
         UTL_FILE.get_line (checkid, checkline);
         IF (check_eof AND against_eof)
         THEN
            retval := TRUE;
            EXIT;
         END IF;
      END;
   END LOOP;
   UTL_FILE.fclose (checkid);
   RETURN retval;
END eqfiles;
```

รูปที่ 3.7 ภาพแสดงก่อนรีแฟคทอริง ICE

```
FUNCTION eqfiles (
   check_this_in          IN   VARCHAR2
  ,check_this_dir_in      IN   VARCHAR2
) RETURN BOOLEAN IS
   checkid        UTL_FILE.file_type;
   checkline      VARCHAR2 (32767);
   retval         BOOLEAN;
   check_eof      BOOLEAN;
   against_eof    BOOLEAN;
   PROCEDURE initialize IS
   BEGIN
      checkid := UTL_FILE.fopen (check_this_dir_in
                 ,check_this_in
                 ,'R'
                 ,max_linesize       => 32767);
   END initialize;
   PROCEDURE cleanup
   IS
   BEGIN
      UTL_FILE.fclose (checkid);
   END cleanup;
BEGIN
   initialize;
   LOOP
      BEGIN
         UTL_FILE.get_line (checkid, checkline);
         IF (check_eof AND against_eof)
         THEN
            retval := TRUE;
            EXIT;
         END IF;
      END;
   END LOOP;
   cleanup;
   RETURN retval;
EXCEPTION WHEN OTHERS
    THEN cleanup;
    RETURN FALSE;
END eqfiles;
```

รูปที่ 3.8 ภาพแสดงหลังรีแฟคทอริง ICE

4. Use Literals

<u>แนวทางในการแก้ไข</u> คือ สร้างคอนสะแตนท์ (Constant) หรือฟังก์ชันให้กับค่าตัวอักษร

```
DECLARE
   l_file_id  UTL_FILE.file_type;
   l_line     VARCHAR2 (32767);
BEGIN
   l_file_id :=
      UTL_FILE.fopen ('c:\temp\install.log'
                      ,'R'
                      ,max_linesize      => 32767);
   LOOP
      BEGIN
         UTL_FILE.get_line (l_file_id, l_line);
      EXCEPTION WHEN NO_DATA_FOUND THEN EXIT;
      END;
      DBMS_OUTPUT.PUT_LINE (l_line);
   END LOOP;
   UTL_FILE.fclose (l_file_id);
END;
```

รูปที่ 3.9 ภาพแสดงก่อนรีแฟคทอริง UL

```
CREATE OR REPLACE PACKAGE utl_file_constants
IS
   c_read_only      CONSTANT VARCHAR2 (1) := 'R';
   c_fopen          CONSTANT VARCHAR2 (100) := 'c:\temp\install.log';
   c_max_linesize   CONSTANT PLS_INTEGER  := 32767;
END utl_file_constants;
```

รูปที่ 3.10 ภาพแสดงการรีแฟคทอริง UL

```
DECLARE
   l_file_id  UTL_FILE.file_type;
   l_line     VARCHAR2 (32767);
BEGIN
   l_file_id := UTL_FILE.fopen (utl_file_constants.c_fopen
                ,utl_file_constants.c_read_only
                ,max_linesize => utl_file_constants.c_max_linesize);
   LOOP
      BEGIN
         UTL_FILE.get_line (l_file_id, l_line);
      EXCEPTION WHEN NO_DATA_FOUND THEN EXIT;
      END;
      DBMS_OUTPUT.PUT_LINE (l_line);
   END LOOP;
   UTL_FILE.fclose (l_file_id);
END;
```

รูปที่ 3.11 ภาพแสดงหลังรีแฟคทอริง UL

5. No Range Values in Numeric FOR Loop

   <u>แนวทางในการแก้ไข</u> คือ ควรต้องป้องกัน โดยการตรวจสอบเงื่อนไขก่อน หรือ ควรใช้ WHILE Loop แทน เนื่องจากมีการตรวจสอบก่อนการทำงานแบบวนซ้ำ

```
CREATE OR REPLACE PROCEDURE process_collection (
   mylist IN my_nested_table_type
   )
IS
BEGIN
   FOR indx IN mylist.FIRST .. mylist.LAST
   LOOP
     process_list_value (mylist (indx));
   END LOOP;
END;
```

รูปที่ 3.12 ภาพแสดงก่อนรีแฟคทอริง NRNFL

```
-- แบบที่ 1
CREATE OR REPLACE PROCEDURE process_collection (
   mylist IN my_nested_table_type
   )
IS
BEGIN
   IF mylist.COUNT > 0
   THEN
      FOR indx IN mylist.FIRST .. mylist.LAST
      LOOP
         process_list_value (mylist (indx));
      END LOOP;
   END IF;
END;

-- แบบที่ 2

CREATE OR REPLACE PROCEDURE process_collection (
   mylist IN my_nested_table_type
   )
IS
   l_row PLS_INTEGER;
BEGIN
   l_row := mylist.FIRST;
   WHILE (l_row IS NOT NULL)
   LOOP
      process_list_value (mylist (l_row));
      l_row := mylist.NEXT (l_row);
   END LOOP;
END;
```

รูปที่ 3.13 ภาพแสดงหลังรีแฟคทอริง RNFL

6. Not Using SUBTYPES and Anchored Types

   <u>แนวทางในการแก้ไข</u> คือ ควรใช้การประกาศแบบอ้างอิงชื่อคอลัมน์ในตารางข้อมูล คือ การใช้<field_name>.%TYPE แต่ถ้าไม่มีตารางข้อมูล ก็ใช้คำสั่ง SUBTYPE เข้ามาช่วยในการประกาศตัวแปร โดยวิธีการนี้ เมื่อมีการเปลี่ยนแปลงความยาวของตัวแปรประเภท VARCHAR2 ในตารางข้อมูล จะไม่กระทบกับซอร์ซโค้ดส่วนนี้

```
DECLARE
   l_company_name    VARCHAR2 (100);
   l_description     VARCHAR2 (10000);
   l_summary         VARCHAR2 (2000);
   l_full_name       VARCHAR2 (200);
BEGIN
    Do_Process1;
END;
```

รูปที่ 3.14 ภาพแสดงก่อนรีแฟคทอริง NUSAT

```
DECLARE
   l_company_name    company.name%TYPE;
   l_description     plsql_types.max_varchar2_t;
   l_summary         plsql_types.max_db_varchar2_t;
   l_full_name       employee_types.full_name_t;
BEGIN
    Do_Process1;
END;
```

รูปที่ 3.15 ภาพแสดงหลังรีแฟคทอริง NUSAT

นิยาม:

BLES *คือ* *Business Logic in Exception Sections*

UCL คือ Using cursor loops with DML

ICE คือ Initialize and Cleanup Logic in Execution Section

UL คือ Use Literals

NRNFL คือ No Range Values in Numeric FOR Loop

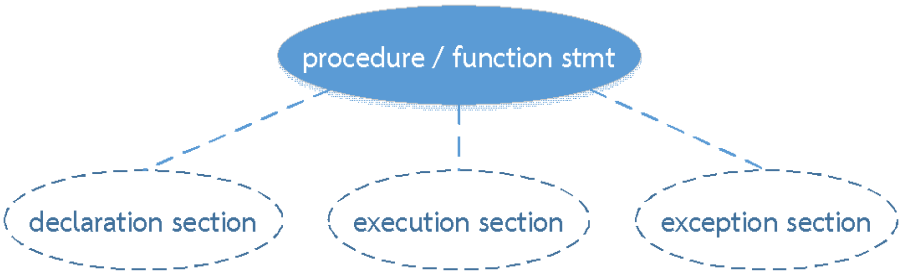
NUSAT คือ Not Using SUBTYPES and Anchored Types

3.2 การออกแบบแผนภาพต้นไม้ของร่องรอยที่ผิดพลาด

งานวิจัยนี้จะใช้แผนภาพต้นไม้เพื่อตรวจสอบหาร่องรอยที่ผิดพลาด โดยจะทำการสร้างแผนภาพต้นไม้ของร่องรอยที่ผิดพลาดทั้ง 6 ประเภท ไว้เป็นต้นแบบ โดยลักษณะโครงสร้างแผนภาพ

ต้นไม้ของแต่ละร่องรอยที่ผิดพลาดจะได้จากการวิเคราะห์โครงสร้างของร่องรอยที่ผิดพลาดแต่ละแบบ เพื่อให้ได้คุณลักษณะสำหรับนำมาสร้างเป็นต้นแบบ โดยมีคุณลักษณะที่ใช้ในการกำหนดแผนภาพต้นไม้ต้นแบบ 3 คุณลักษณะ ดังนี้

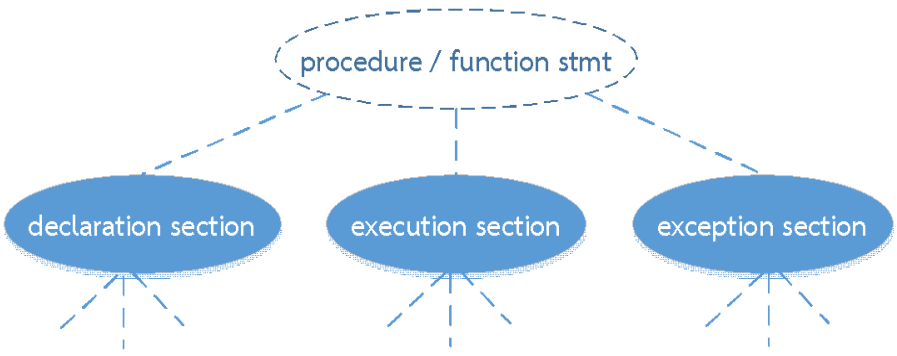
- **โพรซีเยอร์หรือฟังก์ชัน** คือ การกำหนดโหนดเริ่มต้น (Root Node) ให้กับแผนภาพต้นไม้ โดยจะตรวจสอบว่าเป็นโพรซีเยอร์หรือฟังก์ชั่น ดังนั้นโหนดเริ่มต้นของต้นแบบนั้นจะมีลักษณะที่เหมือนกันทั้ง 6 ประเภทร่องรอยที่ผิดพลาด ดังรูปที่ 3.16



รูปที่ 3.16 ภาพแสดงโหนดเริ่มต้นของร่องรอยที่ผิดพลาด

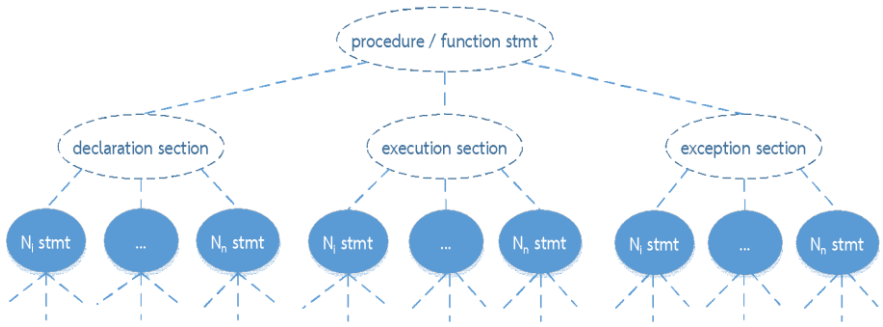
นิยาม: stmt ย่อมาจาก statement

- **บริเวณที่พบร่องรอยที่ผิดพลาด** คือ ส่วนบริเวณพื้นที่เกิดร่องรอยที่ผิดพลาดซึ่งได้จากการวิเคราะห์จากโครงสร้างของภาษาที่แบ่งเป็น 3 บล็อก คือ Declaration Section , Execution Section และ Exception Section ดังนั้น โหนดที่เกิดจากการวิเคราะห์บริเวณที่พบร่องรอยที่ผิดพลาดนี้ จะเป็นโหนดที่เกี่ยวข้องกับบล็อก ดังรูปที่ 3.17



รูปที่ 3.17 ภาพแสดงโหนดจากบริเวณที่พบร่องรอยที่ผิดพลาด

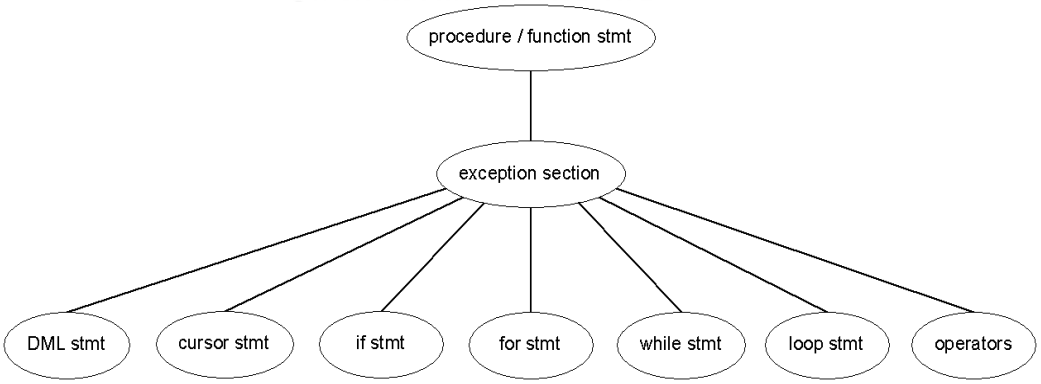
- **บริบทของร่องรอยที่ผิดพลาด** คือ จากการวิเคราะห์จากบริบทของร่องรอยที่ผิดพลาดแต่ละประเภทจะทำให้ได้คุณลักษณะเฉพาะของบริบทในร่องรอยที่ผิดพลาดแต่ละแบบ เพื่อนำมากำหนดเป็นโหนดให้กับต้นแบบ ดังตัวอย่างในรูปที่ 3.17 เป็นตัวอย่างของบริบทที่พบในร่องรอยที่ผิดพลาดต้นแบบ



รูปที่ 3.18 ภาพแสดงตัวอย่างบริบทของร่องรอยที่ผิดพลาด

โดยมีรูปแบบโครงสร้างแผนภาพต้นไม้ที่เป็นต้นแบบและรายละเอียดที่ได้จากการวิเคราะห์ซึ่งแผนภาพต้นไม้ต้นแบบที่ได้จากคุณลักษณะที่ใช้ในการกำหนดแผนภาพทั้ง 3 แบบ มีรายละเอียดดังนี้

1. ร่องรอยที่ผิดพลาดประเภท Business Logic in Exception Sections



รูปที่ 3.19 แผนภาพต้นไม้ต้นแบบของ Business Logic in Exception Sections

<u>คุณลักษณะ</u>

- โพรซีเยอร์หรือฟังก์ชัน : เป็นได้ทั้งโพรซีเยอร์และฟังก์ชัน

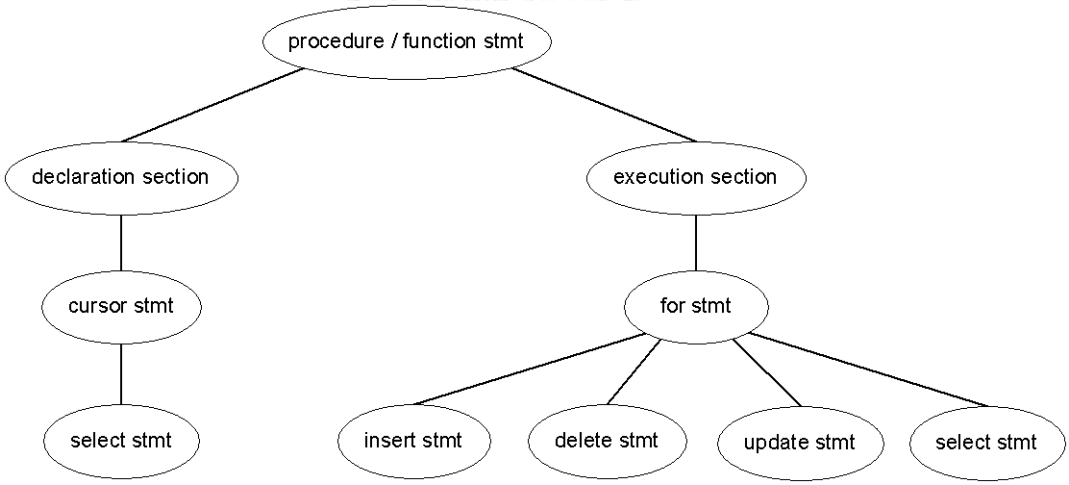
- บริเวณที่พบร่องรอยที่ผิดพลาด : พบบริเวณ exception section
- บริบทของร่องรอยที่ผิดพลาด : ประกอบด้วยบริบทดังนี้ DMS stmt, cursor stmt, if stmt, for stmt, while stmt, loop stmt, operators

นิยาม: DML stmt คือ Data Manipulation Language Statement ประกอบด้วยชุดคำสั่ง คือ insert, delete, update และ select

ข้อกำหนดจากการวิเคราะห์บริบท คือ

- ตรวจสอบในส่วนของ Exception Section
- ตรวจสอบ Business Logic เพิ่มเติมของระดับโหนดใบทุกโหนด เช่น มีส่วนของการคำนวณ หรือ มีการใช้สูตรต่างๆ หรือไม่

2. ร่องรอยที่ผิดพลาดประเภท Using cursor loops with DML



รูปที่ 3.20 แผนภาพต้นไม้ต้นแบบของ Using cursor loops with DML

คุณลักษณะ

- โพรซีเยอร์หรือฟังก์ชัน : เป็นได้ทั้งโพรซีเยอร์และฟังก์ชัน
- บริเวณที่พบร่องรอยที่ผิดพลาด : พบบริเวณ declaration section และ execution section
- บริบทของร่องรอยที่ผิดพลาด : ประกอบด้วยบริบทดังนี้

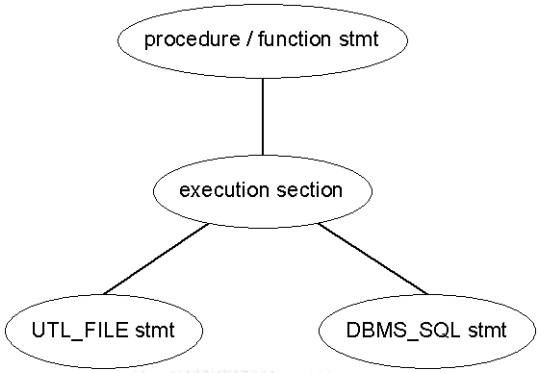
Declaration section ประกอบด้วย cursor stmt และ select stmt

Execution section ประกอบด้วย for stmt และ DML stmt

ข้อกำหนดจากการวิเคราะห์บริบท คือ

- ตรวจสอบในส่วนของ declaration section
- ตรวจสอบในส่วนของ execution section

3. ร่องรอยที่ผิดพลาดประเภท Initialize and Cleanup Logic in Execution Section



รูปที่ 3.21 แผนภาพต้นไม้ต้นแบบของ Initialize and Cleanup Logic in Execution Section

คุณลักษณะ

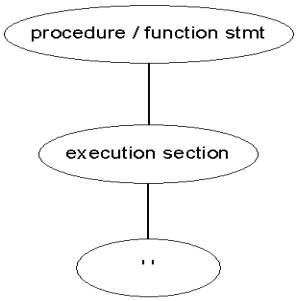
- โพรซีเยอร์หรือฟังก์ชัน : เป็นได้ทั้งโพรซีเยอร์และฟังก์ชัน
- บริเวณที่พบร่องรอยที่ผิดพลาด : พบบริเวณ execution section
- บริบทของร่องรอยที่ผิดพลาด : ประกอบด้วยบริบท UTL_FILE stmt และ DBMS_SQL stmt

ข้อกำหนดจากการวิเคราะห์บริบท คือ

- ตรวจสอบในส่วนของ execution section
- ตรวจสอบในส่วนของโครงสร้างต้องมีลักษณะของ Initialize and Cleanup ที่ชัดเจน เนื่องจากการตรวจสอบเมื่อมีการใช้เพียงแค่โครงสร้างเดียวนั้น คอมไพเลอร์จะให้ผ่านได้ จึงต้องตรวจสอบเพิ่มเติม คือ
    - เมื่อพบโครงสร้างของ UTL_FILE.fopen ต้องมีโครงสร้าง UTL_FILE.fclose

- เมื่อพบโครงสร้างของ DBMS_SQL.OPEN_CURSOR ต้องมีโครงสร้าง DBMS_SQL.CLOSE_CURSOR

4. ร่องรอยที่ผิดพลาดประเภท Use Literals



รูปที่ 3.22 แผนภาพต้นไม้ต้นแบบของ Use Literals

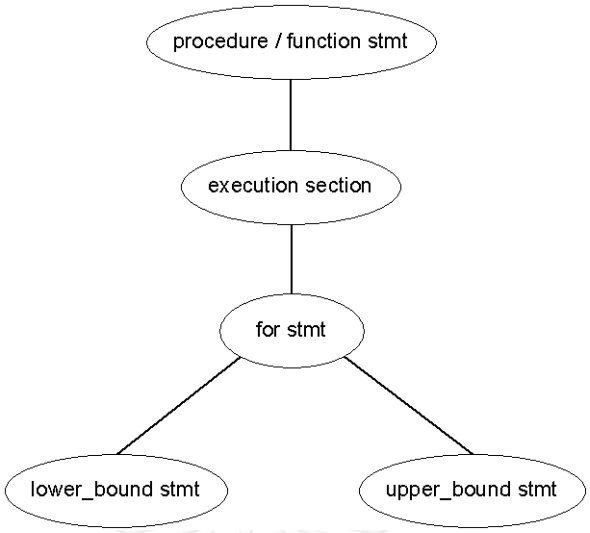
<u>คุณลักษณะ</u>

- โพรซีเยอร์หรือฟังก์ชัน : เป็นได้ทั้งโพรซีเยอร์และฟังก์ชัน
- บริเวณที่พบร่องรอยที่ผิดพลาด : พบบริเวณ execution section
- บริบทของร่องรอยที่ผิดพลาด : ประกอบด้วยบริบทที่เป็นสัญลักษณ์ ' ' ซึ่งเป็นสัญลักษณ์เครื่องหมายคำพูด (quotes)

<u>ข้อกำหนดจากการวิเคราะห์บริบท</u> คือ

- ตรวจสอบในส่วนของ execution section
- ตรวจสอบว่าต้องไม่อยู่ในส่วนของ Values
- ตรวจสอบว่าต้องไม่อยู่ในส่วนของ Where Condition

5. ร่องรอยที่ผิดพลาดประเภท No Range Values in Numeric FOR Loop



รูปที่ 3.23 แผนภาพต้นไม้ต้นแบบของ No Range Values in Numeric FOR Loop

<u>คุณลักษณะ</u>

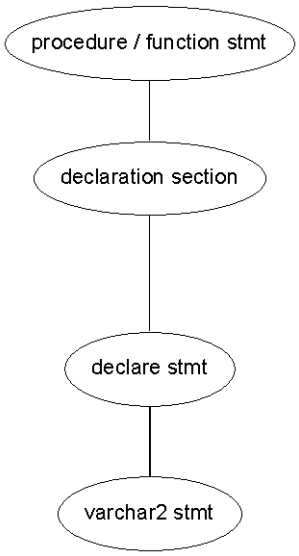
- โพรซีเยอร์หรือฟังก์ชัน : เป็นได้ทั้งโพรซีเยอร์และฟังก์ชัน
- บริเวณที่พบร่องรอยที่ผิดพลาด : พบบริเวณ execution section
- บริบทของร่องรอยที่ผิดพลาด : ประกอบด้วยบริบทดังนี้ for stmt , lower_bound stmt , upper_bound stmt

นิยาม: lower_bound คือ ค่าเริ่มต้นสำหรับ for stmt
upper_bund คือ ค่าสิ้นสุดสำหรับ for stmt

<u>ข้อกำหนดจากการวิเคราะห์บริบท</u> คือ

- ตรวจสอบในส่วนของ execution section
- ตรวจสอบว่า lower_bound และ upper_bound ต้องไม่เป็นค่าว่าง (NULL)
    - เมื่อพบโครงสร้าง เช่น FOR indx IN mylist.FIRST .. mylist.LAST ต้องมีการตรวจสอบค่าของ mylist.FIRST และ mylist.LAST ไม่ให้เป็นค่า NULL เนื่องจากถ้าค่าใดค่าหนึ่งมีค่าเป็น NULL จะทำให้เกิด Error ในระบบ

6. ร่องรอยที่ผิดพลาดประเภท Not Using SUBTYPES and Anchored Types



รูปที่ 3.24 แผนภาพต้นไม้ต้นแบบของ Not Using SUBTYPES and Anchored Types

<u>คุณลักษณะ</u>

- โพรซีเยอร์หรือฟังก์ชัน : เป็นได้ทั้งโพรซีเยอร์และฟังก์ชัน
- บริเวณที่พบร่องรอยที่ผิดพลาด : พบบริเวณ declaration section
- บริบทของร่องรอยที่ผิดพลาด : ประกอบด้วยบริบท VARCHAR2 stmt

<u>ข้อกำหนดจากการวิเคราะห์บริบท</u> คือ

- ตรวจสอบในส่วนของ declaration section
- ตรวจสอบในส่วนของการใช้ varchar2 ของฟังก์ชัน หรือ โพรซีเยอร์ ที่ไม่ได้อยู่ในส่วนของ declaration section จะต้องไม่ตรวจพบว่าเป็นร่องรอยที่ผิดพลาด เนื่องจากการใช้งาน varchar2 ในส่วนนี้ ไม่ได้อยู่ในส่วน declaration section ดังตัวอย่างซอร์ซโค้ดด้านล่างนี้

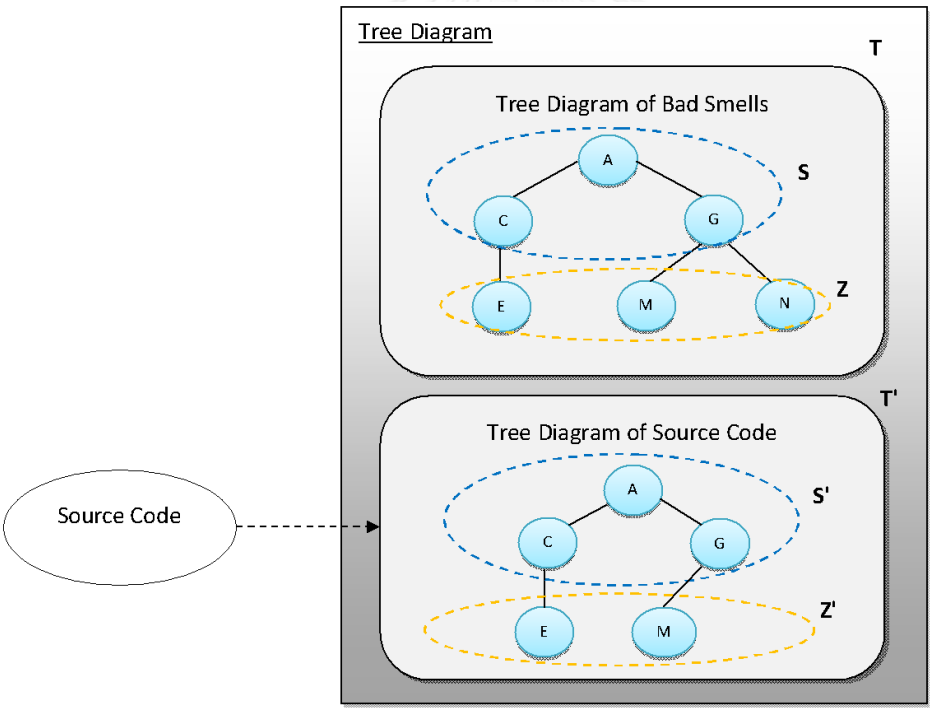
```
function get_budget_year (p_date date) return varchar2
Is
...
Begin
...
```

end;

โดยส่วนที่ต้องตรวจสอบจะเป็นส่วนระหว่าง Is ถึง Begin เนื่องจากโครงสร้างของภาษา PL/SQL ระบุว่าส่วนนั้น คือ declaration section

3.3 การตรวจหาร่องรอยที่ผิดพลาด

หลังจากได้โครงสร้างของแผนภาพต้นไม้เพื่อเป็นต้นแบบของร่องรอยที่ผิดพลาดทั้ง 6 ประเภท ที่ใช้ในงานวิจัยนี้แล้ว จะทำการสร้างแผนภาพต้นไม้ของซอร์ซโค้ดที่จะใช้ในการทดลอง เพื่อเปรียบเทียบโครงสร้างของแผนภาพต้นไม้ต้นแบบกับแผนภาพต้นไม้ของซอร์ซโค้ด ซึ่งมีอัลกอริทึมสำหรับการเปรียบเทียบ โดยมีรายละเอียด และรูปภาพประกอบ ดังรูปที่ 3.25 สำหรับการท่องไปในแผนภาพต้นไม้นั้นจะใช้การค้นหาแนวกว้างลงไปทีละระดับของแผนภาพต้นไม้ (Breadth-first Traversal)



รูปที่ 3.25 ภาพประกอบอัลกอริทึมการตรวจหาร่องรอยที่ผิดพลาด

อัลกอริทึมสำหรับการเปรียบเทียบแผนภาพต้นไม้ เพื่อหาร่องรอยที่ผิดพลาด มีดังนี้

1. ที่ Level ตั้งแต่ 1 จนถึง Level N-1 จะต้องมีโครงสร้างโหนดของแผนภาพต้นไม้ของซอร์ซโค้ด เหมือนกันกับ แผนภาพต้นไม้ต้นแบบ ก็คือจะต้องมีสมาชิกของเซทเหมือนกัน ซึ่งจากรูป 3.25 จะได้ข้อกำหนดของเซท คือ

   $\forall S' \in T' = \forall S \in T \longleftarrow$ Level 1, 2, ... , N-1

   จากรูปที่ 3.25 อธิบายรายละเอียดได้ดังนี้

   T คือ แผนภาพต้นไม้ต้นแบบ

   T' คือ แผนภาพต้นไม้ซอร์ซของโค้ด

   โดยที่ S และ S' จะต้องมีโหนดทุกโหนดที่เหมือนกัน หรือ สมาชิกในเซทเหมือนกัน

   S = {A, C, G}

   S' = {A, C, G}

นิยาม:

N คือ จำนวน Level ทั้งหมดของแผนภาพต้นไม้

2. ที่ Level N (Leaf Node) จะต้องมีโครงสร้างโหนดของแผนภาพต้นไม้ของซอร์ซโค้ดเป็นเซท หรือ สับเซท ของแผนภาพต้นไม้ต้นแบบ จากรูป 3.25 จะได้ข้อกำหนดของเซท คือ

   $Z' \in T'; Z \in T \mid \{Z' \in T'\} \subset \{Z \in T\} \longleftarrow$ Level N

   จากรูปที่ 3.25 อธิบายรายละเอียดได้ดังนี้

   โหนดของ Z' หรือ สมาชิกของ Z' เป็นสับเซท ของโหนดของ Z หรือ สมาชิกของ Z

   Z = {E, M, N}

   Z' = {E, M}

จากอัลกอริทึมทั้ง 2 ข้อนี้ ซึ่งถ้าโครงสร้างเป็นไปตามเงื่อนไขทั้ง 2 ข้อ แสดงว่าซอร์ซโค้ดนั้นมีร่องรอยที่ผิดพลาด โดยการค้นหาในซอร์ซโค้ดเพื่อหาโครงสร้างของแผนภาพต้นไม้ตามอัลกอริทึมที่เสนอนี้จะต้องใช้เงื่อนไขที่ได้จากการวิเคราะห์บริบท ประกอบกันด้วย เพื่อเป็นข้อกำหนดสำหรับการตรวจหาร่องรอยที่ผิดพลาดแต่ละประเภทว่าค้นหาบริเวณใดของซอร์ซโค้ด

โดยรายละเอียดตัวอย่างของการตรวจหาร่องรอยที่ผิดพลาดแต่ละประเภทที่ใช้ในงานวิจัยนี้มีรายละเอียดดังนี้

- ร่องรอยที่ผิดพลาดประเภท Business Logic in Exception Sections มี pseudo code ดังรูปที่ 3.26 และมีรายละเอียดดังนี้

```
    ----+----1----+----2----+----3----+----4----+----5----+
 1  PROGRAM Detect_BLES
 2     Detect_Flag Varchar2(1):='N';
 3  BEGIN
 4     Read_File (.txt or .sq.)
 5     Insert Into Table BLES_SOURCE
 6     Checking_Exception_Section(Excep_Flag);
 7     IF Excep_Flag THEN
 8        IF found DML stmt THEN
 9           getCheckBL;
10           IF CheckBL = TRUE THEN
11              Detection_Flag := 'Y';
12              getString;
13              getLine;
14              getTreeStruct;
15           ELSE
16              Detection_Flag := 'N';
17           END IF;
18        END IF;
19        IF found cursor stmt THEN
20          getCheckBL;
21          IF CheckBL = TRUE THEN
22             Detection_Flag := 'Y';
23             getString;
24             getLine;
25             getTreeStruct;
26          ELSE
27             Detection_Flag := 'N';
28          END IF;
29        IF ... THEN
30        .
31        .
32        END IF;
33        IF Detection_Flag = 'Y' THEN
34           Present_TreeDiagram;
35        END IF;
36     ELSE
37        Detection_Flag := 'N';
38     END IF;
39  END Detect_BLES;
```

รูปที่ 3.26 ภาพแสดง pseudo code ของการตรวจหา BLES

ทำการอ่านไฟล์เข้าสู่ระบบซึ่งมี 2 ประเภทคือ .txt และ .sql หลังจากนั้นทำการนำข้อมูลเข้าสู่ตารางข้อมูล ชื่อ BLES_SOURCE แล้วตรวจสอบโครงสร้างว่ามีส่วนของ Exception Section หรือไม่ โดยถ้าพบว่ามี ก็จะตรวจสอบต่อว่า มี DMS stmt หรือไม่ ถ้ามีก็จะตรวจสอบในส่วนของ Business Logic เพิ่มเติม โดยเรียกผ่านฟังก์ชัน โดยถ้าได้ค่าของฟังก์ชันเป็น TRUE ก็จะกำหนดค่าให้กับ Detection_Flag มีค่าเป็น Y และทำการเก็บค่าสตริง บรรทัด และ เก็บข้อมูลเป็นโครงสร้างของแผนภาพต้นไม้ ลงตารางข้อมูล หลังจากนั้นทำการตรวจสอบ stmt อื่นๆ ว่าพบหรือไม่จนครบทุกโหนดซึ่งตรวจสอบเหมือนกับการตรวจสอบ DMS stmt แล้วสุดท้ายจะตรวจสอบว่าค่าของ

Detection_Flag มีค่าเป็น Y หรือไม่ ซึ่งถ้าเป็น Y จะแสดงแผนภาพต้นไม้ของซอร์ซโค้ด โดยค่า Detection_Flag เป็น Y แสดงว่าเป็นร่องรอยที่ผิดพลาดประเภท BLES

- ร่องรอยที่ผิดพลาดประเภท Using cursor loops with DML

```
PROGRAM Detect_UCL
   Detect_Flag Varchar2(1):='N';
   v_found_left Varchar2(1):='N';
BEGIN
   Read_File (.txt or .sq.)
   Insert Into Table UCL_SOURCE
   Checking_Declaration_Section(Declare_Flag);
   IF Declare_Flag THEN
      IF found cursor stmt THEN
         getString;
         getLine;
         getTreeStruct;
         IF found select stmt THEN
            v_found_left := 'Y';
            getString;
            getLine;
            getTreeStruct;
         END IF;
      ELSE -- Not Found cursor stmt
         Detection_Flag := 'N';
      END IF;
   ELSE
      Detection_Flag := 'N';
   END IF;
   Checking_Execution_Section(Execute_Flag);
   IF Execute_Flag THEN
      IF found for stmt THEN
         getString;
         getLine;
         getTreeStruct;
         IF found insert stmt THEN
            IF v_found_left = 'Y' THEN
               Detection_Flag := 'Y';
               getString;
               getLine;
               getTreeStruct;
            END IF;
         END IF;
         IF found delete stmt THEN
            IF v_found_left = 'Y' THEN
               Detection_Flag := 'Y';
               getString;
               getLine;
               getTreeStruct;
            END IF;
         END IF;
         IF found update stmt THEN
            IF v_found_left = 'Y' THEN
               Detection_Flag := 'Y';
               getString;
               getLine;
               getTreeStruct;
            END IF;
         END IF;
         IF found select stmt THEN
            IF v_found_left = 'Y' THEN
               Detection_Flag = 'Y';
               getString;
               getLine;
               getTreeStruct;
            END IF;
         END IF;
         IF Detection_Flag = 'Y' THEN
            Present_TreeDiagram;
         END IF;
      ELSE -- Not Found for stmt
         Detection_Flag := 'N';
      END IF;
   ELSE
      Detection_Flag := 'N';
   END IF;
END Detect_BLES;
```

รูปที่ 3.27 ภาพแสดง pseudo code ของการตรวจหา UCL

ทำการอ่านไฟล์เข้าสู่ระบบซึ่งมี 2 ประเภทคือ .txt และ .sql หลังจากนั้นทำการนำข้อมูลเข้าสู่ตารางข้อมูล ชื่อ UCL_SOURCE แล้วตรวจสอบโครงสร้างว่ามีส่วนของ Declaration Section หรือไม่ โดยถ้าพบว่ามี ก็จะตรวจสอบต่อว่า มี cursor stmt หรือไม่ ถ้ามีก็จะตรวจสอบต่อว่า มี select stmt หรือไม่ ถ้ามี ก็จะกำหนดค่าให้กับ v_found_left เป็น Y หลังจากนั้นจะตรวจสอบโครงสร้างว่ามีส่วนของ Execution Section หรือไม่ โดยถ้าพบว่ามี ก็จะตรวจสอบว่า มี for stmt หรือไม่ ซึ่งถ้ามี ก็จะตรวจสอบว่ามี insert stmt , delete stmt , update stmt หรือ select stmt หรือไม่ โดยพบเพียงแค่อย่างใดอย่างนึง ก็จะกำหนดค่าของ Detection_Flag เป็น Y โดยเมื่อตรวจหาพบแต่ละขั้นตอนตามในซอร์ซโค้ดจะมีการ เก็บค่าสตริง บรรทัด และ เก็บข้อมูลเป็นโครงสร้างของแผนภาพต้นไม้ ลงตารางข้อมูล แล้วสุดท้ายจะตรวจสอบว่าค่าของ Detection_Flag มีค่าเป็น Y หรือไม่ ซึ่งถ้าเป็น Y จะแสดงแผนภาพต้นไม้ของซอร์ซโค้ด โดยค่า Detection_Flag เป็น Y แสดงว่าเป็นร่องรอยที่ผิดพลาดประเภท UCL

- ร่องรอยที่ผิดพลาดประเภท Initialize and Cleanup Logic in Execution Section

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+--
 1  PROGRAM Detect_ICE
 2     Detect_Flag Varchar2(1):='N';
 3  BEGIN
 4     Read_File (.txt or .sq.)
 5     Insert Into Table ICE_SOURCE
 6     Checking_Execution_Section(Execute_Flag);
 7     IF Execute_Flag THEN
 8        IF found UTL_FILE_FOPEN stmt and UTL_FILE_FCLOSE stmt THEN
 9           Detection_Flag := 'Y';
10           getString;
11           getLine;
12           getTreeStruct;
13        END IF;
14        IF found DBMS_SQL.OPEN_CURSOR stmt and DBMS_SQL.OPEN_CURSOR stmt THEN
15           Detection_Flag := 'Y';
16           getString;
17           getLine;
18           getTreeStruct;
19        END IF;
20        IF Detection_Flag = 'Y' THEN
21           Present_TreeDiagram;
22        END IF;
23     ELSE
24        Detection_Flag := 'N';
25     END IF;
26  END Detect_BLES;
```

รูปที่ 3.28 ภาพแสดง pseudo code ของการตรวจหา ICE

ทำการอ่านไฟล์เข้าสู่ระบบซึ่งมี 2 ประเภทคือ .txt และ .sql หลังจากนั้นทำการนำข้อมูลเข้าสู่ตารางข้อมูล ชื่อ ICE_SOURCE แล้วตรวจสอบโครงสร้างว่ามีส่วนของ Execution Section หรือไม่ โดยถ้าพบว่ามี ก็จะตรวจสอบต่อว่า มี UTL_FILE stmt หรือไม่ ถ้ามีก็จะกำหนดค่าให้กับ Detection_Flag มีค่าเป็น Y และทำการเก็บค่าสตริง บรรทัด และ เก็บข้อมูลเป็นโครงสร้างของแผนภาพต้นไม้ ลงตารางข้อมูล หลังจากนั้นก็จะตรวจสอบต่อว่า มี DBMS_SQL stmt หรือไม่ ถ้ามีก็จะกำหนดค่าให้กับ Detection_Flag มีค่าเป็น Y และทำการเก็บค่าสตริง บรรทัด และ เก็บข้อมูลเป็นโครงสร้างของแผนภาพต้นไม้ ลงตารางข้อมูล แล้วสุดท้ายจะตรวจสอบว่าค่าของ Detection_Flag มีค่าเป็น Y หรือไม่ ซึ่งถ้าเป็น Y จะแสดงแผนภาพต้นไม้ของซอร์ซโค้ด โดยค่า Detection_Flag เป็น Y แสดงว่าเป็นร่องรอยที่ผิดพลาดประเภท ICE

- ร่องรอยที่ผิดพลาดประเภท Use Literals

```
PROGRAM Detect_UL
   Detect_Flag Varchar2(1):='N';
BEGIN
   Read_File (.txt or .sq.)
   Insert Into Table UL_SOURCE
   Checking_Execution_Section(Execute_Flag);
   IF Execute_Flag THEN
      IF found ' ' stmt AND (' ' stmt NOT IN Values) AND (' ' stmt NOT IN Where Condition) THEN
         Detection_Flag := 'Y';
         getString;
         getLine;
         getTreeStruct;
      END IF;
      IF Detection_Flag = 'Y' THEN
         Present_TreeDiagram;
      END IF;
   ELSE
      Detection_Flag := 'N';
   END IF;
END Detect_BLES;
```

รูปที่ 3.29 ภาพแสดง pseudo code ของการตรวจหา UL

ทำการอ่านไฟล์เข้าสู่ระบบซึ่งมี 2 ประเภทคือ .txt และ .sql หลังจากนั้นทำการนำข้อมูลเข้าสู่ตารางข้อมูล ชื่อ UL_SOURCE แล้วตรวจสอบโครงสร้างว่ามีส่วนของ Execution Section หรือไม่ โดยถ้าพบว่ามี ก็จะตรวจสอบต่อว่า มี single quote stmt และ single quote จะต้องไม่อยู่ในส่วนของ values statement แลไม่อยู่ใน where condition statement ซึ่งถ้าตรงตามเงื่อนไขก็จะกำหนดค่าให้กับ Detection_Flag มีค่าเป็น Y และทำการเก็บค่าสตริง บรรทัด และ เก็บข้อมูลเป็นโครงสร้างของแผนภาพต้นไม้ ลงตารางข้อมูล แล้วสุดท้ายจะตรวจสอบว่าค่าของ Detection_Flag มี

ค่าเป็น Y หรือไม่ ซึ่งถ้าเป็น Y จะแสดงแผนภาพต้นไม้ของซอร์ซโค้ด โดยค่า Detection_Flag เป็น Y แสดงว่าเป็นร่องรอยที่ผิดพลาดประเภท UL

- ร่องรอยที่ผิดพลาดประเภท No Range Values in Numeric FOR Loop

```
----+----1----+----2----+----3----+----4----+----5----+----6---
1  PROGRAM Detect_NRNFL
2     Detect_Flag Varchar2(1):='N';
3  BEGIN
4     Read_File (.txt or .sq.)
5     Insert Into Table NRNFL_SOURCE
6     Checking_Execution_Section(Execute_Flag);
7     IF Execute_Flag THEN
8        IF found lower_bound THEN
9           IF No Check lower_bound and upper_bound is NULL Then
10             Detection_Flag := 'Y';
11             getString;
12             getLine;
13             getTreeStruct;
14          ELSE
15             Detection_Flag := 'N';
16          END IF;
17       END IF;
18       IF found upper_bound THEN
19          Detection_Flag := 'Y';
20          getString;
21          getLine;
22          getTreeStruct;
23       END IF;
24       IF Detection_Flag = 'Y' THEN
25          Present_TreeDiagram;
26       END IF;
27    ELSE
28       Detection_Flag := 'N';
29    END IF;
30 END Detect_BLES;
```

รูปที่ 3.30 ภาพแสดง pseudo code ของการตรวจหา NRNFL

ทำการอ่านไฟล์เข้าสู่ระบบซึ่งมี 2 ประเภทคือ .txt และ .sql หลังจากนั้นทำการนำข้อมูลเข้าสู่ตารางข้อมูล ชื่อ NRNFL_SOURCE แล้วตรวจสอบโครงสร้างว่ามีส่วนของ Execution Section หรือไม่โดยถ้าพบว่ามี ก็จะตรวจสอบต่อว่า มี lower_bound stmt หรือไม่ ถ้ามีก็จะกำหนดค่าให้กับ Detection_Flag มีค่าเป็น Y และทำการเก็บค่าสตริง บรรทัด และ เก็บข้อมูลเป็นโครงสร้างของแผนภาพต้นไม้ ลงตารางข้อมูล หลังจากนั้นก็จะตรวจสอบต่อว่า มี upper_bound stmt หรือไม่ ถ้ามีก็จะกำหนดค่าให้กับ Detection_Flag มีค่าเป็น Y และทำการเก็บค่าสตริง บรรทัด และ เก็บข้อมูล

เป็นโครงสร้างของแผนภาพต้นไม้ ลงตารางข้อมูล แล้วสุดท้ายจะตรวจสอบว่าค่าของ Detection_Flag มีค่าเป็น Y หรือไม่ ซึ่งถ้าเป็น Y จะแสดงแผนภาพต้นไม้ของซอร์ซโค้ด โดยค่า Detection_Flag เป็น Y แสดงว่าเป็นร่องรอยที่ผิดพลาดประเภท NRNFL

- ร่องรอยที่ผิดพลาดประเภท Not Using SUBTYPES and Anchored Types

```
    ----+----1----+----2----+----3----+----4----+---
 1  PROGRAM Detect_NUSAT
 2     Detect_Flag Varchar2(1):='N';
 3  BEGIN
 4     Read_File (.txt or .sq.)
 5     Insert Into Table NUSAT_SOURCE
 6     Checking_Declaration_Section(Declare_Flag);
 7     IF Declare_Flag THEN
 8        IF found declare stmt THEN
 9           getString;
10           getLine;
11           getTreeStruct;
12           IF found varchar2 stmt THEN
13              Detection_Flag := 'Y';
14              getString;
15              getLine;
16              getTreeStruct;
17           END IF;
18        END IF;
19        IF Detection_Flag = 'Y' THEN
20           Present_TreeDiagram;
21        END IF;
22     ELSE
23        Detection_Flag := 'N';
24     END IF;
25  END Detect_BLES;
```

รูปที่ 3.31 ภาพแสดง pseudo code ของการตรวจหา NUSAT

ทำการอ่านไฟล์เข้าสู่ระบบซึ่งมี 2 ประเภทคือ .txt และ .sql หลังจากนั้นทำการนำข้อมูลเข้าสู่ตารางข้อมูล ชื่อ NUSAT_SOURCE แล้วตรวจสอบโครงสร้างว่ามีส่วนของ Declaration Section หรือไม่ โดยถ้าพบว่ามี และไม่ใช่ส่วนของประเภทตัวแปรของโพรซีเยอร์หรือฟังก์ชัน ก็จะตรวจสอบต่อว่า มี varchar2 stmt หรือไม่ ถ้ามีก็จะกำหนดค่าให้กับ Detection_Flag มีค่าเป็น Y และทำการเก็บค่าสตริง บรรทัด และ เก็บข้อมูลเป็นโครงสร้างของแผนภาพต้นไม้ ลงตารางข้อมูล แล้วสุดท้ายจะตรวจสอบว่าค่าของ Detection_Flag มีค่าเป็น Y หรือไม่ ซึ่งถ้าเป็น Y จะแสดงแผนภาพต้นไม้ของซอร์ซโค้ด โดยค่า Detection_Flag เป็น Y แสดงว่าเป็นร่องรอยที่ผิดพลาดประเภท NUSAT

บทที่ 4
ออกแบบและพัฒนาเครื่องมือช่วยในการตรวจหาร่องรอยที่ผิดพลาด

ในบทนี้จะอธิบายถึงวิธีการออกแบบและพัฒนาเครื่องมือในการตรวจหาร่องรอยที่ผิดพลาดตามวิธีการที่ได้นำเสนอคือ แผนภาพต้นไม้และการวิเคราะห์บริบท โดยใช้แผนภาพยูเอ็มแอล (UML Diagram) ประกอบด้วย แผนภาพยูสเคส (Use case diagram) แผนภาพคลาส (Class diagram) และแผนภาพซีแควนซ์ (Sequence diagram)

4.1 สภาพแวดล้อมของการพัฒนาเครื่องมือ

เครื่องมือที่สร้างขึ้นเพื่อใช้ทดสอบวิธีการตรวจหาร่องรอยที่ผิดพลาดสำหรับงานวิจัยนี้พัฒนาด้วยภาษาวิชวลเบสิกดอทเน็ท (VB.NET) โดยมีรายละเอียดของสภาพแวดล้อม ดังตารางที่ 4.1

ตารางที่ 4.1 ตารางแสดงสภาพแวดล้อมของการพัฒนาเครื่องมือ

สภาพแวดล้อมด้านซอฟต์แวร์	**รายละเอียด**
ภาษาสำหรับการพัฒนาเครื่องมือ	วิชวลเบสิกดอทเน็ท (VB.NET)
เครื่องมือที่ใช้พัฒนา	Microsoft Visual Basic 2010
ปลั๊กอินสำหรับแสดงกราฟฟิก	DevExpress Version 17.1.3.0
ฐานข้อมูล (Database)	Oracle Database 11g Express Edition
ระบบปฏิบัติการ	Windows 7 Professional
สภาพแวดล้อมด้านฮาร์ดแวร์	**รายละเอียด**
คอมพิวเตอร์ส่วนบุคคล	หน่วยประมวลผล Intel Core i5-2450M 2.50 GHz
หน่วยความจำ	8 GB
หน่วยเก็บข้อมูล	1 TB

4.2 ความต้องการของการพัฒนาเครื่องมือเบื้องต้น

จากวิธีการตรวจหาร่องรอยที่ผิดพลาดที่ได้นำเสนอในบทที่ 3 สามารถนำมากำหนดความต้องการของเครื่องมือ เพื่อให้การพัฒนาเครื่องมือเป็นไปตามที่ได้เสนอวิธีการเอาไว้นั้น มีรายละเอียดดังนี้

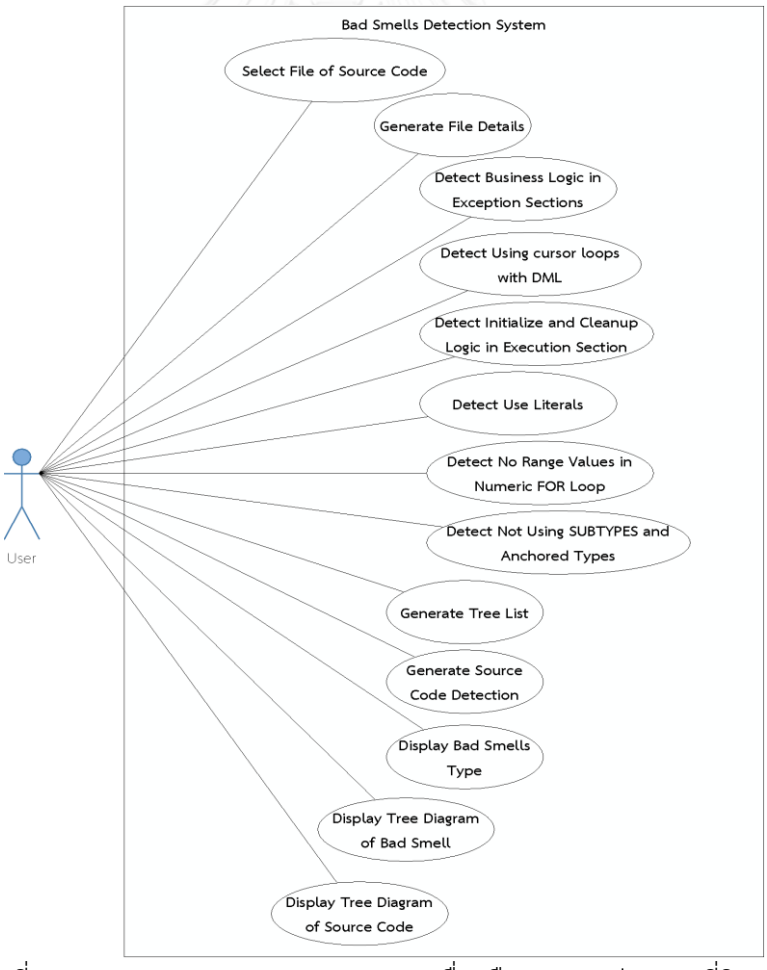
4.2.1 เครื่องมือจะต้องมีความสามารถในการนำเข้าข้อมูลของซอร์ซโค้ดที่เป็นเอกสารประเภท .txt หรือ .sql เพื่อใช้สำหรับการตรวจหาร่องรอยที่ผิดพลาดในซอร์ซโค้ดนั้นๆ

4.2.2 เครื่องมือจะต้องมีความสามารถในการวิเคราะห์บริบทตามที่กำหนดไว้ในข้อกำหนดจากการวิเคราะห์บริบท

4.2.3 เครื่องมือจะต้องมีความสามารถในการตรวจหาร่องรอยที่ผิดพลาดจากซอร์ซโค้ดที่ทำการนำเข้าข้อมูล โดยจะต้องสามารถระบุประเภทของร่องรอยที่ผิดพลาดได้ว่าเป็นประเภทใดจากทั้ง 6 ประเภทที่เลือกสำหรับงานวิจัยนี้ และแสดงแผนภาพต้นไม้ต้นของร่องรอยงที่ผิดพลาดและแผนภาพต้นไม้ของซอร์ซโค้ดได้

4.3 แผนภาพยูสเคส

แผนภาพยูสเคสของเครื่องมือสำหรับการตรวจหาร่องรอยที่ผิดพลาดจะแสดงความต้องการของระบบสำหรับการใช้งานเครื่องมือในมุมมองของผู้ใช้งาน (User) ดังรูปที่ 4.1 โดยจะแบ่งเป็น 3 กลุ่มหลักและมีรายละเอียด ดังนี้



รูปที่ 4.1 ภาพแสดงแผนภาพยูสเคสของเครื่องมือตรวจหาร่องรอยที่ผิดพลาด

4.3.1 ส่วนของการนำเข้าข้อมูล คือ ส่วนของการ Select File of Source Code ซึ่งจะสามารถเลือกไฟล์ที่ต้องการเข้าสู่ระบบได้

4.3.2 ส่วนของการตรวจหาร่องรอยที่ผิดพลาด คือ ส่วนของการประมวลผลจากข้อมูลนำเข้าเพื่อตรวจหาร่องรอยที่ผิดพลาด ประกอบด้วยร่องรอยที่ผิดพลาดและส่วนของการตรวจหาร่องรอยที่ผิดพลาด ดังตารางที่ 4.2

ตารางที่ 4.2 ตารางแสดงส่วนของการตรวจหาร่องรอยที่ผิดพลาด

ประเภทร่องรอยที่ผิดพลาด	ส่วนของการหาร่องรอยที่ผิดพลาด
Business Logic in Exception Sections	Detect Business Logic in Exception Sections
Using cursor loops with DML	Detect Using cursor loops with DML
Initialize and Cleanup Logic in Execution Section	Detect Initialize and Cleanup Logic in Execution Section
Use Literals	Detect Use Literals
No Range Values in Numeric FOR Loop	Detect No Range Values in Numeric FOR Loop
Not Using SUBTYPES and Anchored Types	Detect Not Using SUBTYPES and Anchored Types

4.3.3 ส่วนแสดงผลการตรวจหาร่องรอยที่ผิดพลาด คือ ส่วนของการแสดงผลจากการตรวจหาร่องรอยที่ผิดพลาดว่าพบร่องรอยที่ผิดพลาดหรือไม่และพบร่องรอยที่ผิดพลาดประเภทใด ประกอบด้วย ส่วนของการแสดงผล ดังนี้

1 Generate Tree List คือ ส่วนการสร้างเพื่อแสดงโครงสร้าง Tree List ของร่องรอยที่ผิดพลาดและของซอร์ซโค้ด

2 Generate Source Code Detection คือ ส่วนการสร้างเพื่อแสดงรายละเอียดของการตรวจหาร่องรอยที่ผิดพลาด เช่น บรรทัดที่พบร่องรอยที่ผิดพลาด

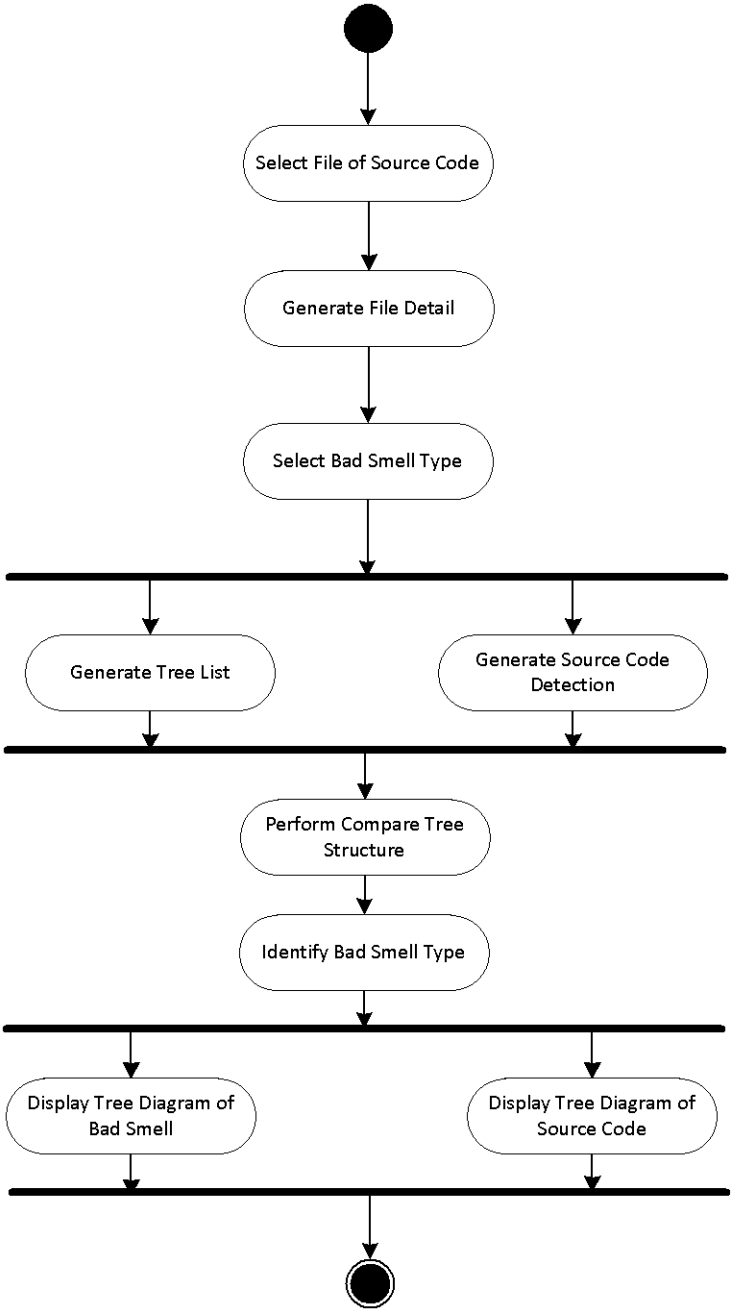
3 Display Bad Smells Type คือ ส่วนของการแสดงประเภทและรายละเอียดของร่องรอยที่ผิดพลาดที่ตรวจพบของซอร์ซโค้ด

4 Display Tree Diagram of Bad Smell คือ ส่วนการแสดงแผนภาพต้นไม้ของร่องรอยที่ผิดพลาด

5 Display Tree Diagram of Source Code คือ ส่วนของการแสดงแผนภาพต้นไม้ของซอร์ซโค้ด

4.4 แผนภาพกิจกรรม

แผนภาพกิจกรรมของเครื่องมือสำหรับการตรวจหาร่องรอยที่ผิดพลาด แสดงกิจกรรมต่างๆ ของเครื่องมือ ดังรูปที่ 4.2

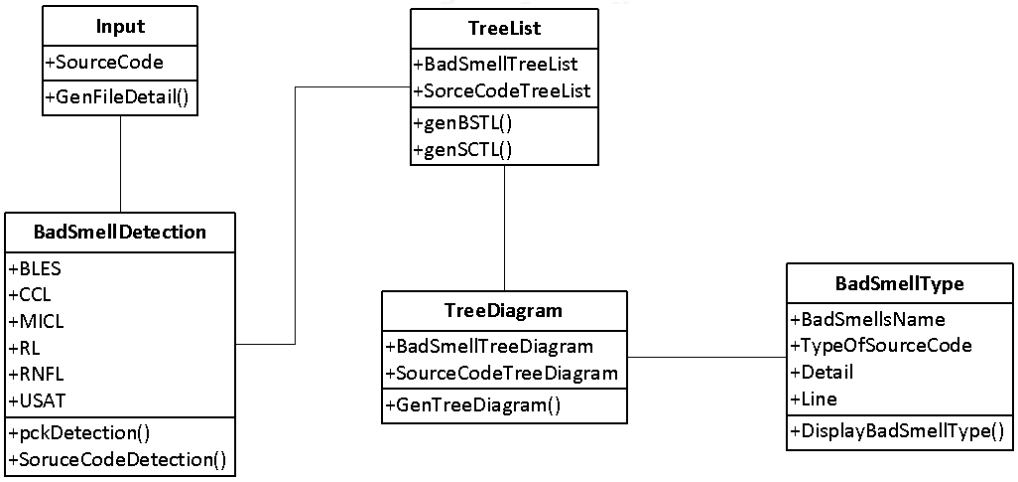


รูปที่ 4.2 ภาพแสดงแผนภาพกิจกรรมของเครื่องมือตรวจหาร่องรอยที่ผิดพลาด

จากรูปที่ 4.2 โดยมีกิจกรรมเริ่มต้นคือการเลือกไฟล์ของซอร์ซโค้ดที่ต้องการตรวจหาร่องรอยที่ผิดพลาด ซึ่งจะสามารถเลือกได้ คือ .txt และ .sql ถัดมาจะเป็นกิจกรรมของการสร้างรายละเอียดไฟล์ซึ่งจะแสดงโครงสร้างของซอร์ซโค้ดที่เลือก หลังจากนั้นทำการเลือกประเภทของร่องรอยที่ผิดพลาดที่ต้องการตรวจหา แล้วจะทำการสร้างแผนภาพต้นไม้แบบทรีลิส (Tree List) ขึ้นมา พร้อมกับการสร้างการตรวจหาซอร์ซโค้ดซึ่งส่วนนี้จะแสดงผลการตรวจหาร่องรอยที่ผิดพลาดในซอร์ซโค้ดตามโครงสร้างของแผนภาพต้นไม้ต้นแบบซึ่งจะแสดงรายละเอียดของโครงสร้างซอร์ซโค้ดและบรรทัดที่ตรวจพบร่องรอยที่ผิดพลาด จากนั้นเป็นกิจกรรมของการเปรียบเทียบโครงสร้างของแผนภาพต้นไม้ต้นแบบและแผนภาพต้นไม้ของซอร์ซโค้ด หลังจากนั้นเป็นกิจกรรมของการระบุชนิดของร่องรอยที่ผิดพลาดที่ตรวจพบว่ามีหรือไม่และเป็นชนิดใด ส่วนการทำงานของกิจกรรมสุดท้ายคือการแสดงแผนภาพต้นไม้ของร่องรอยที่ผิดพลาด และแสดงแผนภาพต้นไม้ของซอร์ซโค้ด

4.5 แผนภาพคลาส

แผนภาพคลาสของเครื่องมือสำหรับการตรวจหาร่องรอยที่ผิดพลาด จะแสดงแผนภาพของส่วนประกอบของระบบและความสัมพันธ์ของเครื่องมือที่ได้พัฒนา ซึ่งแบ่งออกเป็น 5 แผนภาพคลาสหลักๆ คือ แผนภาพคลาส Input แผนภาพคลาส BadSmellDetection แผนภาพคลาส TeeList แผนภาพคลาส TreeDiagram และ แผนภาพคลาส BadSmellType ดังรูปที่ 4.3 โดยมีรายละเอียดดังนี้



รูปที่ 4.3 ภาพแสดงแผนภาพคลาสของเครื่องมือตรวจหาร่องรอยที่ผิดพลาด

4.5.1 แผนภาพคลาส Input

เป็นคลาสส่วนที่เกี่ยวข้องกับการนำข้อมูลเข้าระบบโดยแสดงหน้าจอส่วนติดต่อกับผู้ใช้ระบบ เพื่อให้ผู้ใช้ระบบสามารถป้อนข้อมูลซอร์ซโค้ดที่เป็น .txt หรือ .sql เข้าระบบ โดยเมื่อป้อนข้อมูลเข้าระบบแล้วจะเรียกใช้งานคลาส GenFileDetail ซึ่งคลาสนี้จะทำการนำซอร์ซโค้ดนั้นแสดงที่หน้าจอในส่วนหน้าจอ File Detail ของระบบ เพื่อให้เห็นโครงสร้างทั้งหมดของซอร์ซโค้ดที่ได้นำเข้าสู่ระบบ

4.5.2 แผนภาพคลาส BadSmellDetection

เป็นคลาสที่ทำหน้าที่ในการตรวจหาร่องรอยที่ผิดพลาดของซอร์ซโค้ดที่นำเข้าสู่ระบบแล้วซึ่งสามารถเลือกประเภทของร่องรอยที่ผิดพลาดที่ต้องการค้นหา โดยเมื่อระบุร่องรอยที่ผิดพลาดที่ต้องการค้นหาแล้วจะมีการเรียกใช้งานคลาส pckDetection ซึ่งจะทำการค้นหาในซอร์ซโค้ดว่ามีโครงสร้างที่เหมือนแผนภาพต้นไม้ของร่องรอยที่ผิดพลาดที่เลือกหรือไม่ โดยระบบจะมีการเก็บโครงสร้างของแผนภาพต้นไม้ของร่องรอยที่ผิดพลาดทั้ง 6 ประเภทเอาไว้แล้ว และจะมีการเรียกใช้คลาส SourceCodeDetection เพื่อใช้ในการแสดงรายละเอียดของซอร์ซโค้ดว่าตรวจพบร่องรอยที่ผิดพลาดที่บรรทัดใดของซอร์ซโค้ดบ้าง

4.5.3 แผนภาพคลาส TeeList

เป็นคลาสที่ทำหน้าที่ในการแสดงโครงสร้างของร่องรอยที่ผิดพลาดที่เลือก กับแสดงโครงสร้างของซอร์ซโค้ดที่นำเข้าสู่ระบบ โดยแสดงเป็นลักษณะของทรีลิส ซึ่งถ้าผลของการตรวจหาร่องรอยที่ผิดพลาดไม่พบร่องรอยที่ผิดพลาด คลาสนี้จะแสดงผลทรีลิสเฉพาะของร่องรอยที่ผิดพลาดต้นแบบ แต่ถ้าตรวจพบร่องรอยที่ผิดพลาดก็จะแสดงรายละเอียดทรีลิสของร่องรอยที่ผิดพลาดที่พบในซอร์ซโค้ด โดยจะมีการเรียกใช้งานคลาส genBSTL และ คลาส genSTCL ซึ่งเป็นคลาสที่ทำหน้าที่ในการแสดงทรีลิส ของร่องรอยที่ผิดพลาดและของซอร์ซโค้ด ตามลำดับ

4.5.4 แผนภาพคลาส TreeDiagram

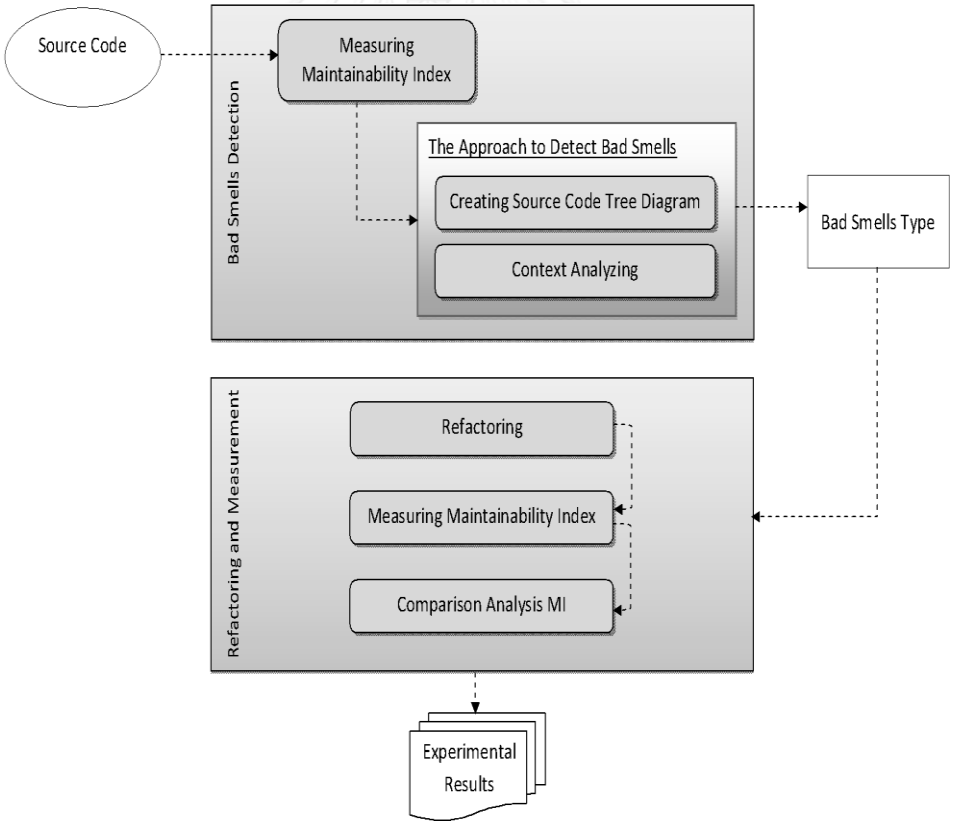
เป็นคลาสที่ทำหน้าที่ในการแสดงโครงสร้างของร่องรอยที่ผิดพลาดที่เลือก กับแสดงโครงสร้างของซอร์ซโค้ดที่นำเข้าสู่ระบบ โดยแสดงเป็นลักษณะของทรีไดอะแกรม โดยมีรายละเอียดแสดงเป็นลักษณะของโหนดและแสดงลำดับชั้นของแต่ละโหนด เพื่อให้เห็นโครงสร้างในรูปแบบทรีไดอะแกรม ซึ่งจะมีการเรียกใช้งานคลาส GenTreeDiagram โดยมีการเก็บโครงสร้างในลักษณะของทรี (Tree) ไว้ให้กับคลาส GenTreeDiagram ได้นำไปใช้ในการสร้างทรีไดอะแกรมได้ คือ มีการกำหนดรูทโหนด กำหนดโหนดพ่อแม่ และกำหนดโหนดลูก

4.5.5 แผนภาพคลาส BadSmellType

เป็นคลาสที่ทำหน้าที่แสดงผลลัพธ์ของการตรวจหาร่องรอยที่ผิดพลาด โดยแสดงรายละเอียดของชื่อร่องรอยที่ผิดพลาดที่พบ แสดงรายละเอียดของซอร์ซโค้ดที่นำเข้าสู่ระบบว่าเป็นซอร์ซโค้ดประเภทใด (Procedure หรือ Function) และแสดงรายละเอียดของซอร์ซโค้ดที่ตรวจหาพบพร้อมกับแสดงบรรทัดที่ตรวจพบ โดยดูจากโครงสร้างของบริบทที่อยู่ในแผนภาพต้นไม้ของร่องรอยที่ผิดพลาดแต่ละประเภท

บทที่ 5
การทดสอบความสามารถของเครื่องมือ

ผู้วิจัยได้ทำการวัดความสามารถของเครื่องมือเพื่อตรวจหาร่องรอยที่ผิดพลาด ที่ได้พัฒนาขึ้นด้วยวิธีการที่นำเสนอในงานวิจัยนี้ โดยการนำซอร์ซโค้ดที่จะใช้ในการทดลองทำการจำลองจุดเสีย (Fault Injection) และทำการวัดค่าดัชนีความสามารถในการบำรุงรักษา (Maintainability Index) ของซอร์ซโค้ดก่อนทำการรีแฟคทอริง หลังจากนั้นจะใช้เครื่องมือที่ได้พัฒนาขึ้นมานี้ ทำการตรวจหาร่องรอยที่ผิดพลาดเพื่อระบุประเภทร่องรอยที่ผิดพลาด และทำการรีแฟคทอริงซอร์ซโค้ดตามวิธีการสำหรับการแก้ไขที่ได้จาก PL/SQL Knowledge Xpert ดังที่ได้สรุปไว้แล้วในหัวข้อ 3.1 โดยหลังจากการรีแฟคทอริงซอร์ซโค้ดแล้วจะทำการวัดค่าดัชนีความสามารถในการบำรุงรักษาอีกครั้ง เพื่อใช้ในการเปรียบเทียบค่าความสามารถในการบำรุงรักษาสำหรับเป็นตัวชี้วัดหรือประเมินผลการทดลอง สำหรับงานวิจัยนี้ ขั้นตอนสำหรับการทดสอบความสามารถของเครื่องมือมีรายละเอียด ดังรูปที่ 5.1



รูปที่ 5.1 ภาพแสดงขั้นตอนการทดสอบความสามารถของเครื่องมือ

5.1 การวัดค่าดัชนีความสามารถในการบำรุงรักษาก่อนรีแฟคทอริง (Measuring Maintainability Index before Refactoring)

การทดสอบความสามารถของเครื่องมือในงานวิจัยนี้ จะเริ่มจากการวัดค่าดัชนีความสามารถในการบำรุงรักษาก่อนการทำรีแฟคทอริง โดยจะนำซอร์ซโค้ดที่เตรียมไว้เพื่อทำการทดลองสำหรับร่องรอยที่ผิดพลาดทั้ง 6 ประเภท มาทำการวัดค่าดัชนีความสามารถในการบำรุงรักษา ซึ่งขั้นตอนการวัดค่าจะต้องมีการเตรียมซอร์ซโค้ดสำหรับการทดลอง โดยอธิบายไว้ในหัวข้อ 5.1.2 และการวัดค่ามีรายละเอียดการคำนวณและเครื่องมือที่ใช้ ซึ่งจะอธิบายในหัวข้อ 5.1.3

5.1.2 การเตรียมร่องรอยที่ผิดพลาดของสโตร์โพรซีเยอร์

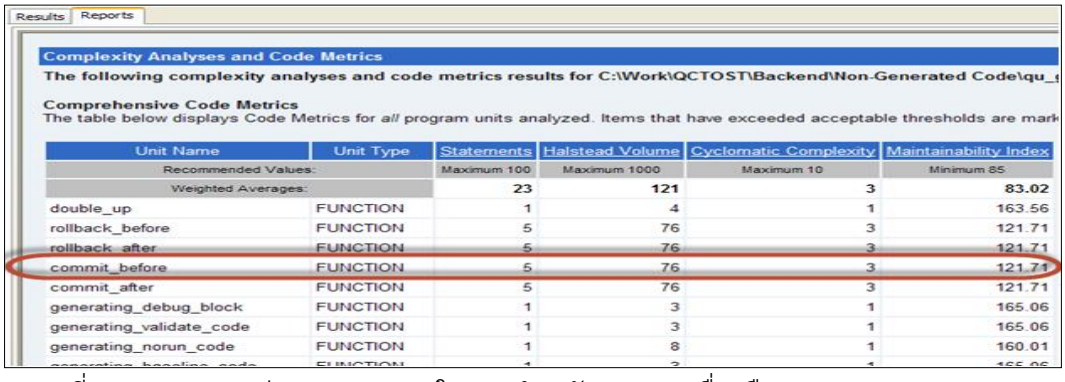
สำหรับการทดสอบความสามารถของเครื่องมือนี้ ผู้ทำวิจัยได้จัดเตรียมซอร์ซโค้ดไว้สำหรับทำการทดลองแบ่งเป็น 3 ชุด สำหรับร่องรอยที่ผิดพลาดทั้ง 6 ประเภท มีรายละเอียดดังตารางที่ 5.1

ตารางที่ 5.1 ตารางแสดงซอร์ซโค้ดสำหรับทำการทดลอง

ข้อมูลชุดที่	แหล่งที่มาของข้อมูล	Smell Injection	ประเภทไฟล์
1	PL/SQL Knowledge Expert	ไม่มี	.txt
2	PL/SQL Knowledge Expert	มี	.sql
3	ระบบงานของ สปสช	มี	.sql

5.1.3 การวัดค่าดัชนีความสามารถในการบำรุงรักษาก่อนการรีแฟคทอริง

งานวิจัยนี้จะใช้ค่าความสามารถในการบำรุงรักษา เพื่อประเมินผลการทดลองและใช้เครื่องมือที่ชื่อ TOAD's CodeXpert สำหรับช่วยคำนวณค่าความสามารถในการบำรุงรักษา การแสดงค่าของความสามารถในการบำรุงรักษาของเครื่องมือ ดังรูปที่ 5.2


Results Reports

Complexity Analyses and Code Metrics

The following complexity analyses and code metrics results for C:\Work\QCTOST\Backend\Non-Generated Code\qu_t

Comprehensive Code Metrics

The table below displays Code Metrics for *all* program units analyzed. Items that have exceeded acceptable thresholds are mark

Unit Name	Unit Type	Statements	Halstead Volume	Cyclomatic Complexity	Maintainability Index
Recommended Values:		Maximum 100	Maximum 1000	Maximum 10	Minimum 85
Weighted Averages:		23	121	3	83.02
double_up	FUNCTION	1	4	1	163.56
rollback_before	FUNCTION	5	76	3	121.71
rollback_after	FUNCTION	5	76	3	121.71
commit_before	FUNCTION	5	76	3	121.71
commit_after	FUNCTION	5	76	3	121.71
generating_debug_block	FUNCTION	1	3	1	165.06
generating_validate_code	FUNCTION	1	3	1	165.06
generating_norun_code	FUNCTION	1	8	1	160.01

รูปที่ 5.2 ภาพแสดงค่าความสามารถในการบำรุงรักษาของเครื่องมือ TOAD's CodeXpert

โดยจากการศึกษาเครื่องมือ เพื่อหาวิธีการที่เครื่องมือใช้ในการคำนวณ มีรายละเอียดดังนี้

aveV = average Halstead Volume V per module
aveV(g') = average Cyclomatic Complexity per module
aveLOC = the average count of Lines Of Code (LOC) per module
perCM = average percent of lines of Comment per module

Maintainability Index (MI) มีสูตรการคำนวณดังนี้
171 - 5.2 * ln(aveV) - 0.23 * aveV(G') - 16.2 * ln(aveLOC) + 50 * sin(sqrt(2.4 * perCM))

จากตัวอย่างรูปที่ 5.2 เมื่อแทนค่า Unit Name ที่ชื่อ commit_before จะได้ดังนี้

aveV = 76
aveV(g') = 3
aveLOC = 5

MI = 171 - 5.2 * ln(76) - 0.23 * (3) - 16.2 * ln(5) + 50 * sin(sqrt(2.4 * 0))
MI = 121.71
ซึ่งตรงกับค่าที่เครื่องมือแสดงไว้ในรายงานดังรูปที่ 5.2 ที่คอลัมน์ Maintainability Index

จากขั้นตอนการวัดค่าดัชนีความสามารถในการบำรุงรักษาก่อนรีแฟคทอริงนี้ จะได้ค่าดัชนีความสามารถในการบำรุงรักษาก่อนรีแฟคทอริง ดังตารางที่ 5.2

ตารางที่ 5.2 ตารางแสดงค่าดัชความสามารถในการบำรุงรักษาก่อนการรีแฟคทอริง

Bad Smells	**ข้อมูลชุดที่ 1**	**ข้อมูลชุดที่ 2**	**ข้อมูลชุดที่ 3**
	MI(1)	**MI(1)**	**MI(1)**
BLES	100.46	87.35	90.79
UCL	131.77	126.53	133.49
ICE	100.28	110.23	90.64
UL	112.91	109.14	108.50
NRNFL	145.59	134.45	91.75
NUSAT	147.60	145.59	146.13

นิยาม:

MI(1) คือ ค่าความสามารถในการบำรุงรักษาก่อนการรีแฟคทอริง

5.2 การสร้างแผนภาพต้นไม้ของซอร์ซโค้ด (Creating Source Code Tree Diagram)

หลังจากทำการวัดค่าความสามารถในการบำรุงรักษาแล้วขั้นตอนถัดมาคือการสร้างแผนภาพต้นไม้ของซอร์ซโค้ด คือการค้นหาในซอร์ซโค้ดด้วยโครงสร้างของแผนภาพต้นไม้ของร่องรอยที่ผิดพลาดเพื่อหาว่าในซอร์ซโค้ดนั้นๆ มีโครงสร้างของแผนภาพต้นไม้ของร่องรอยที่ผิดพลาดหรือไม่ ซึ่งถ้าการตรวจหาพบร่องรอยที่ผิดพลาดก็จะสร้างต้นไม้ของซอร์ซโค้ดนั้นๆ

การสร้างแผนภาพต้นไม้ของซอร์ซโค้ดนั้นจะทำการสร้างโดยอาศัยโครงสร้างต้นแบบของแผนภาพต้นไม้ของร่องรอยที่ผิดพลาดและใช้วิธีการของอัลกอริทึมที่ได้นำเสนอ คือ ตั้งแต่โหนดรากจนถึงโหนดในระดับก่อนโหนดใบจะต้องมีโครงสร้างที่เหมือนกัน คือ เป็นเซทเดียวกัน แต่สำหรับโหนดใบ เป็นเพียงสับเซทได้ ซึ่งถ้าตรวจพบโครงสร้างของแผนภาพต้นไม้ของร่องรอยที่ผิดพลาดในซอร์ซโค้ดและเป็นไปตามตามอัลกอริทึมก็จะทำการสร้างแผนภาพต้นไม้ของซอร์ซโค้ด

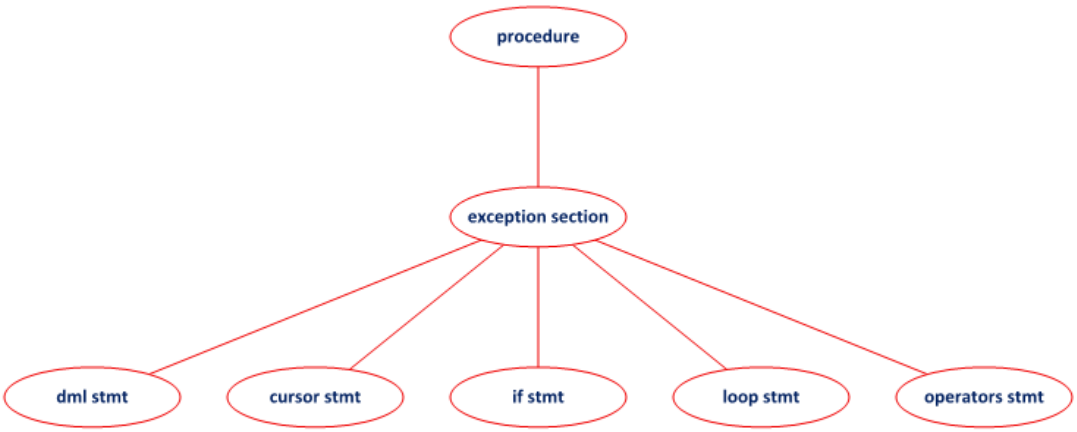
งานวิจัยนี้ได้ทำการทดสอบกับซอร์ซโค้ดสำหรับทำการทดลองทั้ง 3 ชุด โดยมีรายละเอียดของซอร์ซโค้ดและโครงสร้างของแผนภาพต้นไม้ของซอร์ซโค้ด ซึ่งเป็นโครงสร้างแผนภาพต้นไม้ที่สร้างได้จากซอร์ซโค้ด ดังนี้

การทดลองกับชุดข้อมูลที่ 1

- ร่องรอยที่ผิดพลาดประเภท BLES

```
2   PROCEDURE snc (NAME_IN IN VARCHAR2, objtype_in IN PLS_INTEGER)
3   IS
4     sch           VARCHAR2 (100);
5     part1         VARCHAR2 (100);
6     part2         VARCHAR2 (100);
7     dblink        VARCHAR2 (100);
8     part1_type    NUMBER;
9     object_number NUMBER;
10  .....
11  .....
12  .....
13  EXCEPTION
14    WHEN OTHERS
15    THEN
16      DECLARE
17        sqlc       NUMBER        := SQLCODE;
18        whoisowner VARCHAR2 (100) := '';
19        t_owner    VARCHAR2 (100);
20        t_name     VARCHAR2 (100);
21        r_owner    VARCHAR2 (100);
22        r_name     VARCHAR2 (100);
23        s_owner    VARCHAR2 (100);
24        CURSOR c1
25        IS
26          SELECT owner
27               ,table_owner
28               ,table_name
29            FROM ALL_SYNONYMS
30           WHERE synonym_name = r_name;
31        FUNCTION get_owner (NAME IN VARCHAR2)
32          RETURN VARCHAR2
33        AS
34          retval  VARCHAR2 (100) := '';
35          pos     INTEGER;
36        BEGIN
37  .....
38  .....
39  .....
```

รูปที่ 5.3 ภาพแสดงบางส่วนของซอร์ซโค้ดที่ใช้ในการทดลองเพื่อตรวจหา BLES (ชุดข้อมูลที่ 1)

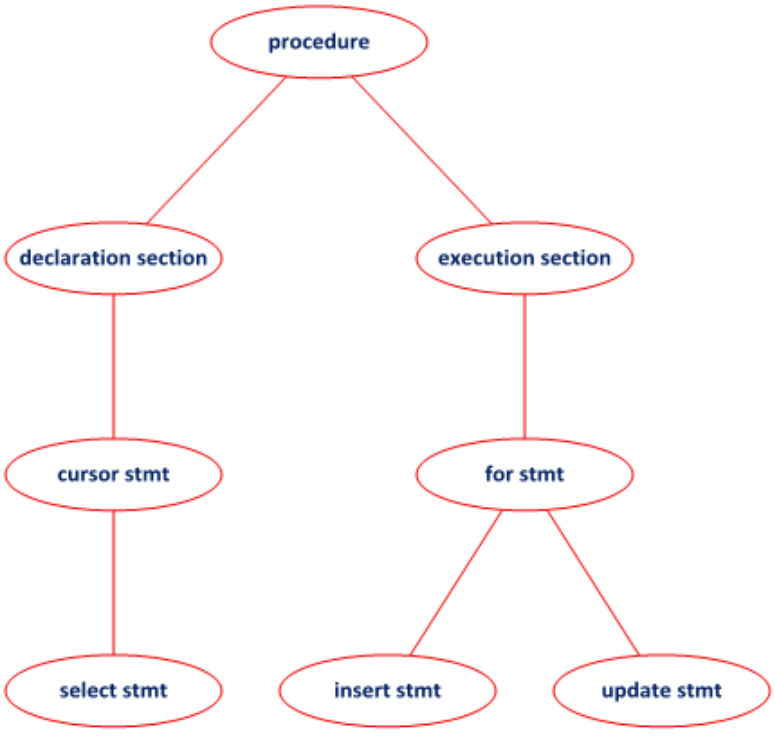


รูปที่ 5.4 ภาพแสดงแผนภาพต้นไม้ของซอร์ซโค้ดจากการตรวจหา BLES (ชุดข้อมูลที่ 1)

- ร่องรอยที่ผิดพลาดประเภท UCL

```
2  PROCEDURE upd_for_dept1 (
3    dept_in IN employee.department_id%TYPE
4   ,newsal IN employee.salary%TYPE) IS
5    CURSOR emp_cur IS
6      SELECT employee_id, salary, hire_date
7       FROM employee
8       WHERE department_id = dept_in;
9  BEGIN
10   FOR rec IN emp_cur
11   LOOP
12     INSERT INTO employee_history (employee_id, salary, hire_date)
13         VALUES (rec.employee_id, rec.salary, rec.hiredate);
14     UPDATE employee
15         SET salary = newsal,
16         hire_date = rec.hiredate
17         WHERE employee_id = rec.employee_id;
18   END LOOP;
19 END upd_for_dept1;
```

รูปที่ 5.5 ภาพแสดงส่วนของซอร์ซโค้ดที่ใช้ในการทดลองเพื่อตรวจหา UCL (ชุดข้อมูลที่ 1)



รูปที่ 5.6 ภาพแสดงแผนภาพต้นไม้ของซอร์ซโค้ดจากการตรวจหา UCL (ชุดข้อมูลที่ 1)

- ร่องรอยที่ผิดพลาดประเภท ICE

```
FUNCTION eqfiles (
  check_this_in       IN  VARCHAR2
 ,check_this_dir_in   IN  VARCHAR2
 ,against_this_in     IN  VARCHAR2
 ,against_this_dir_in IN  VARCHAR2 := NULL
) RETURN BOOLEAN IS
  checkid      UTL_FILE.file_type;
  checkline    VARCHAR2 (32767);
  check_eof    BOOLEAN;
  againstid    UTL_FILE.file_type;
  againstline  VARCHAR2 (32767);
  against_eof  BOOLEAN;
  retval       BOOLEAN;
BEGIN
  checkid :=
    UTL_FILE.fopen (check_this_dir_in
                   ,check_this_in
                   ,'R'
                   ,max_linesize     => 32767
                   );
  againstid :=
    UTL_FILE.fopen (NVL (against_this_dir_in, check_this_dir_in)
                   ,against_this_in
                   ,'R'
                   ,max_linesize     => 32767
                   );
  LOOP
    get_next_line_from_file (checkid, checkline, check_eof);
    get_next_line_from_file (againstid, againstline, against_eof);
    IF (check_eof AND against_eof) THEN
      retval := TRUE;
      EXIT;
    ELSIF (checkline != againstline) OR (check_eof OR against_eof) THEN
      retval := FALSE;
      EXIT;
    END IF;
  END LOOP;
  UTL_FILE.fclose (checkid);
  UTL_FILE.fclose (againstid);
  RETURN retval;
END eqfiles;
```

รูปที่ 5.7 ภาพแสดงส่วนของซอร์ซโค้ดที่ใช้ในการทดลองเพื่อตรวจหา ICE (ชุดข้อมูลที่ 1)

รูปที่ 5.8 ภาพแสดงแผนภาพต้นไม้ของซอร์ซโค้ดจากการตรวจหา ICE (ชุดข้อมูลที่ 1)

- ร่องรอยที่ผิดพลาดประเภท UL

```
2   PROCEDURE B_001 (p_hcode varchar2) IS
3   BEGIN
4   DECLARE
5     l_file_id  UTL_FILE.file_type;
6     l_line     VARCHAR2 (32767);
7   BEGIN
8     l_file_id :=
9       UTL_FILE.fopen ('c:\temp'
10                    ,'install.log'
11                    ,'R'
12                    ,max_linesize      => 32767
13                    );
14    LOOP
15      BEGIN
16        UTL_FILE.get_line (l_file_id, l_line);
17      EXCEPTION
18        WHEN NO_DATA_FOUND
19        THEN
20          EXIT;
21      END;
22      DBMS_OUTPUT.PUT_LINE (l_line);
23    END LOOP;
24    UTL_FILE.fclose (l_file_id);
25  END;
26  END;
```

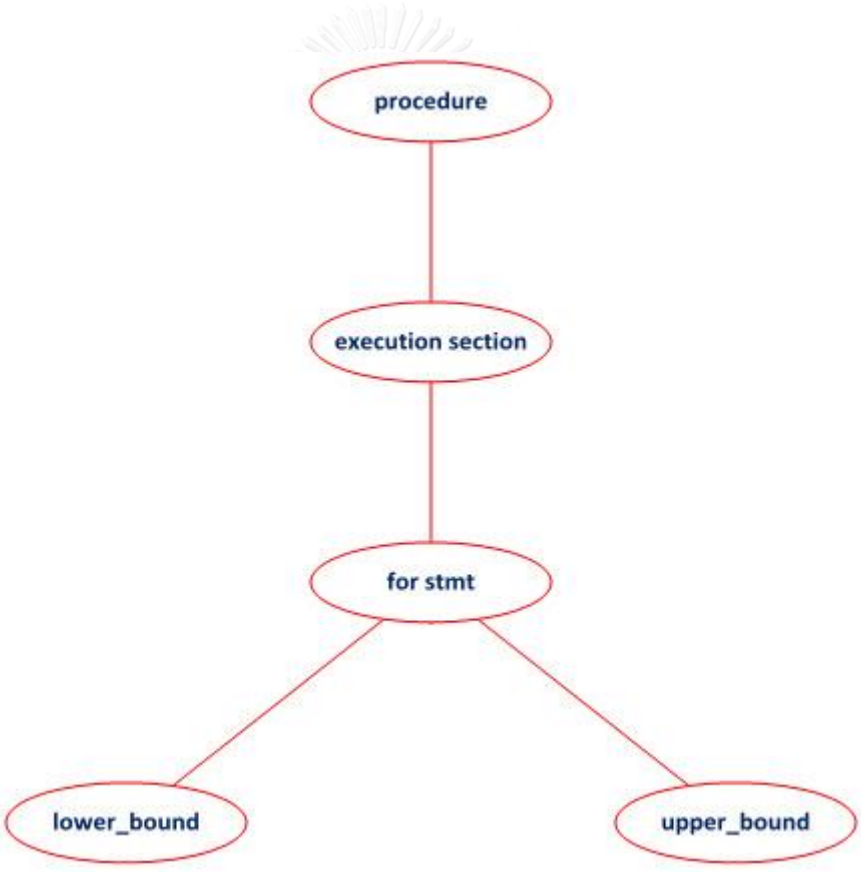
รูปที่ 5.9 ภาพแสดงส่วนของซอร์ซโค้ดที่ใช้ในการทดลองเพื่อตรวจหา UL (ชุดข้อมูลที่ 1)

รูปที่ 5.10 ภาพแสดงแผนภาพต้นไม้ของซอร์ซโค้ดจากการตรวจหา UL (ชุดข้อมูลที่ 1)

- ร่องรอยที่ผิดพลาดประเภท NRNFL

```
2   PROCEDURE process_collection (
3     mylist IN my_nested_table_type
4     )
5   IS
6   BEGIN
7     FOR indx IN mylist.FIRST .. mylist.LAST
8     LOOP
9       process_list_value (mylist (indx));
10    END LOOP;
11  END;
```

รูปที่ 5.11 ภาพแสดงส่วนของซอร์ซโค้ดที่ใช้ในการทดลองเพื่อตรวจหา NRNFL (ชุดข้อมูลที่ 1)

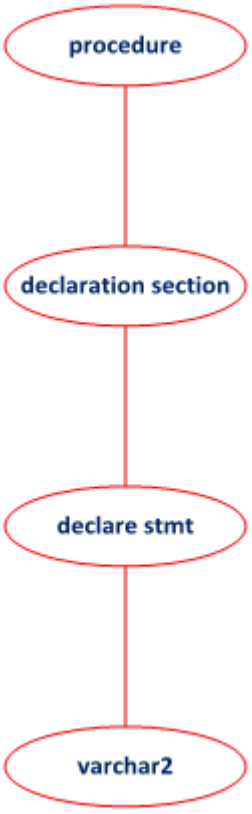


รูปที่ 5.12 ภาพแสดงแผนภาพต้นไม้ของซอร์ซโค้ดจากการตรวจหา NRNFL (ชุดข้อมูลที่ 1)

- ร่องรอยที่ผิดพลาดประเภท NUSAT

```
2   PROCEDURE USAT_001 (p_hcode varchar2) IS
3   BEGIN
4      DECLARE
5           l_company_name   VARCHAR2 (100);
6           l_description    VARCHAR2 (10000);
7           l_summary        VARCHAR2 (2000);
8           l_full_name      VARCHAR2 (200);
9      BEGIN
10        NULL;
11     END;
12  END;
```

รูปที่ 5.13 ภาพแสดงส่วนของซอร์ซโค้ดที่ใช้ในการทดลองเพื่อตรวจหา NUSAT (ชุดข้อมูลที่ 1)



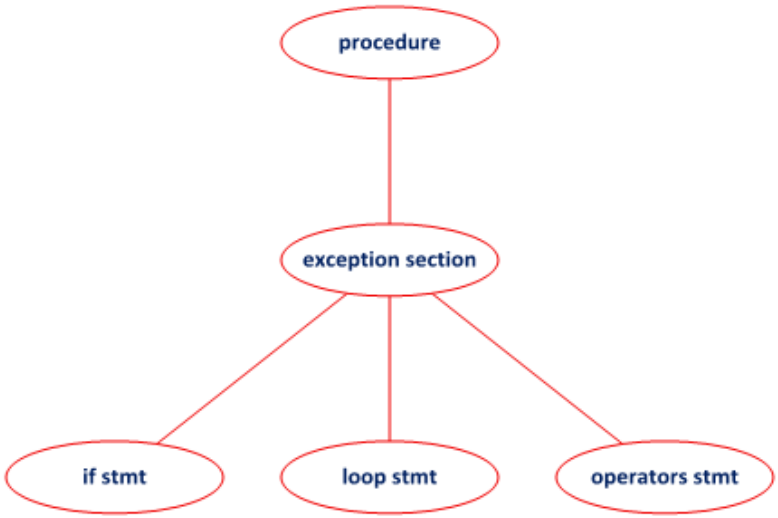
รูปที่ 5.14 ภาพแสดงแผนภาพต้นไม้ของซอร์ซโค้ดจากการตรวจหา NUSAT (ชุดข้อมูลที่ 1)

การทดลองกับชุดข้อมูลที่ 2

- ร่องรอยที่ผิดพลาดประเภท BLES

```
PROCEDURE snc_3 (NAME_IN IN VARCHAR2, objtype_in IN PLS_INTEGER)
IS
  object_number   NUMBER;
BEGIN
  IF object_number IS NULL
  THEN
    DBMS_OUTPUT.put_line (   'Name "'|| NAME_IN|| '" does not identify a valid object.');
  ELSE
    DBMS_OUTPUT.put_line ('Schema: ' || sch);
  END IF;
EXCEPTION
  WHEN NO_DATA_FOUND THEN
    BEGIN
      IF sqlc = -6564
      THEN
        r_name := get_name (NAME_IN);
        r_owner := get_owner (NAME_IN);
        IF r_owner IS NULL
        THEN
          r_owner := USER;
        END IF;
        OPEN c1;
        LOOP
          FETCH c1
           INTO s_owner
              ,t_owner
              ,t_name;
          EXIT WHEN c1%NOTFOUND;
          IF s_owner = r_owner
          THEN
            whoisowner := s_owner;
            EXIT;
          ELSIF s_owner = 'PUBLIC'
          THEN
            whoisowner := s_owner;
          END IF;
        END LOOP;
        CLOSE c1;
        IF (t_owner IS NULL) OR (s_owner = 'PUBLIC')
        THEN
          snc (t_name, objtype_in);
        ELSE
          snc (t_owner || '.' || t_name, objtype_in);
        END IF;
      ELSE
        RAISE;
      END IF;
    END;
END snc_3;
```

รูปที่ 5.15 ภาพแสดงส่วนของซอร์ซโค้ดที่ใช้ในการทดลองเพื่อตรวจหา BLES (ชุดข้อมูลที่ 2)

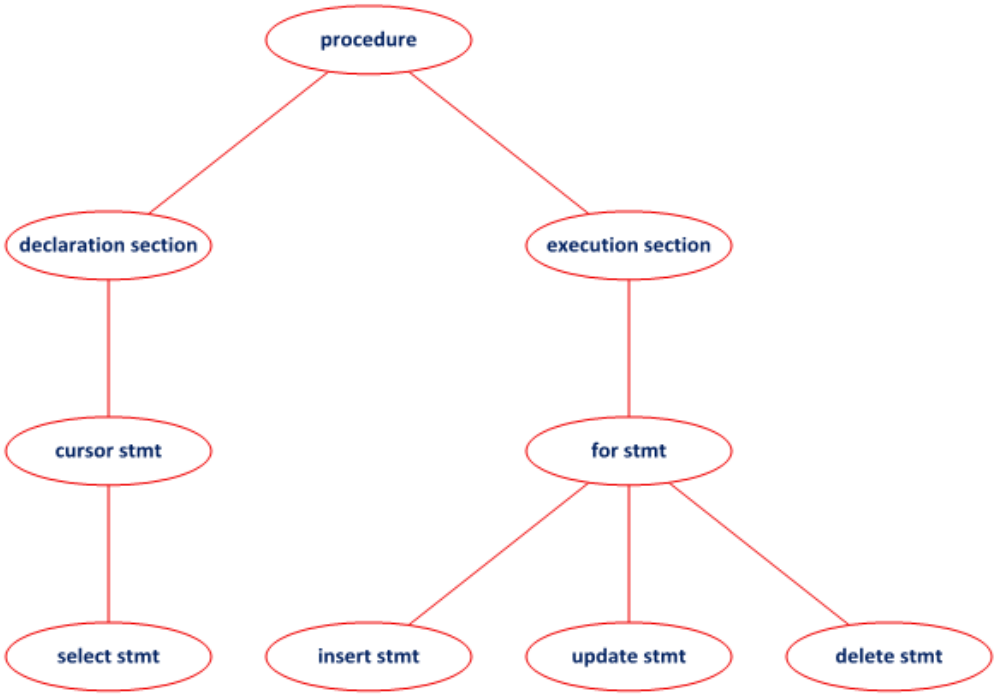


รูปที่ 5.16 ภาพแสดงแผนภาพต้นไม้ของซอร์ซโค้ดจากการตรวจหา BLES (ชุดข้อมูลที่ 2)

- ร่องรอยที่ผิดพลาดประเภท UCL

```
2  PROCEDURE upd_for_dept1 (
3    dept_in IN employee.department_id%TYPE
4   ,newsal IN employee.salary%TYPE)
5  IS
6    CURSOR emp_cur IS
7      SELECT employee_id, salary, hire_date
8        FROM employee
9       WHERE department_id = dept_in;
10 BEGIN
11   FOR rec IN emp_cur
12   LOOP
13     INSERT INTO employee_history (employee_id, salary, hire_date)
14         VALUES (rec.employee_id, rec.salary, rec.hiredate);
15     UPDATE employee
16         SET salary = newsal,
17          hire_date = rec.hiredate
18         WHERE employee_id = rec.employee_id;
19     DELETE employee
20         WHERE hire_date = sysdate;
21   END LOOP;
22 END upd_for_dept1;
```

รูปที่ 5.17 ภาพแสดงส่วนของซอร์ซโค้ดที่ใช้ในการทดลองเพื่อตรวจหา UCL (ชุดข้อมูลที่ 2)



รูปที่ 5.18 ภาพแสดงแผนภาพต้นไม้ของซอร์ซโค้ดจากการตรวจหา UCL (ชุดข้อมูลที่ 2)

- ร่องรอยที่ผิดพลาดประเภท ICE

```
FUNCTION fk_name
  (fk_id_in IN INTEGER,
   fk_table_in IN VARCHAR2,
   fk_id_col_in IN VARCHAR2,
   fk_nm_col_in IN VARCHAR2)
RETURN VARCHAR2 IS
  cur INTEGER := DBMS_SQL.OPEN_CURSOR;
  fdbk INTEGER;
  return_value VARCHAR2(100) := NULL;
BEGIN
  DBMS_SQL.PARSE
    (cur,
     'SELECT ' || fk_nm_col_in ||
     ' FROM ' || fk_table_in ||
     ' WHERE ' || fk_id_col_in || ' = :fk_value',
     DBMS_SQL.NATIVE);
  DBMS_SQL.BIND_VARIABLE (cur, 'fk_value', fk_id_in);
  DBMS_SQL.DEFINE_COLUMN (cur, 1, fk_nm_col_in, 100);
  fdbk := DBMS_SQL.EXECUTE (cur);
  fdbk := DBMS_SQL.FETCH_ROWS (cur);
  IF fdbk > 0
  THEN
    DBMS_SQL.COLUMN_VALUE (cur, 1, return_value);
  END IF;
  DBMS_SQL.CLOSE_CURSOR (cur);
  RETURN return_value;
END;
```

รูปที่ 5.19 ภาพแสดงส่วนของซอร์ซโค้ดที่ใช้ในการทดลองเพื่อตรวจหา ICE (ชุดข้อมูลที่ 2)

รูปที่ 5.20 ภาพแสดงแผนภาพต้นไม้ของซอร์ซโค้ดจากการตรวจหา ICE (ชุดข้อมูลที่ 2)

- ร่องรอยที่ผิดพลาดประเภท UL

```
2   PROCEDURE RL_002(P_CID in varchar2) is
3     l_file_id  UTL_FILE.file_type;
4     l_line     VARCHAR2 (32767);
5   BEGIN
6     l_file_id :=
7       UTL_FILE.fopen ('c:\temp'
8                   ,'install.log'
9                   ,'R'
10                  ,max_linesize     => 32767
11                  );
12    LOOP
13      BEGIN
14        UTL_FILE.get_line (l_file_id, l_line);
15      IF P_CID = '3869900003972' THEN
16       Select Count(*)
17       Into v_count
18       From M57_PERSON;
19      END IF;
20
21      EXCEPTION WHEN NO_DATA_FOUND THEN
22        EXIT;
23      END;
24      DBMS_OUTPUT.PUT_LINE (l_line);
25    END LOOP;
26    UTL_FILE.fclose (l_file_id);
27  END;
```

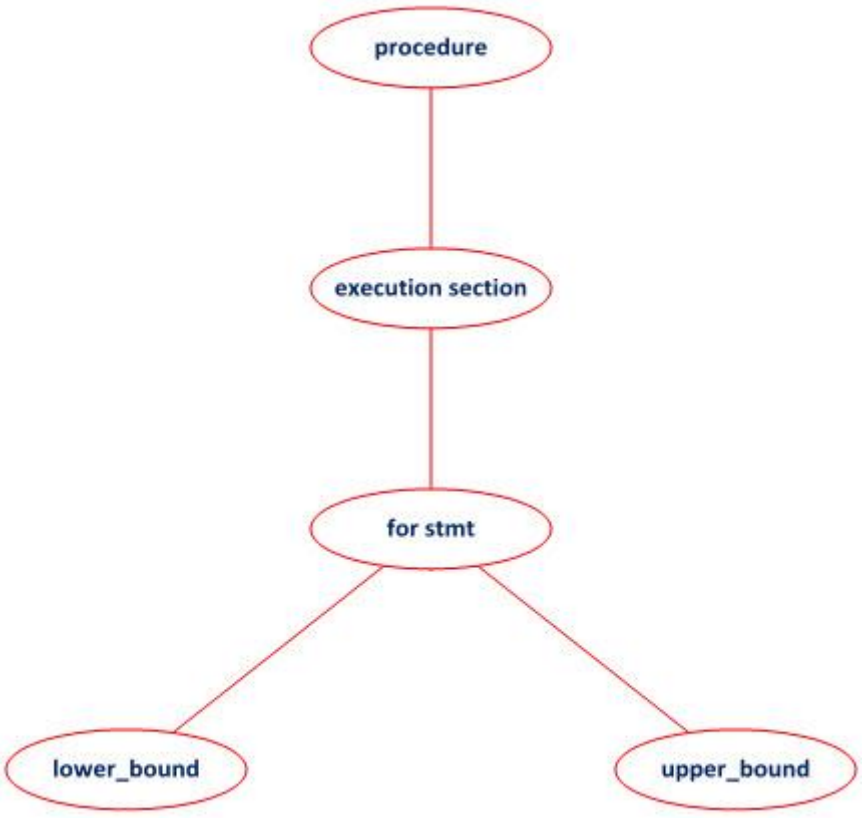
รูปที่ 5.21 ภาพแสดงส่วนของซอร์ซโค้ดที่ใช้ในการทดลองเพื่อตรวจหา UL (ชุดข้อมูลที่ 2)

รูปที่ 5.22 ภาพแสดงแผนภาพต้นไม้ของซอร์ซโค้ดจากการตรวจหา UL (ชุดข้อมูลที่ 2)

- ร่องรอยที่ผิดพลาดประเภท NRNFL

```
2   PROCEDURE process_collection (
3     mylist IN my_nested_table_type
4     )
5   IS
6   BEGIN
7     FOR indx IN mylist.FIRST .. mylist.LAST
8     LOOP
9      Insert Into M57_PERSON(CID,HCODE,HIST_HMAIN)
10     Select CID,HCODE,HIST_HMAIN
11     From TMP57_PERSON
12     Where HCODE = '05811';
13      process_list_value (mylist (indx));
14    END LOOP;
15  END;
```

รูปที่ 5.23 ภาพแสดงส่วนของซอร์ซโค้ดที่ใช้ในการทดลองเพื่อตรวจหา NRNFL (ชุดข้อมูลที่ 2)

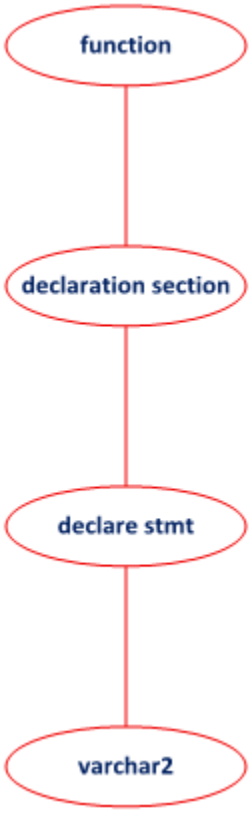


รูปที่ 5.24 ภาพแสดงแผนภาพต้นไม้ของซอร์ซโค้ดจากการตรวจหา NRNFL (ชุดข้อมูลที่ 2)

- ร่องรอยที่ผิดพลาดประเภท NUSAT

```
2   FUNCTION USAT_002 (p_hcode varchar2) return number IS
3   BEGIN
4      DECLARE
5           l_company_name   VARCHAR2 (100);
6           l_description    VARCHAR2 (10000);
7           l_summary        VARCHAR2 (2000);
8           l_full_name      VARCHAR2 (200);
9      BEGIN
10       return  1234;
11     END;
12  END;
```

รูปที่ 5.25 ภาพแสดงส่วนของซอร์ซโค้ดที่ใช้ในการทดลองเพื่อตรวจหา NUSAT (ชุดข้อมูลที่ 2)



รูปที่ 5.26 ภาพแสดงแผนภาพต้นไม้ของซอร์ซโค้ดจากการตรวจหา NUSAT (ชุดข้อมูลที่ 2)

การทดลองกับชุดข้อมูลที่ 3

- ร่องรอยที่ผิดพลาดประเภท BLES

```
procedure RUN_OP_VISIT(P_YYYY in varchar2
                      ,P_MM in varchar2
                      ,P_GYEAR in varchar2) is

 V_SQLSTR varchar2(5000);
 V_GYEAR varchar2(4) := P_GYEAR;
 V_MYEAR varchar2(2) := '';

begin

IF (P_GYEAR is null) THEN
  if (to_number(to_char(sysdate,'mm'))>9) then
    V_GYEAR := to_number(to_char(sysdate,'yyyy')+544);
  else
    V_GYEAR := to_number(to_char(sysdate,'yyyy')+543);
  end if;
 ELSE
      V_GYEAR := P_GYEAR;
 END IF;

IF(to_number(P_MM) > 9)THEN
       V_MYEAR := to_number(to_char(substr(P_GYEAR, 3)- 1)) ;
 ELSE
       V_MYEAR := to_number(to_char(substr(P_GYEAR, 3)));
 END IF;

 INSERT_LOGS(yyyy => p_yyyy
           ,mm => p_mm
           ,msg => '<<<< START PROCESS >>>>'
           ,table_name => 'M'||V_GYEAR||'_'||'DIAGNOSIS'
           ,process_name => 'RUN_OP_VISIT'
           ,row_effect => null);
 delete from MIS_OPVISIT_DATA a
 where a.yyyy = p_yyyy
      and a.mm = P_MM
      and gyear = v_gyear;

 commit;

INSERT_LOGS(yyyy => p_yyyy
          ,mm => p_mm
          ,msg => 'START -> Insert from ' || 'M' || V_MYEAR
          ,table_name => 'M' || V_MYEAR || '_' || 'DIAGNOSIS'
          ,process_name => 'MIS_OPVISIT_DATA'
          ,row_effect => null);

V_SQLSTR := 'INSERT INTO MIS_OPVISIT_DATA
              SELECT a.HCODE
                  ,'|| V_GYEAR ||'
                  ,'|| P_YYYY ||'
                  ,'|| P_MM ||'
...
...
...
                  b.CATM_NHSO, c.NHSO_ZONE, c.PROVINCE_ID, a.DEATH_DATE';

execute immediate v_sqlstr;
commit;

exception when others then
 pkg_utils.insert_except_log ( sqlcode, 'MIS_OPVISIT_DATA', sqlerrm );

 INSERT_LOGS(yyyy => p_yyyy
           ,mm => p_mm
           ,msg => '<<<< FINISH PROCESS >>>>'
           ,table_name => 'M'||V_GYEAR||'_'||'DIAGNOSIS'
           ,process_name => 'RUN_OP_VISIT'
           ,row_effect => null);
end;
```

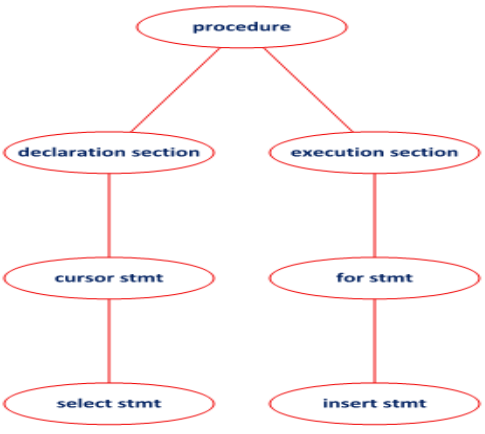
รูปที่ 5.27 ภาพแสดงบางส่วนของซอร์ซโค้ดที่ใช้ในการทดลองเพื่อตรวจหา BLES (ชุดข้อมูลที่ 3)

รูปที่ 5.28 ภาพแสดงแผนภาพต้นไม้ของซอร์ซโค้ดจากการตรวจหา BLES (ชุดข้อมูลที่ 3)

- ร่องรอยที่ผิดพลาดประเภท UCL

```
PROCEDURE REPORT_NUTRI_TBL11(P_QUARTER varchar2,
          P_NLEVEL varchar2,
          P_AGE_START varchar2,
          P_AGE_END varchar2) is
 CURSOR V_CURSOR IS

   SELECT
     HOSPITAL.NHSO_ZONENAME,
     HOSPITAL.PROVINCE_NAME,
     HOSPITAL.HNAME,
     HOSPITAL.HTYPE_NAME,
     HOSPITAL.TYPE_DESC,
     HIST_MAININSCL_UC,
     HIST_MAININSCL_OFC,
     HIST_MAININSCL_SSS,
     HIST_MAININSCL_OTHER
   FROM
     V_HOSPITAL HOSPITAL,

     (SELECT TEMP_NUTRI.HIST_HMAIN_OP AS HIST_HMAIN_OP_TEMP,
       HCODE AS HCODE_TEMP ,

       SUM(CASE WHEN nvl(HIST_MAININSCL,'XXX') in ('UC','UCS','WEL','XXX') THEN SUM_CID ELSE 0 END) as HIST_MAININSCL_UC,
       SUM(CASE WHEN nvl(HIST_MAININSCL,'XXX') in ('OFC') THEN SUM_CID ELSE 0 END) as HIST_MAININSCL_OFC,
       SUM(CASE WHEN nvl(HIST_MAININSCL,'XXX') in ('SSS') THEN SUM_CID ELSE 0 END) as HIST_MAININSCL_SSS,
       SUM(CASE WHEN nvl(HIST_MAININSCL,'XXX') not in('UC','UCS','WEL','XXX','OFC','SSS') THEN SUM_CID ELSE 0 END) as HIST_MAININSCL_OTHER

     FROM TEMP_M56_NUTRI_TBL11  TEMP_NUTRI
     WHERE QUARTER = P_QUARTER
         AND NLEVEL = P_NLEVEL
         AND TEMP_NUTRI.AGE_YEAR between P_AGE_START and P_AGE_END
     GROUP BY HIST_HMAIN_OP, HCODE)

   LEFT JOIN V_HOSPITAL H_HCODE
   ON H_HCODE.HMAIN = NVL(HCODE_TEMP, 'N')
   WHERE HOSPITAL.HMAIN = HIST_HMAIN_OP_TEMP;

begin
  FOR rec IN V_CURSOR
  LOOP
    INSERT INTO TMP_TEMPLATE_NUTRI_56
         (NHSO_ZONENAME,
          PROVINCE_NAME,
          HNAME,
          HTYPE_NAME,
          TYPE_DESC,
          HIST_MAININSCL_UC,
          HIST_MAININSCL_OFC,
          HIST_MAININSCL_SSS,
          HIST_MAININSCL_OTHER )
        VALUES (rec.NHSO_ZONENAME,
        rec.PROVINCE_NAME,
          rec.HNAME,
        rec.HTYPE_NAME,
          rec.TYPE_DESC,
          rec.HIST_MAININSCL_UC,
          rec.HIST_MAININSCL_OFC,
          rec.HIST_MAININSCL_SSS,
          rec.HIST_MAININSCL_OTHER );
  END LOOP;
end REPORT_NUTRI_TBL11;
```

รูปที่ 5.29 ภาพแสดงส่วนของซอร์ซโค้ดที่ใช้ในการทดลองเพื่อตรวจหา UCL (ชุดข้อมูลที่ 3)



รูปที่ 5.30 ภาพแสดงแผนภาพต้นไม้ของซอร์ซโค้ดจากการตรวจหา UCL (ชุดข้อมูลที่ 3)

- ร่องรอยที่ผิดพลาดประเภท ICE

```
FUNCTION CHECK_DMHT(BSLEVEL in varchar2,
                    BPL     in varchar2,
                    BPH     in varchar2,
             DMHT_FILE_DIR in varchar2,
             DMHT_FILE   in varchar2) return varchar2
is
  result varchar2(15);
  fHandle utl_file.file_type;
begin
  fHandle := utl_file.fopen(DMHT_FILE_DIR
                  ,DMHT_FILE
                  ,'W');
 IF (BSLEVEL IS NOT NULL AND ISNUMBER(BSLEVEL) = 'T') AND
    ((BPL IS NOT NULL AND ISNUMBER(BPL) = 'T') OR
     (BPH IS NOT NULL AND ISNUMBER(BPH) = 'T')) THEN

  IF (to_numb(BSLEVEL)between 0 and 99) and
     (to_numb(BPH) between 0 and 119) and
     (to_numb(BPL) between 0 and 79) THEN
    UTL_FILE.PUT_LINE(fHandle,BSLEVEL||','||BPL||','||BPH||' NORMAL');
    result := 'normal';
  ELSIF to_numb(BSLEVEL) between 100 and 125 and
        ((to_numb(BPH) between 120 and 139) or
         (to_numb(BPL) between 80 and 89)) then
    UTL_FILE.PUT_LINE(fHandle,BSLEVEL||','||BPL||','||BPH||' PRE');
    result := 'pre';
  ELSIF to_numb(BSLEVEL) >= 126 and
        (to_number(BPH) >= 140 or to_numb(BPL) between 90 and 99) then
    UTL_FILE.PUT_LINE(fHandle,BSLEVEL||','||BPL||','||BPH||' NEW_CASE');
    result := 'new_case';
  ELSE
    UTL_FILE.PUT_LINE(fHandle,BSLEVEL||','||BPL||','||BPH||' OTHER');
    result := 'OTHER';
  END IF;

 ELSE
  UTL_FILE.PUT_LINE(fHandle,BSLEVEL||','||BPL||','||BPH||' NULL');
  result := NULL;
 END IF;
    utl_file.fclose(fHandle);
    return(result);
end CHECK_DMHT;
```

รูปที่ 5.31 ภาพแสดงส่วนของซอร์ซโค้ดที่ใช้ในการทดลองเพื่อตรวจหา ICE (ชุดข้อมูลที่ 3)

รูปที่ 5.32 ภาพแสดงแผนภาพต้นไม้ของซอร์ซโค้ดจากการตรวจหา ICE (ชุดข้อมูลที่ 3)

- ร่องรอยที่ผิดพลาดประเภท UL

```
FUNCTION CHECK_DM_2(P_DMFAMILY   in number,
                    P_DEATH_DATE in DATE,
                    P_DATE_EXAM  in DATE,
                    P_HEIGHT     in varchar2,
                    P_WEIGHT     in varchar2,
                    P_WAIST_CM   in varchar2,
                    P_HTFAMILY   in varchar2,
                    P_BPL_2      in varchar2,
                    P_BPH_2      in varchar2) return varchar2 is
  v_result varchar2(6);

BEGIN

  IF (nvl(P_BPL_2, 0) != 0 OR nvl(P_BPH_2, 0) != 0) AND(P_DEATH_DATE > P_DATE_EXAM or P_DEATH_DATE is null) THEN

     v_result := 'HT';

  ELSIF (nvl(P_DMFAMILY, 0) IN (1, 2, 9)
   AND nvl(P_HTFAMILY, 0) IN (1, 2, 9)
   AND((to_numb(P_WEIGHT) + to_numb(P_HEIGHT)) IS NOT NULL OR to_numb(P_WAIST_CM) IS NOT NULL)
   AND(P_DEATH_DATE > P_DATE_EXAM or P_DEATH_DATE is null)) THEN

     v_result := 'DM';

  ELSE
     v_result := 'OTHER';

  END IF;

 RETURN(v_result);

EXCEPTION  WHEN OTHERS THEN
   RETURN('OTHER1');
END CHECK_DM_2;
```

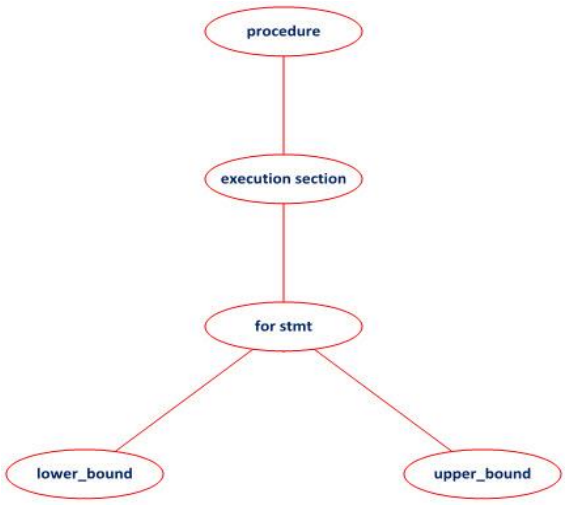
รูปที่ 5.33 ภาพแสดงส่วนของซอร์ซโค้ดที่ใช้ในการทดลองเพื่อตรวจหา UL (ชุดข้อมูลที่ 3)

รูปที่ 5.34 ภาพแสดงแผนภาพต้นไม้ของซอร์ซโค้ดจากการตรวจหา UL (ชุดข้อมูลที่ 3)

- ร่องรอยที่ผิดพลาดประเภท NRNFL

```
procedure run_fp_01(    p_yyyymm in varchar2,
                        p_s_yyyymm in varchar2,
                        p_e_yyyymm in varchar2,
                FP_list IN my_nested_table_type) is

  v_rows number;
  v_errsql varchar2(2000);
  v_proc varchar2(30) := 'FP 01';

begin
    insert into op_temp_logs values (systimestamp, '++Start :: ' || v_proc || ' --> ' || p_yyyymm, null);
    commit;

    begin
        delete from OP_FP_TEMP_ALL_PASS
        where yyyymm = p_yyyymm;
        v_rows := sql%rowcount;
        commit;
        insert into op_temp_logs values (systimestamp, 'Deleted Rows :: '||v_proc||' --> ' || p_yyyymm, v_rows);
        commit;
        v_rows := null;

     For indx In FP_list.First .. FP_list.Last
     Loop
        insert into OP_FP_TEMP_ALL_PASS (YYYYMM,HCODE,PROV1,RGN1
         ,F_ALL
         ,F_PASS
        )
        select
          p_yyyymm,a.hcode,a.prov1,a.rgn1
          ,sum(case when a.status_af = '1' then 1 else 0 end) sum_all
          ,sum(case when a.status_af = '1' and a.status = '1' then 1 else 0 end) sum_pass
        from m55_fp_pp a
        where to_char(a.date_send,'yyyymm') = p_yyyymm
          and to_char(a.date_serv,'yyyymm') between  p_s_yyyymm and p_e_yyyymm
        group by
          a.hcode
          ,a.prov1
          ,a.rgn1;
        v_rows := sql%rowcount;
     End Loop;
        commit;

    EXCEPTION
      WHEN OTHERS THEN
        v_errsql := sqlerrm;
        insert into op_temp_logs values (systimestamp, '**Error :: '||v_proc||' --> ' || p_yyyymm || ' :: ' || v_errsql, null);
        commit;
    END;

    insert into op_temp_logs values (systimestamp, '++Finish :: ' || v_proc || ' --> ' || p_yyyymm, v_rows);
    commit;
end;
```

รูปที่ 5.35 ภาพแสดงส่วนของซอร์ซโค้ดที่ใช้ในการทดลองเพื่อตรวจหา NRNFL (ชุดข้อมูลที่ 3)

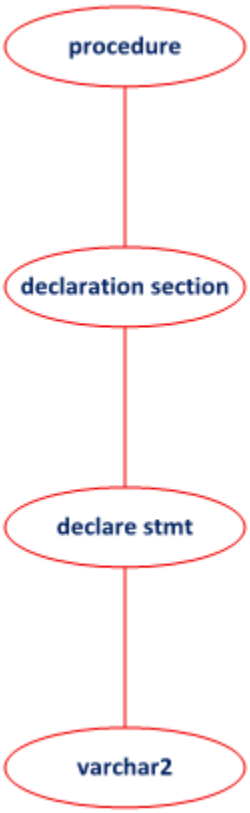


รูปที่ 5.36 ภาพแสดงแผนภาพต้นไม้ของซอร์ซโค้ดจากการตรวจหา NRNFL (ชุดข้อมูลที่ 3)

- ร่องรอยที่ผิดพลาดประเภท NUSAT

```
2  procedure sp_get_provlist(p_result  out ref_csr,
3                            p_zone in varchar2,
4                            p_province in varchar2) is
5      v_hcode varchar2(5):= '05811';
6  begin
7      open p_result for
8      select province_id, province_name, nhso_zone, nhso_zonename
9      from mv_l_province
10     where nhso_zone = nvl(p_zone,nhso_zone)
11       and province_id = nvl(p_province, province_id)
12       and nhso_zone not in ('14','15')
13       and used='Y'
14     order by nhso_zone, province_id;
15 end;
```

รูปที่ 5.37 ภาพแสดงส่วนของซอร์ซโค้ดที่ใช้ในการทดลองเพื่อตรวจหา NUSAT (ชุดข้อมูลที่ 3)



รูปที่ 5.38 ภาพแสดงแผนภาพต้นไม้ของซอร์ซโค้ดจากการตรวจหา NUSAT (ชุดข้อมูลที่ 3)

5.3 การวิเคราะห์บริบท (Context Analyzing)

ในส่วนของการวิเคราะห์บริบทสำหรับการทดสอบความสามารถของเครื่องมือนั้นจะเป็นการทำงานที่กระทำควบคู่กับการสร้างแผนภาพต้นไม้ของซอร์ซโค้ด คือ เมื่อมีการตรวจหาโครงสร้างของร่องรอยที่ผิดพลาดในซอร์ซโค้ดนั้น การวิเคราะห์บริบทจะกำหนดส่วนในการค้นหาว่าจะกระทำกับส่วนใดของซอร์ซโค้ด ซึ่งโครงสร้างของภาษา PL/SQL ประกอบด้วย 3 บล็อก คือ Declaration Section , Execution Section และ Exception Section โดยเมื่อระบุประเภทของร่องรอยที่ผิดพลาดที่ต้องการตรวจหา ข้อกำหนดที่ได้จากการวิเคราะห์บริบทก็จะทำการตรวจสอบเงื่อนไขว่าต้องค้นหาในส่วนใดของซอร์ซโค้ด และจะมีข้อกำหนดเพิ่มเติมของแต่ละร่องรอยที่ผิดพลาด ดังที่ได้แสดงรวมอยู่ใน หัวข้อที่ 3.2

5.4 การรีแฟคทอริง (Refactoring)

ขั้นตอนการรีแฟคทอริงเพื่อทดสอบความสามารถของเครื่องมือในงานวิจัยนี้จะเป็นการรีแฟคทอริงด้วยตนเอง (Manually Refactoring) โดยจะทำตามขั้นตอนที่แสดงไว้ที่หัวข้อ 3.1 ในส่วนของแนวทางในการแก้ไข ซึ่งจะทำการรีแฟคทอริงกับซอร์ซโค้ดสำหรับทำการทดลองทั้ง 3 ชุด

5.5 การวัดค่าดัชนีความสามารถในการบำรุงรักษาหลังรีแฟคทอริง (Measuring Maintainability Index after Refactoring)

ขั้นตอนนี้จะทำการวัดค่าความสามารถในการบำรุงรักษาหลังรีแฟคทอริงของซอร์ซโค้ดสำหรับทำการทดลองทั้ง 3 ชุด โดยจะได้ค่าดังตารางที่ 5.3

ตารางที่ 5.3 ตารางแสดงค่าความสามารถในการบำรุงรักษาหลังการรีแฟคทอริง

Bad Smells	ข้อมูลชุดที่ 1	ข้อมูลชุดที่ 2	ข้อมูลชุดที่ 3
	MI(2)	MI(2)	MI(2)
BLES	100.08	84.54	90.74
UCL	126.96	121.31	129.57
ICE	97.54	109.97	87.23
UL	112.91	108.62	107.72
NRNFL	134.68	127.19	90.25
NUSAT	145.59	144.12	145.83

นิยาม:

MI(2) คือ ค่าความสามารถในการบำรุงรักษาหลังการรีแฟคทอริง

5.6 การวิเคราะห์และเปรียบเทียบค่าดัชนีความสามารถในการบำรุงรักษา (Comparison Analysis MI)

ขั้นตอนนี้จะเป็นการเปรียบเทียบค่าดัชนีความสามารถในการบำรุงรักษาก่อนและหลังการทำรีแฟคทอริงเพื่อประเมินความสามารถของเครื่องมือที่ได้พัฒนาขึ้นตามวิธีการที่ได้นำเสนอในงานวิจัยนี้ ซึ่งค่าที่เปลี่ยนแปลงของ MI(1) และ MI(2) มีคุณลักษณะบ่งบอกดังนี้

MI(1) > MI(2) is Positive
MI(1) = MI(2) is Unaltered
MI(1) < MI(2) is Negative

นิยาม:

Positive คือ มีการเปลี่ยนแปลงในทางที่ดี
Negative คือ มีการเปลี่ยนแปลงในทางที่ไม่ดี
Unaltered คือ ไม่มีการเปลี่ยนแปลง

5.7 ผลการทดลอง (Experimental Results)

ผลการทดลองสำหรับการทดสอบความสามารถของเครื่องมือที่ได้พัฒนาขึ้นตามวิธีการที่นำเสนอมีรายละเอียดดังตารางที่ 5.4 ซึ่งแสดงค่า MI ที่ได้จากการทดลองทั้งหมด

ตารางที่ 5.4 แสดงค่าเปรียบเทียบ MI(1) และ MI(2)

Bad Smells	ข้อมูลชุดที่ 1		ข้อมูลชุดที่ 2		ข้อมูลชุดที่ 3	
	MI(1)	MI(2)	MI(1)	MI(2)	MI(1)	MI(2)
BLES	100.46	100.08	87.35	84.54	90.79	90.74
UCL	131.77	126.96	126.53	121.31	133.49	129.57
ICE	100.28	97.54	110.23	109.97	90.64	87.23
UL	112.91	112.91	109.14	108.62	108.50	107.72
NRNFL	145.59	134.68	134.45	127.19	91.75	90.25
NUSAT	147.60	145.59	145.59	144.12	146.13	145.83

สำหรับตารางที่ 5.5 , 5.6 และ 5.7 เป็นตารางการเปรียบเทียบค่า MI(1) และ MI(2) ของแต่ละชุดข้อมูลและเพื่อแสดงให้เห็นการเปลี่ยนแปลงของค่า MI

ตารางที่ 5.5 แสดงการเปลี่ยนแปลงของค่า MI สำหรับชุดข้อมูลที่ 1

Bad Smells	ชุดข้อมูลที่ 1				
	MI(1)	MI(2)	MI(1) > MI(2)	MI(1) = MI(2)	MI(1) < MI(2)
BLES	100.46	100.08	●		
UCL	131.77	126.96	●		
ICE	100.28	97.54	●		
UL	112.91	112.91		●	
NRNFL	145.59	134.68	●		
NUSAT	147.60	145.59	●		

ตารางที่ 5.6 แสดงการเปลี่ยนแปลงของค่า MI สำหรับชุดข้อมูลที่ 2

Bad Smells	ชุดข้อมูลที่ 2				
	MI(1)	MI(2)	MI(1) > MI(2)	MI(1) = MI(2)	MI(1) < MI(2)
BLES	87.35	84.54	●		
UCL	126.53	121.31	●		
ICE	110.23	109.97	●		
UL	109.14	108.62	●		
NRNFL	134.45	127.19	●		
NUSAT	145.59	144.12	●		

ตารางที่ 5.7 แสดงการเปลี่ยนแปลงของค่า MI สำหรับชุดข้อมูลที่ 3

Bad Smells	ชุดข้อมูลที่ 3				
	MI(1)	MI(2)	MI(1) > MI(2)	MI(1) = MI(2)	MI(1) < MI(2)
BLES	90.79	90.74	●		
UCL	133.49	129.57	●		
ICE	90.64	87.23	●		
UL	108.50	107.72	●		
NRNFL	91.75	90.25	●		
NUSAT	146.13	145.83	●		

สำหรับตารางที่ 5.8 เป็นตารางแสดงค่าเฉลี่ยจากข้อมูลที่ใช้ในการทดสอบทั้ง 3 ชุดรวมกัน เพื่อให้เห็นข้อมูลเฉลี่ยของผลการทดลองทั้งหมด

ตารางที่ 5.8 แสดงการเปลี่ยนแปลงเฉลี่ยของค่า MI จากข้อมูลทั้ง 3 ชุด

Bad Smells	ชุดข้อมูลค่าเฉลี่ย				
	MI(1)	MI(2)	MI(1) > MI(2)	MI(1) = MI(2)	MI(1) < MI(2)
BLES	92.87	91.79	●		
UCL	130.60	125.95	●		
ICE	100.38	98.25	●		
UL	110.18	109.75	●		
NRNFL	123.93	117.37	●		
NUSAT	146.44	145.18	●		

บทที่ 6
บทสรุปและข้อเสนอแนะ

6.1 บทสรุป

งานวิจัยนี้นำเสนอวิธีการในการตรวจหาร่องรอยที่ผิดพลาดของโปรแกรมที่เก็บอยู่ในฐานข้อมูลด้วยการใช้แผนภาพต้นไม้และการวิเคราะห์บริบท สำหรับร่องรอยที่ผิดพลาด 6 ประเภท คือ Business Logic in Exception Sections , Using cursor loops with DML , Initialize and Cleanup Logic in Execution Section , Use Literals , No Range Values in Numeric FOR Loop และ Not Using SUBTYPES and Anchored Types โดยทำการสร้างแผนภาพต้นไม้ของร่องรอยที่ผิดพลาดทั้ง 6 ประเภท เพื่อเป็นต้นแบบสำหรับการตรวจหาร่องรอยที่ผิดพลาดในซอร์ซโค้ด และการตรวจหาร่องรอยที่ผิดพลาดจะต้องใช้ข้อกำหนดที่ได้จากการวิเคราะห์บริบทเพื่อช่วยในการตรวจหาร่องรอยที่ผิดพลาด จากวิธีการที่นำเสนอนี้ ได้นำมาพัฒนาเป็นเครื่องมือสำหรับตรวจหาร่องรอยที่ผิดพลาดที่พัฒนาด้วยภาษาวิชวลเบสิกดอทเน็ท เพื่อให้สามารถตรวจหาและระบุประเภทร่องรอยที่ผิดพลาดของซอร์ซโค้ด

ในการทดสอบความสามารถของเครื่องมือจะใช้ข้อมูลทดสอบ 3 ชุด สำหรับทดสอบร่องรอยที่ผิดพลาดทั้ง 6 ประเภท โดยหาค่าดัชนีความสามารถในการบำรุงรักษาก่อนและหลังรีแฟคทอริงของชุดข้อมูลทดสอบทั้งหมด

ผลจากการทดสอบความสามารถของเครื่องมือพบว่าสามารถตรวจหาร่องรอยที่ผิดพลาดประเภท Business Logic in Exception Sections , Using cursor loops with DML , Initialize and Cleanup Logic in Execution Section , Use Literals , No Range Values in Numeric FOR Loop และ Not Using SUBTYPES and Anchored Types ได้ และค่าดัชนีความสามารถในการบำรุงรักษาของการทดลองนั้นมีการเปลี่ยนแปลงในทางที่ดี

6.2 ข้อเสนอแนะ

1. วิธีการที่นำเสนอสำหรับตรวจหาร่องรอยที่ผิดพลาดในงานวิจัยนี้ใช้สำหรับภาษา PL/SQL เท่านั้น ซึ่งสามารถนำวิธีการที่นำเสนอนี้ไปประยุกต์ใช้กับภาษาอื่นๆ

2. ควรศึกษาและพัฒนาเพิ่มเติมเพื่อให้สามารถตรวจหาร่องรอยที่ผิดพลาดด้วยแผนภาพต้นไม้ โดยที่ไม่ต้องจำกัดขอบเขตเฉพาะที่เป็นฟังก์ชันหรือโพรซีเยอร์

3. ควรศึกษาเพิ่มเติมถึงวิธีการในการประเมินผลการทดลองด้วยวิธีประเมินผลทางตรง เช่น วิธีการ พรีซิชัน-รีคอล (Precision-Recall)

6.3 ข้อจำกัดของงานวิจัย

1. การตรวจหาร่องรอยที่ผิดพลาดโดยใช้แผนภาพต้นไม้ต้องได้โครงสร้างที่เป็นไปตามอัลกอริทึมที่นำเสนอเท่านั้น คือ ตั้งแต่โหนดรากถึงโหนดที่ไม่ใช่โหนดใบ จะต้องมีโครงสร้างเหมือนกัน ส่วนโหนดใบสามารถเป็นโครงสร้างที่เป็นสับเซทของต้นแบบได้เท่านั้น จึงสามารถตรวจหาร่องรอยที่ผิดพลาดได้

2. ซอร์ซโค้ดที่จะใช้สำหรับการตรวจหาร่องรอยที่ผิดพลาดต้องเป็นฟังก์ชันหรือโพรซีเยอร์เท่านั้น เนื่องจากการกำหนดโหนดแรกที่ระบุว่าเป็นฟังก์ชัน หรือ โพรซีเยอร์ โดยถ้าไม่พบบริบทนี้ ก็จะไม่สามารถใช้วิธีการของแผนภาพต้นไม้ในการตรวจหาร่องรอยที่ผิดพลาด

3. การประเมินผลการทดลองของงานวิจัยเป็นการเมินผลทางอ้อมโดยดูจากค่าดัชนีความสามารถในการบำรุงรักษาซึ่งยังไม่ใช่การประเมินผลทางตรง

6.4 ผลงานตีพิมพ์

ชื่อหัวข้องานวิจัย “ การตรวจหาร่องรอยที่ผิดพลาดของโปรแกรมที่เก็บอยู่ในฐานข้อมูล (Detection of Stored Procedure Bad Smells) ” ผลงานนี้ได้รับคัดเลือกให้ถูกตีพิมพ์ในงานประชุมทางวิชาการนานาชาติ “ The 2nd International Conference on Software Engineering (ICOSE 2017) ” ซึ่งได้จัดขึ้นที่เมืองปักกิ่ง สาธารณรัฐประชาชนจีน ระว่างวันที่ 25 - 27 มิถุนายน 2560

รายการอ้างอิง

[1] Y. Kataoka, T. Imai, H. Andou, and T. Fukaya, "A quantitative evaluation of maintainability enhancement by refactoring," in *International Conference on Software Maintenance, 2002. Proceedings.*, 2002, pp. 576-585.

[2] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts, "Refactoring: Improving the Design of Existing Code. Addision Wesley Professional," pp. 13-72, 1990.

[3] T. Peanlert and P. Muengchaisri, "Bad-Smell Detection For Refactoring Using Object-Oriented SoftwareMetrics," International Conference on Computer Science, Software Engineering, InformationTechnology, e-Business, and Applications ,CSITeA,Cairo, Eypt, December 2004.

[4] F. A. Fontana, E. Mariani, A. Mornioli, R. Sormani, and A. Tonello, "An Experience Report on Using Code Smells Detection Tools," in *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, 2011, pp. 450-457.

[5] D. C. Nascimento, C. E. Pires, and T. Massoni, "PL/SQL Advisor: a Static Analysis-based Tool to Suggest Improvements for Stored Procedures," in *9th Brazilian Symposium on Information Systems (SBSI'13), João Pessoa, Brazil*, 2013, pp. 343-354.

[6] M. Habringer, M. Moser, and J. Pichler, "Reverse Engineering PL/SQL Legacy Code: An Experience Report," in *2014 IEEE International Conference on Software Maintenance and Evolution*, 2014, pp. 553-556.

[7] (2008). *Knowledge Xpert for PLSQL*. Available: http://toadsoft.com/EditionRNs/KXPLSQL_10.1_ReleaseNotes.htm

[8] N. Yaowarattanaprasert, "Design and Development of an Approach for detecting Flaws in Software Design Model Using Graph Diagram and Tree Diagram," Master of Science Program in Software Engineer, Computer Engineer, Chulalongkorn University, 2013.

[9] B. Tan and L. Zeng, "A performance optimization based on stored procedure in RDBS project," in *2010 International Conference on Computer and Communication Technologies in Agriculture Engineering*, 2010, pp. 594-597.

[10] H.-S. Lee and K.-G. Doh, "Tree-pattern-based duplicate code detection," presented at the Proceedings of the ACM first international workshop on Data-intensive software management and mining, Hong Kong, China, 2009.

[11] G. Buehrer, B. W. Weide, and P. A. G. Sivilotti, "Using parse tree validation to prevent SQL injection attacks," presented at the Proceedings of the 5th international workshop on Software engineering and middleware, Lisbon, Portugal, 2005.

[12] Y. Ito, A. Hazeyama, Y. Morimoto, H. Kaminaga, S. Nakamura, and Y. Miyadera, "A Method for Detecting Bad Smells and ITS Application to Software Engineering Education," in *2014 IIAI 3rd International Conference on Advanced Applied Informatics*, 2014, pp. 670-675.

[13] F. Palomba, "Textual Analysis for Code Smell Detection," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, 2015, pp. 769-771.

[14] B. Walter and P. Martenka, "Looking for Patterns in Code Bad Smells Relations," in *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, 2011, pp. 465-466.

[15] M. Alt, iota, iota, #351, iota, H. S*, et al.*, "Automated procedure clustering for reverse engineering PL/SQL programs," presented at the Proceedings of the 31st Annual ACM Symposium on Applied Computing, Pisa, Italy, 2016.

ภาคผนวก

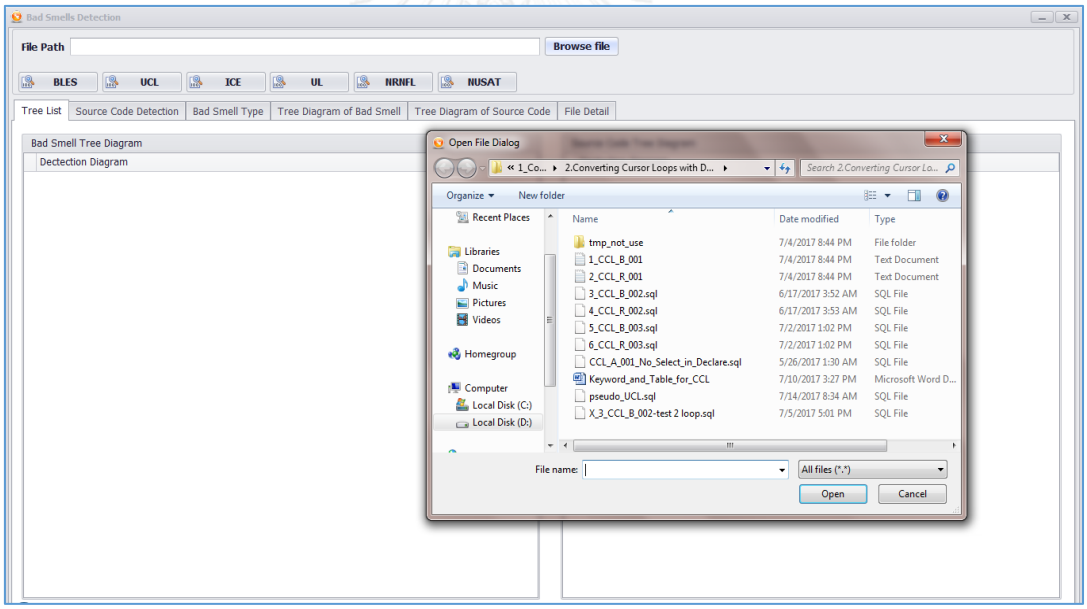
ภาคผนวก ก.

คู่มือการใช้งานเครื่องมือในการตรวจหาร่องรอยที่ผิดพลาด

ในภาคผนวกนี้จะอธิบายการใช้งานเครื่องมือในการตรวจหาร่องรอยที่ผิดพลาด ซึ่งจะแบ่งเป็น 3 ส่วนหลัก คือ การนำข้อมูลซอร์ซโค้ดเข้าระบบ การเลือกประเภทของร่องรอยที่ผิดพลาดที่ต้องการตรวจหา และการแสดงผลการตรวจหาร่องรอยที่ผิดพลาด

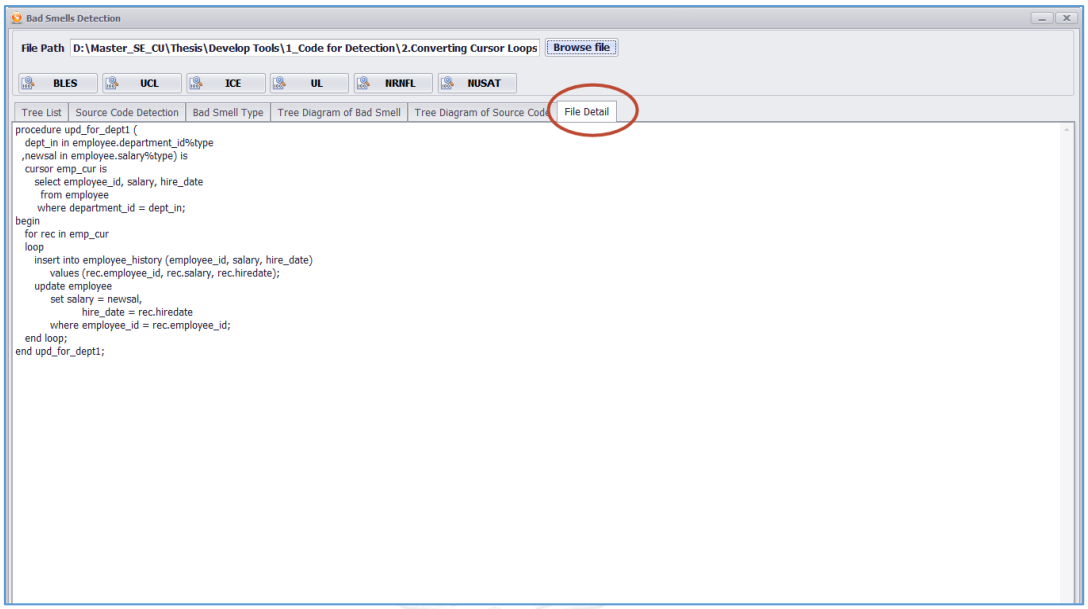
ก.1 การนำข้อมูลซอร์ซโค้ดเข้าระบบ

เมื่อผู้ใช้งานเปิดระบบขึ้นมาแล้ว ผู้ใช้สามารถเลือกไฟล์ข้อมูลซอร์ซโค้ดที่ต้องการตรวจหาร่องรอยที่ผิดพลาดด้วยการกดปุ่ม Browse file โดยสามารถเลือกได้ทีละไฟล์และจะต้องเป็นนามสกุล .txt หรือ .sql เท่านั้น ดังรูป ที่ ก.1



รูปที่ ก.1 ภาพแสดงหน้าจอการเลือกไฟล์เข้าสู่ระบบ

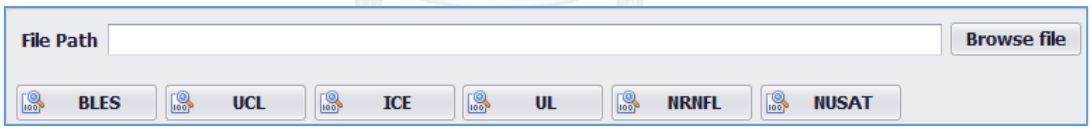
หลังจากที่ผู้ใช้งานเลือกไฟล์เข้าสู่ระบบแล้ว ระบบจะทำการแสดงโครงสร้างทั้งหมดของซอร์ซโค้ดในแท็บที่ชื่อ File Detail ดังแสดงในรูปที่ ก.2



รูปที่ ก.2 ภาพแสดงหน้าจอ File Detail

ก.2 การเลือกประเภทของร่องรอยที่ผิดพลาด

หลังจากเลือกไฟล์เข้าสู่ระบบแล้ว ก็ทำการเลือกร่องรอยที่ผิดพลาดที่ต้องการตรวจหา โดยจะมีรายละเอียดให้เลือกดังรูปที่ ก.3 โดยเมื่อมีการเลือกแล้ว ตัวอักษรบนปุ่มจะเปลี่ยนเป็นสีแดง

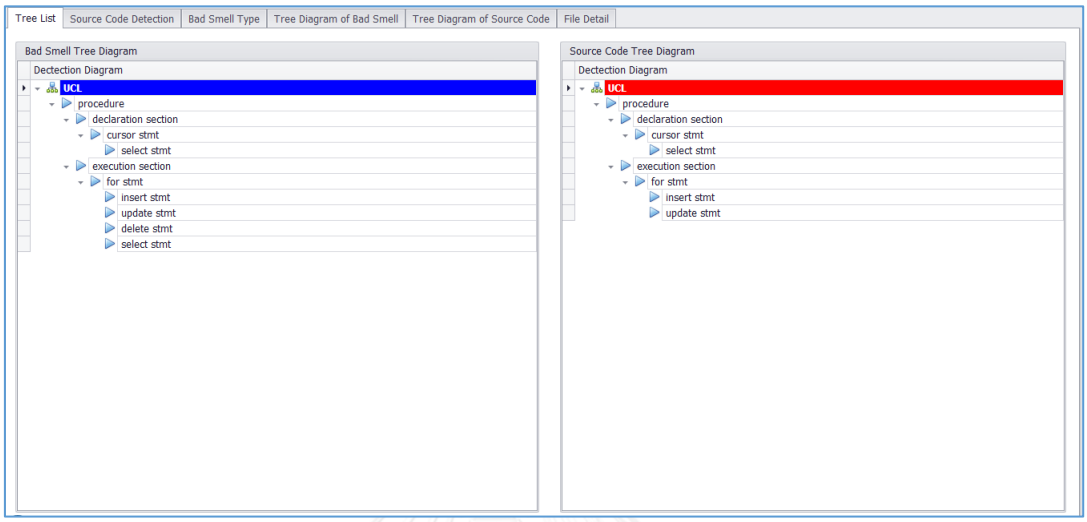


รูปที่ ก.3 ภาพแสดงหน้าจอตัวเลือกร่องรอยที่ผิดพลาด

ก.3 การแสดงผลการตรวจหาร่องรอยที่ผิดพลาด

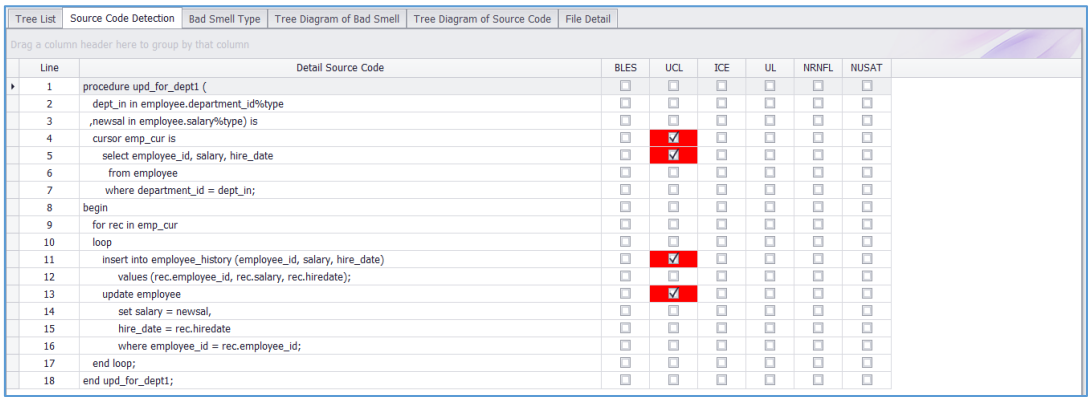
หลังจากเลือกประเภทของร่องรอยที่ผิดพลาดที่ต้องการตรวจหาแล้ว ก็จะได้ผลแสดงในแท็บของเครื่องมือดังนี้

1. แท็บ Tree List แสดงรายละเอียดของโครงสร้างแบบทรีลิสของร่องรอยที่ผิดพลาดต้นแบบกับ โครงสร้างแบบทรีลิสของซอร์ซโค้ด ดังรูปที่ ก.4 ซึ่งถ้ามีการตรวจหาร่องรอยที่ผิดพลาดไม่พบ ก็จะไม่แสดงในฝั่งของซอร์ซโค้ด



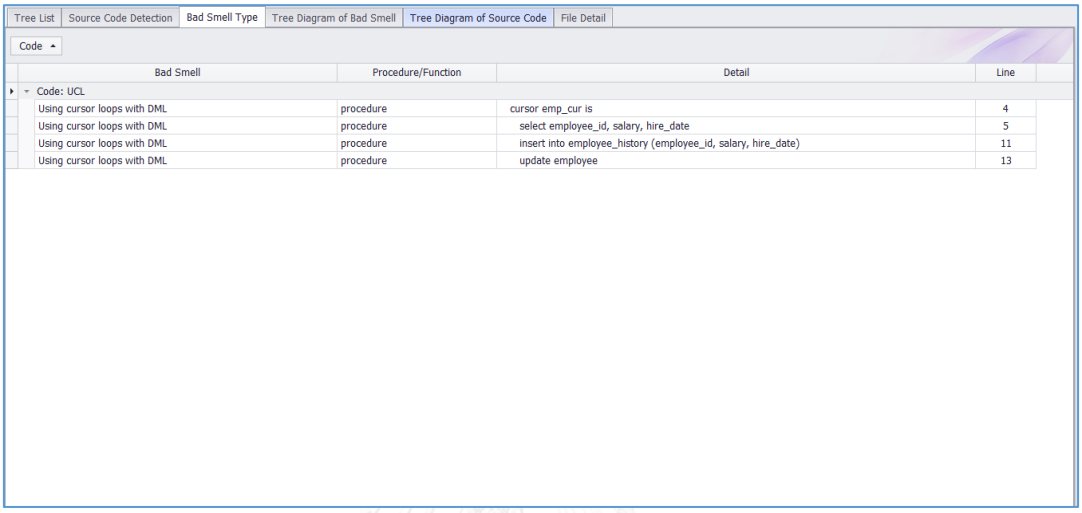
รูปที่ ก.4 ภาพแสดงหน้าจอทรีลิส

2. แท็บ Source Code Detection แสดงรายละเอียดของการตรวจหาร่องรอยที่ผิดพลาดของซอร์ซโค้ด โดยมีส่วนของโครงสร้างทั้งหมดของซอร์ซโค้ด รวมถึงการแสดงบรรทัดที่มีการตรวจพบร่องรอยที่ผิดพลาด ดังรูปที่ ก.5



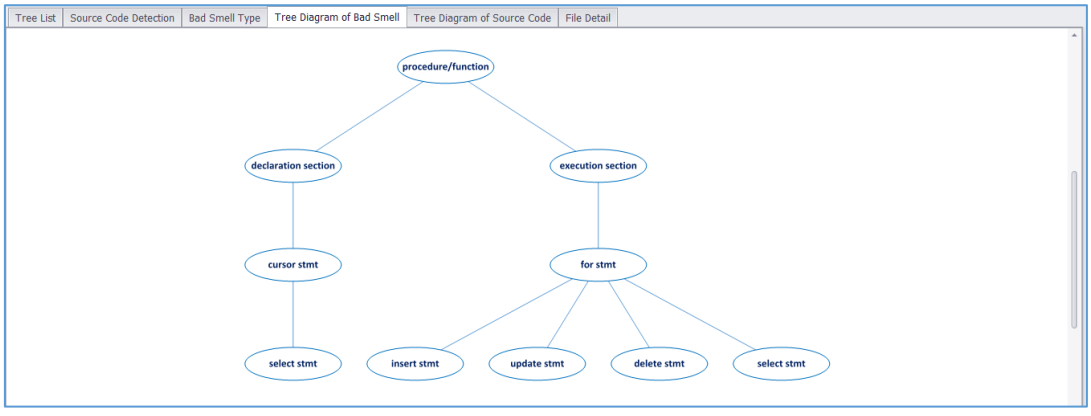
รูปที่ ก.5 ภาพแสดงหน้าจอการตรวจหาร่องรอยที่ผิดพลาด

3. แท็บ Bad Smell Type แสดงการระบุประเภทของร่องรอยที่ผิดพลาดที่เจอในซอร์ซโค้ดโดยจะมีการแสดงชื่อของร่องรอยที่ผิดพลาด แสดงว่าเป็นซอร์ซโค้ดประเภทโพรซีเยอร์หรือฟังก์ชัน แสดงรายละเอียดของซอร์ซโค้ดที่พบร่องรอยที่ผิดพลาด และแสดงบรรทัดที่ตรวจพบร่องรอยที่ผิดพลาด



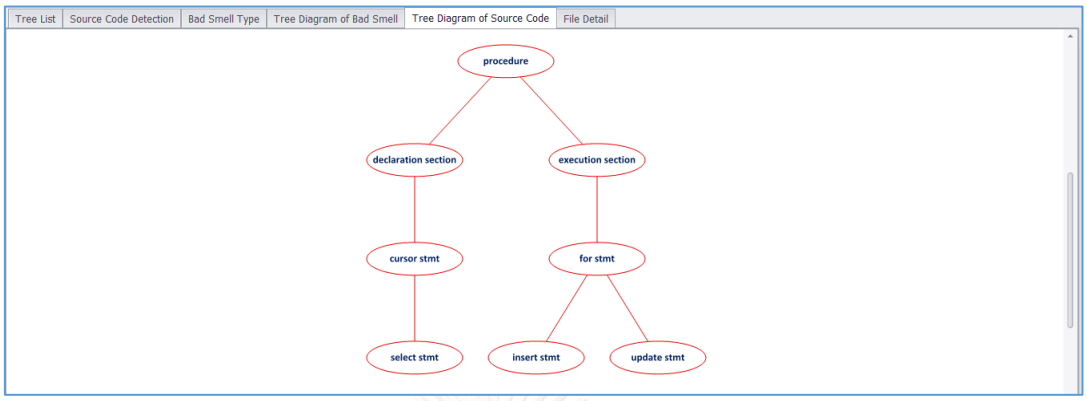
รูปที่ ก.6 ภาพแสดงหน้าจอผลการค้นหาร่องรอยที่ผิดพลาด

4. แท็บ Tree Diagram of Bad Smell คือ หน้าจอสำหรับแสดงโครงสร้างแผนภาพต้นไม้ของร่องรอยที่ผิดพลาดต้นแบบ



รูปที่ ก.7 ภาพแสดงหน้าจอของแผนภาพต้นไม้ของร่องรอยที่ผิดพลาด

5. แท็บ Tree Diagram of Source Code คือ หน้าจอสำหรับแสดงโครงสร้างแผนภาพต้นไม้ของซอร์ซโค้ด ซึ่งถ้าไม่พบร่องรอยที่ผิดพลาดก็จะไม่มีการแสดงแผนภาพต้นไม้



รูปที่ ก.8 ภาพแสดงแผนภาพต้นไม้ของซอร์ซโค้ด

ประวัติผู้เขียนวิทยานิพนธ์

นางสาวสุทธิกานต์ เนาวรัตน์ สำเร็จการศึกษาระดับปริญญาตรี หลักสูตรวิทยาศาสตรบัณฑิต สาขาคอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยกรุงเทพ ปีการศึกษา 2545 และเข้าศึกษาต่อในหลักสูตรวิทยาศาสตรมหาบัณฑิตสาขาวิศวกรรมซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2556