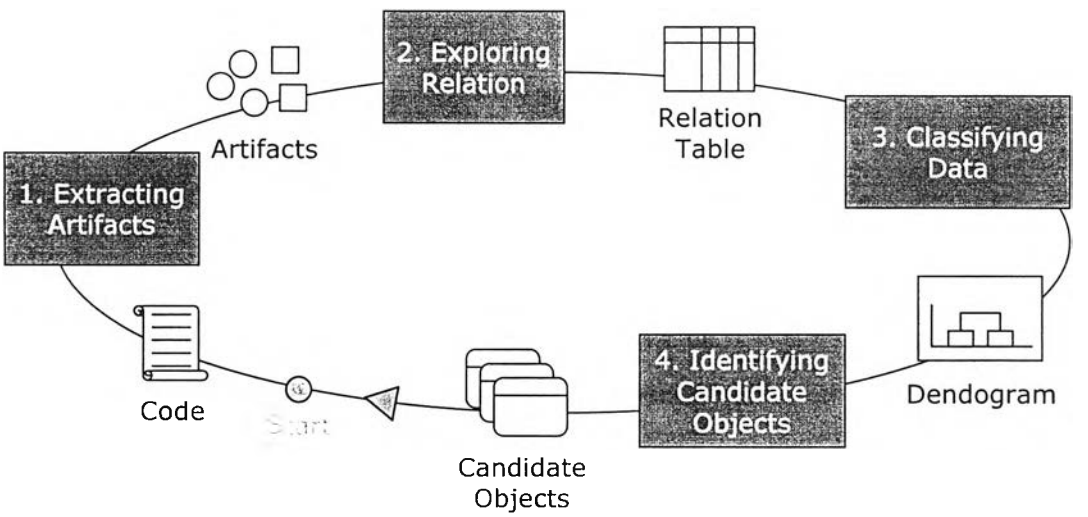
# บทที่ 3
# การออกแบบวิธีการระบุวัตถุซอฟต์แวร์ที่เป็นไปได้ซึ่งประยุกต์ใช้วิธีการจัดกลุ่มข้อมูลแบบลำดับชั้นและการให้ค่าน้ำหนัก

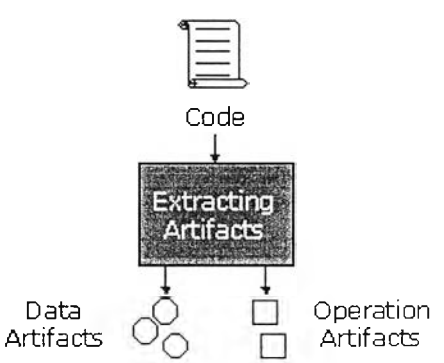
เนื้อหาในบทนี้เกี่ยวข้องกับการออกแบบวิธีการระบุวัตถุซอฟต์แวร์ที่เป็นไปได้ ซึ่งได้นำเอาวิธีการจัดกลุ่มข้อมูลแบบลำดับชั้นและการให้ค่าน้ำหนักมาประยุกต์ใช้ เพื่อช่วยให้การทำงานมีรายละเอียดถูกต้องมากขึ้น ขั้นตอนการทำงานของวิธีการดังกล่าวแบ่งออกเป็น 4 ส่วน ได้แก่ (1) การคัดแยกส่วนประกอบเดิม (Extracting Artifacts) (2) การค้นหาความสัมพันธ์ระหว่างส่วนประกอบเดิม (Exploring Relation) (3) การจัดเรียงส่วนประกอบเดิม (Classifying Artifacts) และ (4) การระบุวัตถุซอฟต์แวร์ที่เป็นไปได้ (Identifying Candidate Objects) ดังแสดงในรูปที่ 3.1




รูปที่ 3.1 ขั้นตอนการทำงานของวิธีการระบุวัตถุซอฟต์แวร์ฯ

## 3.1 การคัดแยกส่วนประกอบเดิม

การคัดแยกส่วนประกอบเดิม คือ การนำโปรแกรมต้นฉบับมาวิเคราะห์เพื่อคัดแยกส่วนประกอบเดิมที่ต้องการออกมา ดังแสดงในรูปที่ 3.2




รูปที่ 3.2 ขั้นตอนการคัดแยกส่วนประกอบเดิม

ส่วนประกอบเดิมที่ถูกคัดแยกแบ่งออกเป็น 2 ประเภท ได้แก่ ส่วนข้อมูลและส่วนคำสั่ง ซึ่งมีรายละเอียดดังต่อไปนี้

i. ส่วนข้อมูล คือ ส่วนประกอบที่มีหน้าที่เก็บข้อมูลในโปรแกรมเชิงโครงสร้าง อาจเทียบได้กับสเตทในโปรแกรมเชิงวัตถุ ส่วนข้อมูลในภาษาซีมี 2 แบบ ได้แก่

   c. สตรักท์ (Struct) เป็นส่วนข้อมูลเชิงผสม (Aggregated Type) ใช้สำหรับสร้างชนิดข้อมูลที่มีความหมายพิเศษ ตัวอย่างเช่น struct stack, struct queue ฯลฯ สตรักท์มีไวยากรณ์ดังแสดงในรูปที่ 3.3

```
structure-type-specifier:
        structure-type-definition
        structure-type-reference

structure-type-defintion:
        struct identifier_opt { field-list }

structure-type-reference:
        struct identifier

field-list:
        component-declaration
        field-list component-declaration
```

รูปที่ 3.3 ไวยากรณ์ของสตรักท์

   d. ตัวแปรโกลบอล (Global Variable) เป็นส่วนข้อมูลเชิงเดี่ยว (Single Type) ใช้สำหรับเก็บค่าที่มีความสำคัญหรือค่าร่วม (Shared Value) ระหว่างโมดูล ตัวแปรโกลบอลมีไวยากรณ์ดังแสดงในรูปที่ 3.4

```
declaration:
        declaration-specifiers initialized-declarator-list ;
```

รูปที่ 3.4 ไวยากรณ์ของตัวแปรโกลบอล

ii. ส่วนคำสั่ง คือ ส่วนประกอบที่มีหน้าที่ทำงานตามคำสั่งของผู้เขียนโปรแกรม ส่วนคำสั่งในภาษาซี คือ ฟังก์ชัน ซึ่งมีไวยากรณ์ดังแสดงในรูปที่ 3.5

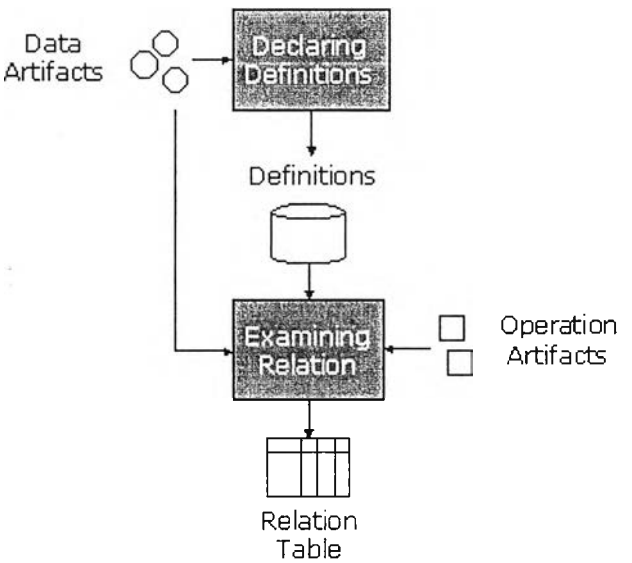
```
function-definition:
        declaration-specifiers_opt declarator declaration-list_opt compound-statement

compound-statement:
        { declaration-list_opt statement-list_opt }
```

รูปที่ 3.5 ไวยากรณ์ของฟังก์ชัน

## 3.2 การค้นหาความสัมพันธ์ระหว่างส่วนประกอบเดิม

การค้นหาความสัมพันธ์ระหว่างส่วนประกอบเดิม คือ การวิเคราะห์เพื่อค้นหาความเกี่ยวเนื่องระหว่างส่วนประกอบเดิมที่คัดแยกได้ การทำงานในขั้นตอนนี้ประกอบด้วยขั้นตอนย่อยสองขั้น (ดังแสดงในรูปที่ 3.6) ได้แก่ (1) การกำหนดนิยามความสัมพันธ์ และ (2) การตรวจสอบความสัมพันธ์ ซึ่งมีรายละเอียดดังต่อไปนี้




รูปที่ 3.6 ขั้นตอนการค้นหาความสัมพันธ์ระหว่างส่วนประกอบเดิม

i. การกำหนดนิยามความสัมพันธ์ (Declaring Definitions)

การกำหนดนิยามความสัมพันธ์ หมายถึง การนำนิยามความสัมพันธ์ทั่วไป (General Relation-ship Definition) ไปประยุกต์เข้ากับส่วนข้อมูล เพื่อสร้างเป็นนิยามความสัมพันธ์เฉพาะเจาะจง (Specific Relationship Definition) สำหรับใช้ในขั้นตอนการตรวจสอบความสัมพันธ์ นิยามความสัมพันธ์ทั่วไปที่กล่าวถึงนั้น หมายถึง ลักษณะความสัมพันธ์โดยทั่วไปที่ส่วนคำสั่งมีต่อส่วนข้อมูล แบ่งได้เป็น 3 แบบ ดังรายละเอียดต่อไปนี้

a. แบบใช้เป็นค่าส่งออก (Return-type Relationship) หมายถึง ความสัมพันธ์ซึ่งเกิดขึ้นจากการที่ 'ส่วนคำสั่ง' ใช้ 'ส่วนข้อมูล' เป็นค่าส่งออก ยกตัวอย่างเช่น ความสัมพันธ์ระหว่าง 'struct stack' กับ 'initStack' ในตารางที่ 3.1

ตารางที่ 3.1 ตัวอย่างความสัมพันธ์แบบใช้เป็นค่าส่งออก

| Data Artifact | Operation Artifact |
|---|---|
| **struct stack** {<br>int *base, *sp, size;<br>} | **struct stack*** initStack (int sz) |

b. แบบใช้เป็นค่าส่งเข้า (Argument Relationship) หมายถึง ความสัมพันธ์ซึ่งเกิดขึ้นจากการที่ 'ส่วนคำสั่ง' ใช้ 'ส่วนข้อมูล' เป็นค่าส่งเข้า ยกตัวอย่างเช่น ความสัมพันธ์ระหว่าง 'struct stack' กับ 'pop' ในตารางที่ 3.2

ตารางที่ 3.2 ตัวอย่างความสัมพันธ์แบบใช้เป็นค่าส่งเข้า

| Data Artifact | Operation Artifact |
|---|---|
| **struct stack** {<br>int *base, *sp, size;<br>} | int pop **(struct stack* s)** |

c. แบบเรียกใช้ (Use Relationship) หมายถึง ความสัมพันธ์ซึ่งเกิดขึ้นจากการที่ 'ส่วนคำสั่ง' เรียกใช้ 'ส่วนข้อมูล' ภายในตัวฟังก์ชัน ยกตัวอย่างเช่น ความสัมพันธ์ระหว่าง 'struct stack' กับ 'isEmptyStack' ในตารางที่ 3.3

ตารางที่ 3.3 ตัวอย่างความสัมพันธ์แบบเรียกใช้

| Data Artifact | Operation Artifact |
|---|---|
| struct stack {<br>int ***base, *sp,** size;<br>} | int isEmptyStack (struct stack* s) {<br>return **(s->sp == s->base);**<br>} |

เมื่อนำนิยามความสัมพันธ์ทั่วไปทั้งสามแบบมาประยุกต์เข้ากับส่วนข้อมูล จะได้เป็นนิยามความสัมพันธ์เฉพาะเจาะจงของส่วนข้อมูลนั้นๆ ดังตัวอย่างในตารางที่ 3.4

ตารางที่ 3.4 ตัวอย่างนิยามความสัมพันธ์เฉพาะเจาะจงของ struct stack

| Definition | Description |
|---|---|
| RSTK | Return struct stack |
| ASTK | Has struct stack argument |
| USTK | Use struct stack |

ii. การตรวจสอบความสัมพันธ์ (Examining Relation)

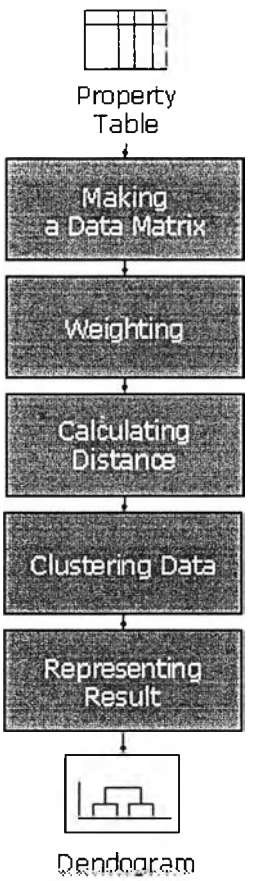
การตรวจสอบความสัมพันธ์ คือ การค้นหาความเกี่ยวเนื่องระหว่างส่วนคำสั่งกับส่วนข้อมูลคู่หนึ่งๆ ว่ามีลักษณะตรงกับนิยามความสัมพันธ์เฉพาะเจาะจงที่ตั้งไว้หรือไม่ ภายหลังการตรวจสอบ จะบันทึกค่าที่ได้ลงในตารางความสัมพันธ์ ดังตัวอย่างในตารางที่ 3.5

ตารางที่ 3.5 ตัวอย่างตารางความสัมพันธ์

| | RSTK | ASTK | USTK |
|---|---|---|---|
| **initStack** | T | | T |
| **isEmptyStack** | | T | T |
| **push** | | T | T |
| **pop** | | T | T |

### 3.3 การจัดเรียงส่วนประกอบเดิม

การจัดเรียงส่วนประกอบเดิม คือ การนำวิธีการจัดกลุ่มข้อมูลฯ และการให้ค่าน้ำหนักมาประยุกต์ใช้เพื่อช่วยจัดเรียงส่วนคำสั่งให้เป็นคลัสเตอร์ การทำงานในขั้นตอนนี้แบ่งออกเป็นขั้นตอนย่อย 5 ขั้น (ดังแสดงในรูปที่ 3.7) ได้แก่ (1) การสร้างเมตริกซ์ข้อมูล (2) การให้ค่าน้ำหนัก (3) การคำนวณค่าระยะทาง (4) การจัดเรียงข้อมูล และ (5) การแสดงผล ซึ่งมีรายละเอียดดังต่อไปนี้




รูปที่ 3.7 ขั้นตอนการจัดเรียงส่วนประกอบเดิม

1) การสร้างเมตริกซ์ข้อมูล (Making a Data Matrix)

การสร้างเมตริกซ์ข้อมูล คือ การเปลี่ยนรูปแบบข้อมูลในตารางความสัมพันธ์ซึ่งมีค่าเป็นแบบจริง-เท็จ ให้มาเป็นค่าเชิงปริมาณ ซึ่งทำได้โดยแทนค่า 'Y' ด้วยค่า '1.0' และแทนค่า 'N' ด้วยค่า '0.0' แล้วบันทึกผลที่ได้ลงในเมตริกซ์ข้อมูล ดังตัวอย่างในตารางที่ 3.6

ตารางที่ 3.6 ตัวอย่างเมตริกซ์ข้อมูล

| | | | |
|---|---|---|---|
| **initStack** | 1.0 | 0.0 | 1.0 |
| **isEmptyStack** | 0.0 | 1.0 | 1.0 |
| **push** | 0.0 | 1.0 | 1.0 |
| **pop** | 0.0 | 1.0 | 1.0 |

2) การให้ค่าน้ำหนัก (Weighting)

การให้ค่าน้ำหนัก คือ การกำหนดค่าให้กับความสัมพันธ์ที่เกี่ยวข้องกับส่วนข้อมูลชนิดใดชนิดหนึ่งเป็นพิเศษ ซึ่งทำได้ทั้งการให้ความสำคัญมากขึ้น (เพิ่มค่าน้ำหนัก) หรือให้ความสำคัญน้อยลง (หรือลดค่าน้ำหนัก) วิธีการนี้เลือกให้น้ำหนักกับเฉพาะความสัมพันธ์ที่เกิดจากส่วนข้อมูลสตรักท์ เนื่องจากเหตุผลสองประการคือ

a. ส่วนข้อมูลสตรักท์ เป็นส่วนข้อมูลที่มีความสำคัญต่อการระบุวัตถุซอฟต์แวร์ที่เป็นไปได้มากกว่าส่วนข้อมูลชนิดอื่นๆ เนื่องจากส่วนข้อมูลสตรักท์เป็นโครงสร้างข้อมูลที่มีความหมาย (ยกตัวอย่างเช่น struct student) ส่วนข้อมูลสตรักท์จึงเป็นตัวเลือกอันดับแรกๆ ของการเลือกไปสร้างเป็นวัตถุซอฟต์แวร์ที่เป็นไปได้[3]
b. การทดลองให้ค่าน้ำหนักแก่ความสัมพันธ์ทีละชนิด จะช่วยให้การศึกษาผลของค่าน้ำหนักที่มีต่อการทำงานทำได้ง่ายกว่าการทดลองให้ค่าน้ำหนักแก่ความสัมพันธ์หลายๆ ชนิดพร้อมกัน

การให้ค่าน้ำหนักสามารถทำได้โดยกำหนดค่าตัวแปร $w_k$ ในสูตรยูคลิเดียนชนิดให้ค่าน้ำหนักให้เท่ากับปริมาณที่ต้องการ เช่น กำหนดค่า $w_k$ ให้เท่ากับ 1.01 หากต้องการเพิ่มค่าน้ำหนัก 1%

3) การคำนวณค่าระยะทาง (Calculating Distance)

การคำนวณค่าระยะทาง คือ การนำคุณสมบัติด้านต่างๆ ของส่วนคำสั่งมาแทนลงในสูตรยูคลิเดียนชนิดให้ค่าน้ำหนัก เพื่อหาตัวเลขซึ่งบอกปริมาณความแตกต่างระหว่างส่วนคำสั่งคู่หนึ่งๆ ภายหลังการคำนวณ จะบันทึกผลที่ได้ลงในเมตริกซ์ความแตกต่าง ดังตัวอย่างในตารางที่ 3.7

ตารางที่ 3.7 ตัวอย่างเมตริกซ์ความแตกต่าง

| | initStack | isEmptyStack | push | pop |
|---|---|---|---|---|
| **initStack** | x | 1.41 | 0.00 | 0.00 |
| **isEmptyStack** | | x | 0.00 | 0.00 |
| **push** | | | x | 0.00 |
| **pop** | | | | x |

4) การจัดกลุ่มข้อมูล (Clustering Data)

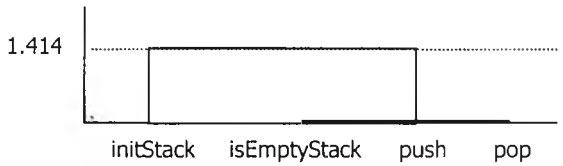
การจัดกลุ่มข้อมูล คือ การนำส่วนคำสั่งที่มีความแตกต่างกันน้อยที่สุดมาจัดให้อยู่ภายในคลัสเตอร์เดียวกัน โดยปฏิบัติตามขั้นตอนที่ระบุไว้ในอัลกอริทึมจัดเรียงข้อมูลแบบลำดับชั้นและกฎเลือกค่ามากสุด

5) การแสดงผล (Representing Result)

การแสดงผล คือ การนำลำดับการสร้างและรวมคลัสเตอร์ที่เกิดขึ้นระหว่างการทำงาน มาแสดงในเดนโดแกรม ดังตัวอย่างในรูปที่ 3.8

---

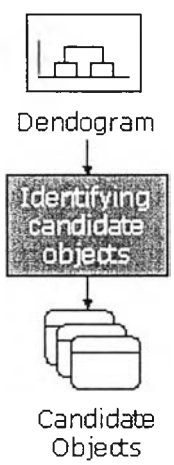
[3] ในบางโปรแกรม ส่วนข้อมูลสตรักท์อาจมีความสำคัญน้อยกว่าส่วนข้อมูลตัวแปรโกลบอลก็เป็นไปได้ ซึ่งถ้าเป็นเช่นนั้น ก็สามารถปรับการทำงานของวิธีการนี้ให้ดีขึ้นได้ด้วยการลดความสำคัญของส่วนข้อมูลสตรักท์ลงมา




รูปที่ 3.8 ตัวอย่างเดนโดแกรม

### 3.4 การระบุวัตถุซอฟต์แวร์ที่เป็นไปได้

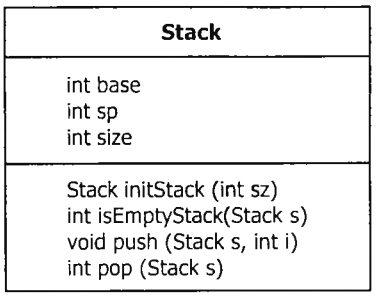
การระบุวัตถุวัตถุซอฟต์แวร์ที่เป็นไปได้ คือ การนำส่วนประกอบเดิมที่มีความสัมพันธ์กันมาประกอบเป็นโครงสร้างที่มีลักษณะคล้ายวัตถุซอฟต์แวร์ (ดังแสดงในรูปที่ 3.9) โดยนำส่วนข้อมูลมาแปลงเป็นสเตทและนำส่วนคำสั่งมาแปลงเป็นเมธอด




รูปที่ 3.9 ขั้นตอนการระบุวัตถุซอฟต์แวร์ที่เป็นไปได้

การระบุวัตถุซอฟต์แวร์ที่เป็นไปได้ไม่มีกฎแน่นอนตายตัว เพราะการระบุวัตถุซอฟต์แวร์ที่เป็นไปได้จัดเป็นการออกแบบ (Design) อย่างหนึ่งซึ่งไม่สามารถชี้ถูกชี้ผิดได้ อย่างไรก็ตาม การระบุวัตถุซอฟต์แวร์ที่เป็นไปได้มีเกณฑ์กว้างๆ ที่สามารถใช้เป็นแนวทางการทำงาน ซึ่งมีรายละเอียดดังนี้

1) พิจารณาผลการจัดเรียงส่วนคำสั่งที่แสดงไว้ในเดนโดแกรม แล้วเลือกเอาคลัสเตอร์ที่สามารถสื่อถึงโอเปอเรชันของสิ่งใดสิ่งหนึ่งได้ เช่น เลือกเอาคลัสเตอร์ที่ประกอบด้วย initStack, isEmptyStack, push และ pop เพราะสามารถสื่อถึงโอเปอเรชันของแถวลำดับได้
2) พิจารณาว่าส่วนคำสั่งภายในคลัสเตอร์ที่เลือกนั้น ส่วนใหญ่แล้วมีความสัมพันธ์กับส่วนข้อมูลใด เช่น จากการพิจารณาส่วนคำสั่งภายในคลัสเตอร์ที่เลือกในข้อหนึ่ง พบว่าส่วนใหญ่แล้วมีความสัมพันธ์กับส่วนข้อมูล struct stack
3) นำส่วนคำสั่งจากคลัสเตอร์ที่เลือกไว้ในข้อหนึ่ง มารวมกับส่วนข้อมูลที่เลือกไว้ในข้อสอง สร้างเป็นวัตถุซอฟต์แวร์ที่เป็นไปได้ ดังแสดงในรูปที่ 3.10




รูปที่ 3.10 ตัวอย่างวัตถุซอฟต์แวร์ที่เป็นไปได้