บทที่ 5
ขั้นตอนการทดลองใช้วิธีการระบุวัตถุซอฟต์แวร์ที่เป็นไปได้

เนื้อหาในบทนี้เกี่ยวข้องกับรายละเอียดของการทดลองใช้วิธีการระบุวัตถุซอฟต์แวร์ที่เป็นไปได้ โดยได้แบ่งการอธิบายออกเป็น 4 หัวข้อ ได้แก่ (1) โปรแกรมที่ใช้ทดลอง (2) สภาพแวดล้อมการทดลอง (3) ขั้นตอนการทดลอง และ (4) การวัดผลการทดลอง ซึ่งมีรายละเอียดดังต่อไปนี้

5.1 โปรแกรมที่ใช้ทดลอง

โปรแกรมที่นำมาใช้ในการทดลองได้แก่ โปรแกรมเชิงโครงสร้างภาษาซีที่มีลักษณะแตกต่างกัน 3 แบบ ซึ่งก็คือ (1) โปรแกรมแถวลำดับ (Stack) แถวคอย (Queue) แบบแยกส่วน (Modular Case) (2) โปรแกรมแถวลำดับแถวคอยแบบผูกติด (Tangled Case) และ (3) โปรแกรมลงทะเบียนพนักงาน ซึ่งมีรายละเอียดดังนี้

5.1.1 โปรแกรมแถวลำดับแถวคอยแบบแยกส่วน

โปรแกรมนี้มีที่มาจากงานของ Siff กับ Reps [7] จุดประสงค์ที่นำกลับใช้อีกครั้ง ก็เพราะต้องการเปรียบเทียบผลที่ได้จากการทดลองครั้งนี้กับผลที่ตีพิมพ์ในบทความ โปรแกรมมีส่วนประกอบทั้งหมด 10 ส่วน ซึ่งได้แก่ สตรักท์ 2 โครงสร้างและฟังก์ชัน 8 ฟังก์ชัน (ดังแสดงในรูปที่ 5.1) Siff กับ Reps เรียกโครงสร้างของโปรแกรมนี้ว่าเป็นแบบแยกส่วน เพราะว่าการทำงานของฟังก์ชันต่างๆ ถูกกำหนดขอบเขตการทำงานไว้ชัดเจน ซึ่งทำให้ฟังก์ชันไม่สามารถเรียกใช้ค่าจากส่วนข้อมูลที่ไม่เกี่ยวข้องโดยตรงได้

```
#define QUEUE_SIZE 10

struct stack {
  int *base, *sp, size;
};

struct queue {
  struct stack *front, *back;
};

struct stack* initStack(int sz) {
  struct stack* s = (struct stack*)
malloc(sizeof(struct stack));
  s->sp = (int*)malloc(sz*sizeof(int));
  s->base = s->sp;
  s->size = sz;
  return s;
}

struct queue* initQ() {
  struct queue* q = (struct queue*)
malloc(sizeof(struct queue));
  q->front = initStack(QUEUE_SIZE);
  q->back = initStack(QUEUE_SIZE);
  return q;
}

int isEmptyStack(struct stack* s) {
  return (s->sp == s->base);
}

int isEmptyQueue(struct queue* q) {
  return (isEmptyStack(q->front) &&
isEmptyStack(q->back));
}

void push(struct stack* s, int i) {
  /*no stack overflow check*/
  *(s->sp) = i;
  s->sp++;
}

void enQueue(struct queue* q, int i) {
  push(q->front, i);
}

int pop(struct stack* s) {
  if (isEmptyStack(s))
        return -1;
  s->sp--;
  return (*(s->sp));
}

int deQueue(struct queue* q) {
  if (isEmptyQueue(q))
        return -1;
  if (isEmptyStack(q->back))
        while (!isEmptyStack(q->front))
          push(q->back, pop(q->front));
  return pop(q->back); }
```

รูปที่ 5.1 โปรแกรมแถวลำดับแถวคอยแบบแยกส่วน

5.1.2 โปรแกรมแถวลำดับแถวคอยแบบผูกติด

โปรแกรมนี้มีที่มาจากงานของ Siff กับ Reps [7] เช่นเดียวกับโปรแกรมแรก โปรแกรมมีส่วนประกอบทั้งหมด 10 ส่วน ซึ่งได้แก่ สตรักท์ 2 โครงสร้างและฟังก์ชัน 8 ฟังก์ชัน ส่วนประกอบส่วนใหญ่ของโปรแกรมนี้มีรายละเอียดเหมือนกับส่วนประกอบของโปรแกรมแรก เว้นแต่ฟังก์ชัน isEmptyQueue กับฟังก์ชัน enQueue ที่ถูกดัดแปลงให้เรียกใช้ค่าจากแถวลำดับโดยตรง (แทนการเรียกโดยอ้อมเหมือนในโปรแกรมที่ผ่านมา) ดังแสดงในรูปที่ 5.2 ฟังก์ชันทั้งสองซึ่งเป็นฟังก์ชันของแถวคอยจึงมีความเกี่ยวเนื่องกับส่วนข้อมูลของแถวลำดับ Siff กับ Reps จึงเรียกโครงสร้างของโปรแกรมนี้ว่าเป็นแบบผูกติด

```
int isEmptyQueue(struct queue* q){
  return (q->front->sp==q->front->base) && (q-
>back->sp==q->back->base);
}
```

```
void enQueue(struct queue* q, int i){
  *(q->front->sp) = i;
  q->front->sp++;
}
```

รูปที่ 5.2 รายละเอียดของโปรแกรมแถวลำดับแถวคอยแบบผูกติดส่วนที่ถูกแก้ไข

5.1.3 โปรแกรมทะเบียนพนักงาน

โปรแกรมนี้มีที่มาจากเว็บไซต์ http://www.plantet-source-code.com จุดประสงค์ที่นำโปรแกรมนี้มาทดลอง ก็เพราะต้องการใช้เป็นตัวแทนของโปรแกรมที่มีส่วนข้อมูลมากกว่าหนึ่งแบบ กล่าวคือ มีทั้งส่วนข้อมูลชนิดสตรักท์ และส่วนข้อมูลชนิดตัวแปรโกลบอล โปรแกรมมีส่วนประกอบทั้งหมด 15 ส่วน ซึ่งได้แก่ สตรักท์ 2 โครงสร้าง ตัวแปรโกบอล 3 ตัว และฟังก์ชัน 10 ฟังก์ชัน ดังแสดงในรูปที่ 5.3

5.2 สภาพแวดล้อมการทดลอง

การทดลองใช้วิธีการระบุวัตถุซอฟต์แวร์ที่เป็นไปได้ฯ กระทำในเครื่องคอมพิวเตอร์ส่วนบุคคลที่มีหน่วยประมวลผล รุ่น Intel™ Celeron 333 หน่วยความจำขนาด 128 MB ใช้ระบบปฏิบัติการ รุ่น Microsoft Windows 98 SE และติดตั้งระบบปฏิบัติงานจาวา รุ่น JRE 1.3

5.3 ขั้นตอนการทดลอง

ขั้นตอนการทดลองใช้วิธีการระบุวัตถุซอฟต์แวร์ที่เป็นไปได้ฯ เริ่มต้นจาก (1) คัดแยกส่วนประกอบเดิมออกจากโปรแกรมต้นฉบับ (2) หาความสัมพันธ์ระหว่างส่วนประกอบเดิมที่คัดแยกได้ (3) จัดเรียงส่วนประกอบเดิมโดยใช้วิธีการจัดกลุ่มข้อมูลแบบลำดับชั้นร่วมกับการให้ค่าน้ำหนัก และ (4) วัดผลการทดลอง เมื่อทำการทดลองกับโปรแกรมที่กำหนดไว้จนครบทุกโปรแกรมแล้ว ให้เปลี่ยนไปนำวิธีการระบุวัตถุซอฟต์แวร์ที่เป็นไปได้แบบอื่น คือ วิธีการที่ประยุกต์จากวิธีการจัดกลุ่มข้อมูลฯ (แบบปกติ) และวิธีการที่ประยุกต์จากวิธีการวิเคราะห์คอนเซ็ปต์มาทดลองใช้ แล้วจึงนำผลลัพธ์ที่ได้จากวิธีการทั้งสามชนิดมาเปรียบเทียบกัน

```
#include <stdio.h>
#include <string.h>

int max_pos, cur_pos;

struct list {
  struct employee *data;
  struct list *next, *back;
};

struct list *head_ptr;

struct employee {
  char *name;
  int age, salary;
};

void add_pos(int i) {
  max_pos = max_pos + i;
}

int clear_pos(int i) {
  if (max_pos<=i)
        return -1;
  else
        return max_pos-i;
}

int check_pos() {
  int avai_pos = max_pos - cur_pos;

  if (avai_pos >0)
        return avai_pos;
  else
        return -1;
}

void init() {
  head_ptr = (struct list*) malloc(sizeof(struct
list));
  head_ptr->data = NULL;
  head_ptr->next = NULL;
  head_ptr->back = NULL;
}

void add(struct employee *e) {
  struct list *new_list;
  struct list *next_list;

  new_list = (struct list*) malloc(sizeof(struct list));
  new_list->data = e;
  new_list->next = head_ptr->next;
  new_list->back = head_ptr;

  head_ptr->next = new_list;
  next_list = new_list->next;
  if (next_list != NULL)
        next_list->back = new_list;
}

void del(struct list *del_list) {
  struct list *back_list;
  struct list *next_list;

  back_list = del_list->back;
  next_list = del_list->next;
  back_list->next = next_list;
  if (next_list != NULL)
        next_list->back = back_list;
}

int enroll(char *n, int a, int s){
  struct employee *new_emp;

  if (check_pos>0) {
        new_emp = (struct employee*)
malloc(sizeof(struct employee));
        new_emp->name = n;
        new_emp->age = a;
        new_emp->salary = s;
        add(new_emp);
        printf("add: %s %d %d\n", new_emp-
>name, new_emp->age, new_emp->salary);
        cur_pos++;
        return 1;
  } else {
        printf("Position is full");
              return -1;
  }
}

struct list* search(char* n) {
  struct list *cur_ptr = head_ptr->next;
  while (cur_ptr != NULL && strcmp(cur_ptr->data-
>name, n) !=0 )
        cur_ptr = cur_ptr->next;
        printf("found: %s \n", cur_ptr->data-
>name);
        return (cur_ptr);
}

int dismiss(char *n) {
  struct list *del_list= (struct list*) search(n);

  if (cur_pos>1) {
        if (del_list != NULL ) {
          del(del_list);
          cur_pos--;
          return 1;
        } else {
          printf("Warning: employee not found");
          return -1;
        }
  } else {
        printf("Warning: only one employee
left");
        return -1;
  }
}

void print() {
  struct list *cur_ptr = head_ptr->next;
  while (cur_ptr != NULL) {
        printf("print: %s %d %d\n", cur_ptr-
>data->name, cur_ptr->data->age, cur_ptr->data-
>salary);
        cur_ptr = cur_ptr->next;
  }
}
```

รูปที่ 5.3 โปรแกรมลงทะเบียนพนักงาน

5.4 การวัดผลการทดลอง

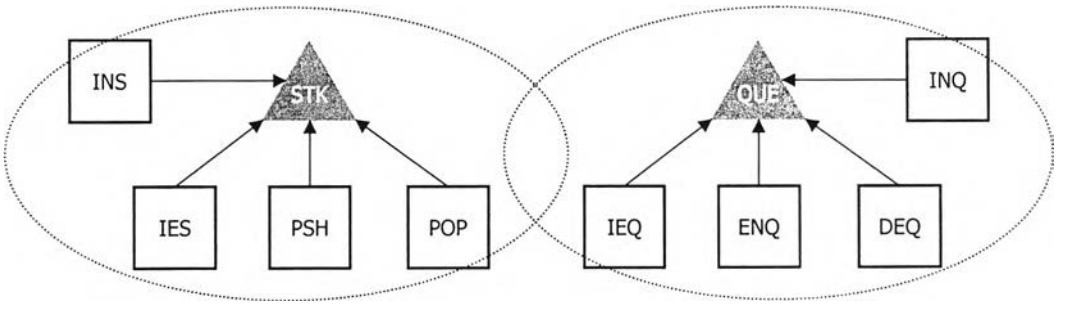
การวัดผลการทดลองสามารถทำได้โดยใช้วิธีการกำหนดกลุ่มส่วนคำสั่งที่ต้องการไว้ล่วงหน้า ก่อนที่จะนำวิธีการชนิดต่างๆ มาทดลองใช้ เมื่อตรวจผลการทดลองแล้วพบว่ามีกลุ่มส่วนคำสั่งดังกล่าวปรากฏอยู่ในผลลัพธ์ของการจัดกลุ่ม ให้ถือว่าวิธีการนั้นๆ ทำงานได้ผล แต่ถ้าไม่พบก็ให้ถือว่าวิธีการนั้นๆ ทำงานผิดพลาด

กลุ่มส่วนคำสั่งที่ถูกกำหนดไว้ล่วงหน้า หรือที่เรียกว่า กลุ่มสมบูรณ์ (Complete Module) หมายถึง กลุ่มส่วนคำสั่งที่มีความหมายสื่อถึงโอเปอเรชันของสิ่งใดสิ่งหนึ่งได้ ยกตัวอย่างเช่น กลุ่มส่วนคำสั่ง initStack, isEmptyStack, push และ pop เป็นกลุ่มสมบูรณ์ เพราะการรวมตัวกันของส่วนคำสั่งเหล่านี้สามารถสื่อถึงโอเปอเรชันของแถวลำดับได้ ในขณะที่กลุ่มส่วนคำสั่ง initStack, isEmptyStack, enQueuen, deQueue ไม่จัดว่าเป็นกลุ่มสมบูรณ์ เพราะการรวมตัวกันของส่วนคำสั่งเหล่านี้ไม่สามารถสื่อถึงโอเปอเรชันของสิ่งใดได้ ไม่ว่าจะเป็นโอเปอเรชันของแถวลำดับหรือว่าโอเปอเรชันของแถวคอย การกำหนดกลุ่มสมบูรณ์ให้กับโปรแกรมทั้งสามแบบ มีรายละเอียดและเหตุผลดังต่อไปนี้

5.4.1 กลุ่มสมบูรณ์ของโปรแกรมแถวลำดับแถวคอยแบบแยกส่วน

เมื่อพิจารณากราฟอ้างอิงในรูปที่ 5.4 จะเห็นได้ว่า ส่วนคำสั่งกระจายออกเป็นสองกลุ่มอย่างชัดเจน ซึ่งได้แก่ กลุ่มที่เกี่ยวข้องกับส่วนข้อมูล STK และกลุ่มที่เกี่ยวข้องกับส่วนข้อมูล QUE เมื่อนำส่วนคำสั่งทั้งสองกลุ่มมาพิจารณาความหมาย จะพบว่าการรวมตัวของส่วนคำสั่งที่เกี่ยวข้องกับส่วนข้อมูล STK สามารถสื่อได้ถึงโอเปอเรชันของแถวลำดับ และการรวมตัวของส่วนคำสั่งที่เกี่ยวข้องกับส่วนข้อมูล QUE สามารถสื่อได้ถึงโอเปอเรชันของแถวคอย

ด้วยเหตุนี้จึงกำหนดให้ กลุ่มสมบูรณ์ของโปรแกรมแถวลำดับแถวคอยแบบแยกส่วน คือ กลุ่มที่ประกอบด้วยส่วนคำสั่ง INS, IES, PSH กับ POP และกลุ่มที่ประกอบด้วยส่วนคำสั่ง INQ, IEQ, ENQ กับ DEQ



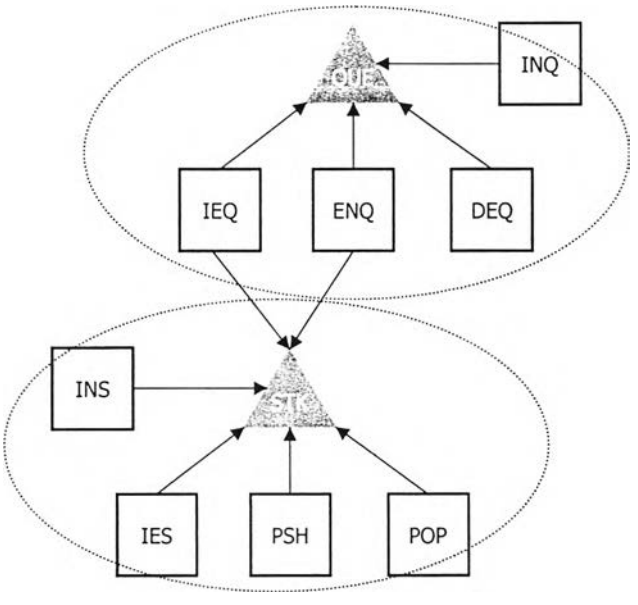
รูปที่ 5.4 กราฟอ้างอิงของโปรแกรมแถวลำดับแถวคอยแบบแยกส่วน

5.4.2 กลุ่มสมบูรณ์ของโปรแกรมแถวลำดับแถวคอยแบบผูกติด

เมื่อพิจารณากราฟอ้างอิงในรูปที่ 5.5 จะเห็นได้ว่า ส่วนคำสั่งกระจายออกเป็นสองกลุ่ม ซึ่งได้แก่ กลุ่มที่เกี่ยวข้องกับส่วนข้อมูล STK และกลุ่มที่เกี่ยวข้องกับส่วนข้อมูล QUE การกระจายตัวของส่วนคำสั่งในกรณีนี้ไม่ชัดเจนเท่ากรณีแรก เพราะมีเส้นเชื่อมจากส่วนคำสั่งแถวคอยข้ามไปสู่ส่วนข้อมูลแถวลำดับ อย่างไรก็ตาม เมื่อนำส่วนคำสั่งทั้งสองกลุ่มมาพิจารณาความหมาย จะพบว่าการรวมตัวของส่วนคำสั่งที่เกี่ยวข้องกับส่วนข้อมูล STK

(ยกเว้นส่วนคำสั่ง IEQ กับ ENQ) สามารถสื่อได้ถึงโอเปอเรชันของแถวลำดับ และการรวมตัวของส่วนคำสั่งที่เกี่ยวข้องกับส่วนข้อมูล QUE สามารถสื่อได้ถึงโอเปอเรชันของแถวคอย

ด้วยเหตุนี้จึงกำหนดให้ กลุ่มสมบูรณ์ของโปรแกรมแถวลำดับแถวคอยแบบแยกส่วน คือ กลุ่มที่ประกอบด้วยส่วนคำสั่ง INS, IES, PSH กับ POP และกลุ่มที่ประกอบด้วยส่วนคำสั่ง INQ, IEQ, ENQ กับ DEQ

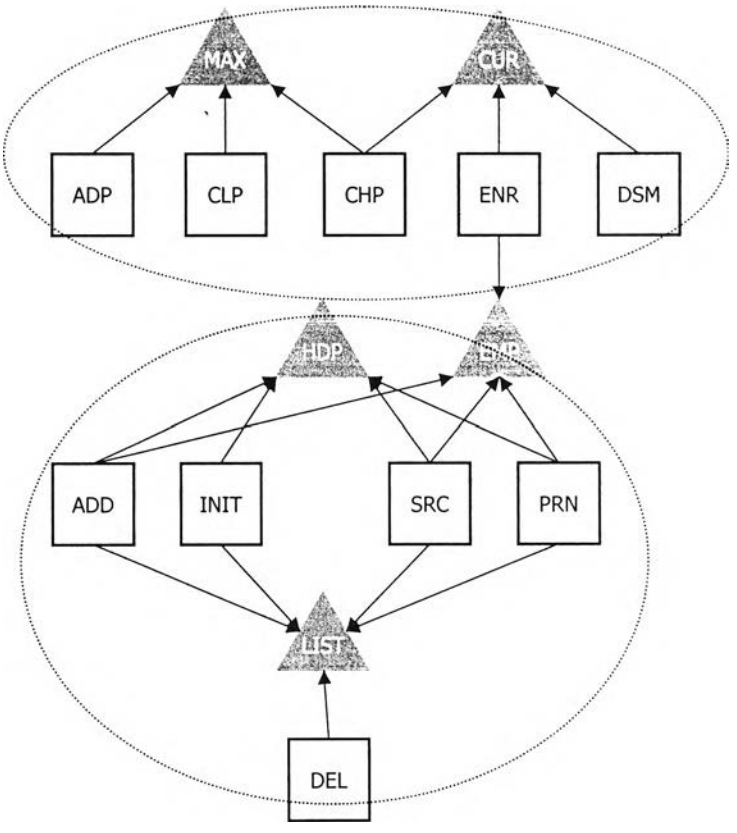


รูปที่ 5.5 กราฟอ้างอิงของโปรแกรมแถวลำดับแถวคอยแบบผูกติด

5.4.3 กลุ่มสมบูรณ์ของโปรแกรมลงทะเบียนพนักงาน

เมื่อพิจารณากราฟอ้างอิงในรูปที่ 5.6 จะเห็นได้ว่า โปรแกรมลงทะเบียนพนักงานมีความซับซ้อนมากขึ้น การกระจายตัวของส่วนคำสั่งไม่ชัดเจนเหมือนสองโปรแกรมที่ผ่านมา อย่างไรก็ตาม ยังคงพอมองเห็นในภาพรวมได้ว่า ส่วนคำสั่งกระจายตัวออกเป็นสองกลุ่มใหญ่ ซึ่งก็คือ กลุ่มที่เกี่ยวข้องกับส่วนข้อมูล MAX กับ CUR และกลุ่มที่เกี่ยวข้องกับส่วนข้อมูล HDP, EMP กับ LIST เมื่อนำส่วนคำสั่งทั้งสองกลุ่มมาพิจารณาความหมาย จะพบว่าการรวมตัวของส่วนคำสั่งที่เกี่ยวข้องกับส่วนข้อมูล MAX กับ CUR สามารถสื่อได้ถึงโอเปอเรชันการจัดการจำนวนพนักงาน ส่วนกลุ่มที่เกี่ยวข้องกับส่วนข้อมูล HDP, EMP กับ LIST สามารถสื่อได้ถึงโอเปอเรชันการจัดการข้อมูลพนักงานในลิสต์

ด้วยเหตุนี้จึงกำหนดให้ กลุ่มสมบูรณ์ของโปรแกรมลงทะเบียนพนักงาน คือ กลุ่มที่ประกอบด้วยส่วนคำสั่ง ADP, CLP, CHKP, ENR กับ DSM และกลุ่มที่ประกอบด้วยส่วนคำสั่ง ADD, INT, SRC, PRN กับ DEL



รูปที่ 5.6 กราฟอ้างอิงของโปรแกรมลงทะเบียนพนักงาน