บทที่ 2

ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 ทฤษฎีที่เกี่ยวข้อง

ทฤษฎีที่เกี่ยวข้องในงานวิจัยนี้ ได้แก่ ร่องรอยที่ไม่ดี (Bad-Smell) ตรรกะเพรดิเคต (Predicate Logic) กฎการอนุมาน (Inference Rule) และ โปรล็อก (Prolog) ซึ่งมีรายละเอียดดังต่อไปนี้

2.1.1 ร่องรอยที่ไม่ดี

ร่องรอยที่ไม่ดี คือ ลักษณะของซอฟต์แวร์ที่ไม่ดีที่เกิดขึ้นจากขั้นตอนการออกแบบ หรือการเขียนโปรแกรม จึงเป็นผลให้ซอฟต์แวร์ทำงานได้ไม่มีประสิทธิภาพ [2] ซึ่งการค้นหาร่องรอยที่ไม่ดีนี้ถือว่าเป็นขั้นตอนที่สำคัญในวิธีการทำรีแฟคทอริง รีแฟคทอริงเป็นการเปลี่ยนแปลงโครงสร้างภายในซอฟต์แวร์แต่ไม่ส่งผลกระทบต่อพฤติกรรมของซอฟต์แวร์ หรือ พฤติกรรมของซอฟต์แวร์ไม่มีการเปลี่ยนแปลงไปจากระบบเดิม ร่องรอยที่ไม่ดีมีทั้งหมด 22 ประเภท แสดงในตารางที่ 2.1

ร่องรอยที่ไม่ดีทั้งหมดนี้สามารถแบ่งออกเป็นกลุ่มได้ 7 กลุ่มด้วยกัน [4] คือ

1. Bloaters คือ บางส่วนในโปรแกรม เช่น คลาส หรือ เมทธอดสามารถเพิ่มเติมให้มีขนาดใหญ่ได้ แต่ไม่สามารถจัดการได้อย่างมีประสิทธิภาพ
2. Object-Oriented Abusers คือ การใช้วิธีออกแบบเชิงวัตถุช่วยในการแก้ปัญหาแต่ใช้ไม่เต็มความสามารถ
3. Change Preventers คือ โครงสร้างของโปรแกรมที่สามารถแก้ไข หรือ เปลี่ยนแปลงได้ยาก หรือ มีการออกแบบที่ไม่ดีส่งผลให้การเปลี่ยนแปลง หรือ แก้ไขทำได้ยาก
4. Dispensables คือ บางส่วนของโปรแกรมที่ไม่ได้ใช้งานควรเอาออกจากโปรแกรม
5. Encapsulators คือ การลดร่องรอยที่ไม่ดีแบบหนึ่งทำให้เกิดร่องรอยที่ไม่ดีแบบอื่น
6. Couplers คือ ร่องรอยที่ไม่ดีที่มีการเข้าคู่กัน (Coupling) สูงแต่เป็นการขัดแย้ง
7. Others คือ ร่องรอยที่ไม่ดีที่ไม่ได้อยู่ในประเภทที่กล่าวมาแล้ว

ร่องรอยที่ไม่ดีทั้งหมดสามารถแบ่งเป็นกลุ่มได้ ดังแสดงในตารางที่ 2.2

ตารางที่ 2.1 ประเภทของร่องรอยที่ไม่ดี [9]

ลำดับ	ชื่อ	ความหมาย
1	Duplicate Code	โครงสร้างโค้ดที่เหมือนกันมากกว่า 1 ที่
2	Long Method	เมทธอดที่มีขนาดใหญ่
3	Large Class	คลาสที่มีฟังก์ชัน และอินสแตนท์จำนวนมาก
4	Long Parameter List	มีการผ่านพารามิเตอร์มาก ทำให้การทำความเข้าใจยากขึ้น
5	Divergent Change	คลาสหนึ่งรองรับการเปลี่ยนแปลงหลายๆ อย่างที่เกิดขึ้น
6	Shotgun Surgery	มีการเพิ่มฟังก์ชันหนึ่งต้องทำการแก้ไขในหลายๆคลาส ดังนั้น ความสัมพันธ์ระหว่างคลาส ควรเป็นหนึ่งต่อหนึ่ง
7	Feature Envy	เมื่อมีเมทธอดที่มีการเรียกใช้คุณสมบัติหรือเมทธอดในคลาสอื่นมากกว่าคลาสที่เป็นเจ้าของเมทธอด
8	Data Clumps	พบว่ามีการใช้กลุ่มข้อมูลชุดเดียวกันในหลายๆที่ภายในโค้ดซึ่งกลุ่มของข้อมูลประกอบด้วย คุณลักษณะภายในคลาส หรือพารามิเตอร์ในเมทธอด
9	Primitive Obsession	การใช้ข้อมูลชนิดตัวอักษรและวันที่ในโปรแกรมมีการสร้างเป็นคลาส เมื่อมีการใช้ข้อมูลเหล่านั้นจึงอาจทำให้คลาสมีขนาดใหญ่ขึ้น
10	Switch Statement	การมี Switch statement ปรากฏในโปรแกรมทำให้ไม่เป็นโปรแกรมเชิงวัตถุ
11	Parallel Inheritance Hierarchy	เมื่อมีการสร้างคลาสลูกคลาสหนึ่งจะทำให้เกิดเป็นคลาสลูกอีกคลาสหนึ่งด้วย
12	Lazy Class	คลาสที่ไม่มีฟังก์ชันการทำงานหลักๆ
13	Speculative Generality	มีการสร้างคลาสหรือเมทธอดทำหน้าที่เฉพาะกรณีพิเศษทำให้การทำความเข้าใจและการบำรุงรักษาซอฟต์แวร์ยากขึ้น เช่น มีการสร้างคลาสหรือเมทธอดขึ้นมาเป็นกรณีทดสอบ
14	Temporary Field	การพบอ็อบเจกต์ที่มีตัวแปรที่ใช้สำหรับเก็บค่าผลลัพธ์ชั่วคราวในโค้ด

ตารางที่ 2.1 ประเภทของร่องรอยที่ไม่ดี (ต่อ)

ลำดับ	ชื่อ	ความหมาย
15	Message Chains	การที่มีการเรียกใช้ข้อมูลของอ็อบเจกต์หนึ่งจากอีกอ็อบเจกต์หนึ่งซึ่งไม่ได้เรียกใช้จากอ็อบเจกต์นั้นโดยตรง
16	Middle Man	คลาสที่ทำหน้าที่เป็นตัวกลางการติดต่อระหว่างอ็อบเจกต์
17	Inappropriate Intimacy (General Form)	การที่พบว่ามีการพยายามเรียกใช้คุณลักษณะที่เป็นไพรเวท (Private) ระหว่างคลาส
18	Alternative Class with Difference Interfaces	การที่มีเมทธอดที่ทำหน้าที่เดียวกันแต่มีซิกเนเจอร์ (Signature) ต่างกัน
19	Incomplete Library Class	การที่มีคลาสไลบรารี่ (Library Class) ที่ไม่สมบูรณ์ทำให้การทำความเข้าใจคลาส เพื่อนำกลับมาใช้ใหม่ทำได้ยากขึ้น
20	Data Class	คลาสที่มีฟังชันหลักเป็นการกำหนดค่าให้กับข้อมูล และ ในคลาสมีเพียงเอคเซสเซอร์เมทธอด (Accessor Method) เท่านั้น
21	Refused Bequest	คุณสมบัติที่คลาสลูกไม่ต้องการใช้แต่ได้รับการสืบทอดคุณสมบัติจากคลาสแม่มาทั้งหมด
22	Comment	เมื่อมีการแก้ไขร่องรอยที่ไม่ดีแล้วควรเขียนคำอธิบายการทำงานในแต่ละส่วนเพื่อให้ง่ายต่อการทำความเข้าใจ

ตารางที่ 2.2 ร่องรอยที่ไม่ดีแบ่งตามกลุ่ม [4]

ลำดับ	ชื่อกลุ่ม	ร่องรอยที่ไม่ดี
1	Bloater	Long Method, Large Class, Primitive Obsession, Long Parameter List and Data Clumps
2	Object-Oriented Abusers	Switch Statements, Temporary Field, Refused Bequest, Alternative Class with Different Interfaces and Parallel Inheritance Hierarchies
3	Change Preventers	Divergent Change and Shotgun Surgery
4	Dispensables	Lazy Class, Data Class, Duplicate Code and Speculative Generality
5	Encapsulators	Message Chains and Middle Man
6	Couplers	Feature Envy and Inappropriate Intimacy
7	Others	Incomplete Library Class and Comments

2.1.2 ตรรกะเพรดิเคต

ตรรกะเพรดิเคตเป็นการแทนความรู้ (Knowledge Representation) วิธีหนึ่งโดยส่วนมากจะใช้ในเรื่องของปัญญาประดิษฐ์ [11] ตรรกะเพรดิเคตมีไวยากรณ์ (Syntax) และความหมาย (Semantic) ที่กำหนดขึ้นอย่างชัดเจน จะเรียกสูตรที่ถูกต้องตามหลักไวยากรณ์ว่า สูตรรูปดี (Well Form Formula) ตรรกะเพรดิเคตจะประกอบด้วยสัญลักษณ์พื้นฐาน คือ สัญลักษณ์แสดงตรรกะเพรดิเคต (Predicate Symbol) สัญลักษณ์แสดงตัวแปร (Variable Symbol) สัญลักษณ์แสดงฟังก์ชัน (Function Symbol) สัญลักษณ์แสดงค่าคงที่ (Constant Symbol) และ เครื่องหมายวงเล็บ ตัวอย่าง การแทนความรู้ของรายละเอียด และ ความสัมพันธ์ของคลาส A ด้วยตรรกะเพรดิเคต มีรายละเอียดดังรูปที่ 2.1 และ รูปที่ 2.2

```
1 package sample;
2
3 public class A {
4     private String str="";
5     public void ma(){
6
7     }
8 }
9
```

รูปที่ 2.1 ซอร์สโค้ดภาษาจาวาของคลาส A

```
jpackage(sample,1,1).
jclass (A,3,8).
jfield(str,String,4,4).
jmethod(ma,viod,null,5,7).
owns(sample,A).
owns(A,str).
owns(A,ma).
```

รูปที่ 2.2 การแทนความรู้ด้วยเพรดิเคตของคลาส A

จากรูปที่ 2.2 สามารถอธิบายได้ดังนี้ คือ

1. jpackage (sample,1,1). เป็นการบอกรายละเอียดเกี่ยวกับแพ็กเกจ โดยที่
   jpackage : แทน แพ็กเกจ ซึ่งเป็นสัญลักษณ์เพรดิเคต
   sample : แทน ชื่อของแพ็กแกจ ซึ่งเป็นอาร์กิวเมนต์ตัวที่หนึ่ง
   1 : แทน บรรทัดเริ่มต้น ซึ่งเป็นอาร์กิวเมนต์ตัวที่สอง
   1 : แทน บรรทัดสุดท้าย ซึ่งเป็นอาร์กิวเมนต์ตัวที่สาม
2. jclass (A,3,8). เป็นการบอกรายละเอียดเกี่ยวกับคลาส โดยที่
   jclass : แทน คลาส ซึ่งเป็นสัญลักษณ์เพรดิเคต
   A : แทน ชื่อของคลาส ซึ่งเป็นอาร์กิวเมนต์ตัวที่หนึ่ง
   3 : แทน บรรทัดเริ่มต้น ซึ่งเป็นอาร์กิวเมนต์ตัวที่สอง
   8 : แทน บรรทัดสุดท้าย ซึ่งเป็นอาร์กิวเมนต์ตัวที่สาม

3. jfield(str,String,4,4). เป็นการบอกรายละเอียดเกี่ยวกับคุณลักษณะโดยที่
   jfield : แทนคุณลักษณะ ซึ่งเป็นสัญลักษณ์เพรดิเคต
   str : แทน ชื่อคุณลักษณะ ซึ่งเป็นอาร์กิวเมนต์ตัวที่หนึ่ง
   String : แทน ชนิดของคุณลักษณะ ซึ่งเป็นอาร์กิวเมนต์ตัวที่สอง
   4 : แทน บรรทัดเริ่มต้น ซึ่งเป็นอาร์กิวเมนต์ตัวที่สาม
   4 : แทน บรรทัดสุดท้าย ซึ่งเป็นอาร์กิวเมนต์ตัวที่สี่
4. jmethod(viod,ma,null,5,7). เป็นการบอกรายละเอียดเกี่ยวกับเมทธอด โดยที่
   jmethod : แทน เมทธอด ซึ่งเป็นสัญลักษณ์เพรดิเคต
   void : แทน return type ซึ่งเป็นอาร์กิวเมนต์ตัวที่หนึ่ง
   ma : แทน ชื่อของเมทธอด ซึ่งเป็นอาร์กิวเมนต์ตัวที่สอง
   null : แทน พารามิเตอร์ ซึ่งเป็นอาร์กิวเมนต์ตัวที่สาม
   5 : แทน บรรทัดเริ่มต้น ซึ่งเป็นอาร์กิวเมนต์ตัวที่สี่
   7 : แทน บรรทัดสุดท้าย ซึ่งเป็นอาร์กิวเมนต์ตัวที่ห้า
5. owns(sample,A). เป็นการบอกว่า A อยู่ใน “sample” ซึ่งในที่นี้ “sample” คือ แพ็กเกจ และ A คือ คลาส
6. owns(A,str). เป็นการบอกว่า str อยู่ใน A ซึ่งในที่นี้ A คือ คลาส และ str คือ คุณลักษณะ
7. owns(A,ma). เป็นการบอกว่า ma อยู่ใน A ซึ่งในที่นี้ A คือ คลาส และ ma คือ เมทธอด

2.1.3 กฎการอนุมาน

เมื่อสังเกตสิ่งต่างๆ หรือ ความสัมพันธ์ของโดเมนที่สนใจ และ นำมาเขียนให้อยู่ในรูปของตรรกะเพรดิเคตได้แล้วสามารถมองได้ว่าสิ่งที่เขียนในรูปของตรรกะเพรดิเคต คือ ความรู้ที่ทราบในโดเมนนั้นๆ ข้อดีของการแทนความรู้ คือ สามารถอนุมาน (Inference) [11] เพื่อหาข้อเท็จจริงใหม่ๆ หรือ ผลสรุปที่แฝงอยู่ในความรู้นั้นได้ เรียกกฎที่ใช้อนุมาน เพื่อหาความรู้ใหม่ว่า กฎการอนุมาน ซึ่งจะทำหน้าที่สร้างสูตรใหม่จากหลายๆสูตร สูตรใหม่ที่เกิดขึ้นนี้ เรียกว่า ทฤษฎี (Theorem)

2.1.4 โปรล็อก

โปรล็อก [11] เป็นภาษาคอมพิวเตอร์ชนิดหนึ่งที่ใช้ในการแก้ปัญหาทางด้านสัญลักษณ์ โดยใช้พื้นฐานของตรรกะเพรดิเคต องค์ประกอบที่สำคัญของโปรล็อกมี 3 ส่วนด้วยกัน คือ ข้อเท็จจริงโปรล็อก (Prolog Facts) กฎโปรล็อก (Prolog Rule) และ ข้อคำถามโปรล็อก (Prolog Query)

กฎโปรล็อก หรือ ในตรรกะเพรดิเคตจะเรียกว่า อนุประโยค (Clause) จะมีสองส่วน คือ ส่วนหัว (Head) และ ส่วนลำตัว (Body) ซึ่งจะเขียนอยู่ในรูป H:- B1,B2,...,Bn โดยมีความหมายว่าถ้า B1,B2,...,Bn เป็นจริง แล้ว H จะเป็นจริงด้วย หรือ ถ้าจะมองในลักษณะของโปรแกรมจะได้ว่าถ้าต้องการแก้ปัญหา H ต้องแก้ปัญหาย่อย B1,B2,....,Bn ก่อน

2.2 งานวิจัยที่เกี่ยวข้อง

2.2.1 งานวิจัย "Bad-Smell Detection Using Object-Oriented Software Metrics"

งานวิจัยนี้ [9] ได้ออกแบบมาตรวัดซอฟต์แวร์เชิงวัตถุในการตรวจจับร่องรอยที่ไม่ดี และได้แนะนำวิธีในการทำรีแฟคทอริงที่ใช้กับร่องรอยที่ไม่ดี 6 ประเภท คือ Long method, Large Class, Long parameter list, Switch statement, Lazy class และ Feature envy โดยการอธิบาย และ การกำหนดนิยามของมาตรวัดสำหรับการตรวจจับ และ การหาตำแหน่งของร่องรอยที่ไม่ดีจะแบ่งออกเป็น 6 ขั้นตอน คือ

1. นิยาม คือ การอธิบายลักษณะของร่องรอยที่ไม่ดีแต่ละประเภท
2. แรงจูงใจ คือ รายละเอียดของผลกระทบของร่องรอยที่ไม่ดีต่อซอฟต์แวร์ และ อธิบายถึงการพบร่องรอยที่ไม่ดีนั้นได้อย่างไร
3. วิธีการวัด คือ วิธีการที่ใช้เพื่อค้นหาร่องรอยที่ไม่ดี
4. มาตรวัดร่องรอยที่ไม่ดี คือ การกำหนดมาตรวัดสำหรับตรวจจับร่องรอยที่ไม่ดี โดยมีการใช้มาตรวัดที่มีอยู่แล้ว และ มาตรวัดที่สร้างขึ้นมาใหม่
5. การกำหนดค่าของการพิจารณา คือ ช่วงของค่ามาตรวัดที่เป็นร่องรอยที่ไม่ดี
6. การประยุกต์รีแฟคทอริง คือ ข้อเสนอแนะวิธีการรีแฟคทอริงสำหรับการปรับปรุงซอฟต์แวร์ เพื่อกำจัดร่องรอยที่ไม่ดี

โดยสามารถสรุปมาตรวัด และ การประยุกต์วิธีการรีแฟคทอริงสำหรับร่องรอยที่ไม่ดีทั้ง 6 แบบที่ใช้ในงานวิจัยนี้ แสดงรายละเอียดในตารางที่ 2.3

ตารางที่ 2.3 แสดงมาตรวัดร่องรอยที่ไม่ดีและวิธีรีแฟคทอริง

ร่องรอยที่ไม่ดี	มาตรวัด	วิธีรีแฟคทอริง
Long method	NOS, NOP, NOT	extract method, replace method with method object, replace temp with query, introduce parameter object, preserve whole object, decompose conditional
Large Class	NIM,NIV,TCC	extract class, extract subclass, extract interface, duplicate observed data
Long parameter list	NOP	Replace parameters with method, preserve whole object
Switch statement	NOSS	Move method, replace type code with state/strategy, replace parameter with explicit method, introduce null object
Lazy class	NIM,NIV,DIT	Inline class, collapse hierarchy
Feature envy	NCDM,NCDA,NCRA	move method, extract method, move attribute

โดยทั้งนี้ ผู้วิจัยได้สร้างเครื่องมือในการตรวจจับร่องรอยที่ไม่ดีสำหรับภาษาจาวาด้วยมาตรวัดที่ได้ออกแบบในงานวิจัยนี้ และ นำเครื่องมือที่ได้มาประยุกต์ใช้กับโปรแกรมตัวอย่าง ผลการทดสอบผู้ทำการวิจัย สรุปได้ว่าหลังจากการตรวจจับร่องรอยที่ไม่ดี และ ประยุกต์ใช้วิธีรีแฟคทอริงที่ได้นำเสนอ ทำให้ร่องรอยที่ไม่ดีที่มีอยู่ในโปรแกรมถูกแก้ไข

2.2.2 **งานวิจัย** "Detecting Design Flaws via Metric in Object-Oriented System"

งานวิจัยนี้ [5] นำเสนอมาตรวัดที่ใช้สำหรับค้นหาร่องรอยที่ไม่ดีสำหรับซอฟต์แวร์เชิงวัตถุ โดยออกแบบวิธีการในการค้นหาร่องรอยที่ไม่ดีในซอร์สโค้ด 5 ขั้นตอนด้วยกัน คือ การกำหนดแรงจูงใจ (Motivation) การกำหนดวิธีการ (Strategy) การออกแบบมาตรวัด (Metrics) การคำนวณค่ามาตรวัด (Measurement) และ การตรวจสอบผลที่พบ (Finding) ซึ่ง 3 ขั้นตอนแรกจะเป็นการอธิบายนิยาม และ เทคนิคที่ใช้ในการค้นหา ส่วน 2 ขั้นตอนหลังเป็นการนำไปประยุกต์ใช้ โดยผู้วิจัยได้นำวิธีการเหล่านี้ไปประยุกต์ใช้กับการค้นหาร่องรอยที่ไม่ดี 2 ประเภท คือ Data Class และ God Class สามารถสรุปได้ดังนี้

Data Class เป็นคลาสที่มีหน้าที่หลักในการกำหนดค่าให้กับข้อมูล และ ภายในคลาสมีเพียงเอคเซสเซอร์เมทธอด (Accessors Method) ทำให้มีผลกระทบต่อการบำรุงรักษาซอฟต์แวร์ การทดสอบซอฟต์แวร์ และ การทำความเข้าใจซอฟต์แวร์ คือ ถ้ามีการเปลี่ยนแปลงใน Data Class จะทำให้กระทบกับคลาสอื่นที่เรียกใช้ข้อมูลใน Data Class นั้น ถ้ามีคลาสมาเรียกใช้ข้อมูลใน Data Class เหมือนกันจะเกิดซอร์สโค้ดที่ซ้ำซ้อนขึ้น ส่วนมาตรวัดที่ใช้ในการค้นหามีอยู่ 3 ประเภท ด้วยกันดังนี้

1. มาตรวัด Weight of a Class (WOC) คือ จำนวนของเมทธอดที่ไม่ใช่เอคเซสเซอร์เมทธอด (Accessors Method) ในคลาสหารด้วยจำนวนสมาชิกของอินเทอร์เฟส (Interface) ทั้งหมด โดยไม่รวมสมาชิกที่สืบทอดมา โดยมีข้อกำหนดค่าพิจารณาอยู่ที่ 0 - 0.33
2. มาตรวัด Number of Public Attribute (NOPA) คือ จำนวนคุณลักษณะที่ไม่สืบทอดมา แต่มีการประกาศในอินเทอร์เฟสของคลาส โดยมีข้อกำหนดค่าพิจารณาอยู่ที่ NOPA $\geq$ 5
3. มาตรวัด Number of Accessor Method (NOAM) คือ จำนวนเอคเซสเซอร์เมทธอดที่ไม่สืบทอดมาแต่มีการประกาศในอินเทอร์เฟสของคลาส โดยมีข้อกำหนดค่าพิจารณาอยู่ที่ NOAM $\geq$ 3

God Class เป็นคลาสที่ทำการควบคุมการทำงานหลักทั้งหมดในระบบอัจฉริยะ (Intelligence System) ทำให้ส่งผลกระทบต่อการนำกลับมาใช้ใหม่ และ การทำความเข้าใจ เพราะ God Class มีหลายฟังก์ชันการทำงาน และ มีความรับผิดชอบที่หลากหลาย มาตรวัดที่ใช้มีดังนี้

1. มาตรวัด Access of Foreign Data (AOFD) คือ จำนวนคลาสภายนอกที่ยอมให้คลาสนั้นเข้าถึงข้อมูล ทั้งโดยตรง และ ผ่านเอคเซสเซอร์เมทธอด โดยไม่รวมถึงอินเนอร์คลาส (Inner Class) และ คลาสที่สืบทอดคุณสมบัติ (Super Class) โดยมีข้อกำหนดค่าพิจารณาอยู่ที่ AOFD $\geq$ 3
2. มาตรวัด Weighted Method Count (WMC) คือ ความซับซ้อนของเมทธอดทั้งหมดภายในคลาส
3. มาตรวัด Tight Class Cohesion (TCC) คือ จำนวนความสัมพันธ์ของเมทธอดที่สื่อสารกันโดยตรง โดยมีข้อกำหนดค่าพิจารณาอยู่ที่ 0 – 0.33

จากนั้นได้ทำการสร้างเครื่องมือสำหรับค้นหาร่องรอยที่ไม่ดีในภาษา C++ โดยมี 2 ส่วน คือ TableGen สำหรับเก็บข้อมูลการออกแบบต่างๆ เช่น คลาส เมทธอด การเรียกใช้เมทธอด เป็น

ต้น และ การนำมาตรวัดมาคำนวณ โดยใช้เครื่องมือฐานข้อมูลเชิงสัมพันธ์ (Relational Database Engine)

2.2.3 **งานวิจัย** "An Automated Refactoring Approach to Design Pattern-Based Program Transformation in Java Programs"

งานวิจัยนี้ [7] ได้นำเสนอวิธีการในการทำรีแฟคทอริงแบบอัตโนมัติ สำหรับซอร์สโค้ดภาษาจาวา โดยทำตามแบบรูปการออกแบบ (Design Pattern) ขั้นตอนในการทำจะมีอยู่ 2 ส่วนด้วยกัน คือ ส่วนของการอนุมานเพื่อที่จะระบุเซตของแบบรูปการออกแบบที่จะนำเสนอ โดยใช้สมการความสัมพันธ์ และ ส่วนของการเปลี่ยนแปลงให้ไปอยู่ในรูปแบบของแบบรูปการออกแบบที่ได้เลือกไว้ในตอนต้น โดยทำตามวิธีการรีแฟคทอริง

ในส่วนของการอนุมาน เป็นการทำเพื่อที่จะหาความสัมพันธ์ของข้อมูลทั้งหมดที่จำเป็นต้องมีในการเขียนโปรแกรมเชิงวัตถุ เช่น คลาส ลำดับความสัมพันธ์ คุณลักษณะ และเมทธอดของคลาส การเรียกใช้เมทธอด เป็นต้น โดยโปรแกรม P สามารถแสดงได้ด้วย 8 พจน์ด้วยกัน คือ (C,I,M,R,L,V,Ø,n) โดย

C แทน Class

I แทน Interfaces

M แทน Method

R แทน ความสัมพันธ์ class, Interfaces และ Method

L : C U I $\rightarrow$ชื่อคลาส หรือ อินเทอร์เฟส

V : R$\rightarrow$VISIBILITY

Ø : M $\rightarrow$ C_0 x TEXT x C_r x C_{p1} x....x C_{pi} x....x C_{pn}

n : C X C X TEXT $\rightarrow$MULTIPLICITY

จากนั้นจะทำการอนุมานเพื่อหาความสัมพันธ์ที่ซ่อนอยู่เพิ่มเติม โดยกฎที่อนุมานขึ้นมาจะมี 3 ลักษณะด้วยกัน คือ กฎที่สามารถอนุมานได้โดยตรงจากรายละเอียดการเขียนโปรแกรม หรือการออกแบบเชิงวัตถุ กฎการอนุมานที่ได้จากการวิเคราะห์บางส่วน และ การอ้างอิงจากกฎที่ได้ในส่วนแรก และ กฎการอนุมานที่ได้จากเรื่องราวการเปลี่ยนแปลงที่ต้องมาจากโปรแกรม 2 เวอร์ชันขึ้นไป

จากนั้นก็จะใช้กฎที่ได้เพื่ออธิบายความสัมพันธ์ของโปรแกรม รวมทั้งแบบรูปการออกแบบเหล่านั้น เพื่อที่จะหาว่าโปรแกรมเหมาะสมที่จะใช้แบบรูปการออกแบบประเภทใด โดยใช้โปรล็อกช่วยในการจัดการเกี่ยวกับเรื่องนี้ จากนั้นขั้นตอนสุดท้ายปรับปรุงซอร์สโค้ดให้เป็นไปตามแบบรูปการออกแบบที่เลือกโดยใช้วิธีการรีแฟคทอริง

การทำรีแฟคทอริงในงานวิจัยนี้จะใช้วิธีรีแฟคทอริงตามที่ Cinneide and Nixon [6] ได้นำเสนอไว้ ในการทดลองผู้วิจัยได้ใช้ แบบรูปการออกแบบประเภท แอบสแทคท์แฟคทอรี (Abstract Factory Design Patterns) เป็นตัวอย่าง

ในงานวิจัยนี้ [7] ได้ใช้วิธีรีแฟคทอริง 15 วิธี ดังนั้นผู้วิจัยคิดว่าควรใช้วิธีรีแฟคทอริงให้มากกว่านี้เพื่อที่จะใช้ได้กับแบบรูปการออกแบบที่มากขึ้น