โครงการตรวจวัดและวิเคราะห์คุณภาพของซอฟต์แวร์

Measure and Analyze Quality of Software Project

โดย

นายพรศักดิ์ แซ่ก๊วย

รหัส 39067008

รายงานนี้เป็นส่วนหนึ่งของวิชาโครงการพัฒนาระบบงาน

หลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ

ภาคเรียนที่ 2 ปีการศึกษา 2540

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ชื่อหัวข้อ	โครงการตรวจวัดและวิเคราะห์คุณภาพของซอฟต์แวร์
นักศึกษา	นายพรศักดิ์ แซ่ก๊วย
อาจารย์ที่ปรึกษา	ดร. เอื้อน ปิ่นเงิน
ระดับการศึกษา	วิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
แขนงวิชา	วิทยาการสารสนเทศ
พ.ศ.	2540

บทคัดย่อ

บทความประกอบโครงการพัฒนาระบบงานฉบับนี้ ได้อธิบายถึงแนวความคิด,ขั้นตอนการพัฒนาโครงการตลอดจนทฤษฎีที่ใช้อ้างอิงและผลลัพธ์ที่ได้จากโครงการตรวจวัดและวิเคราะห์ค่าความซับซ้อนของซอฟต์แวร์ โดยการพัฒนาระบบเพื่อใช้ในการตรวจวัดค่าความซับซ้อนของข้อมูลที่อยู่ในรูปของโปรแกรมภาษา EASYTRIEVE PLUS แล้วนำค่าพารามิเตอร์ที่ได้มาคำนวณหาค่าความซับซ้อน โดยใช้ทฤษฎีของแมคเคบ ซึ่งใช้วัดค่าความซับซ้อนของโปรแกรม โดยดูจากกราฟกระแสการควบคุม (CFG) ของโปรแกรมนั้นๆ ผลจากโครงการแสดงให้เห็นว่า ค่าความซับซ้อนของโปรแกรมที่ได้มีค่าค่อนข้างสูง แต่สามารถนำค่ากลางของค่าดังกล่าวมากำหนดให้เป็นค่ามาตราฐานของค่าความซับซ้อนของซอฟต์แวร์ ในการพัฒนาโปรแกรมขององค์กรต่อไปได้

Title	Measure and Analyze Quality of Software Project
Student	Mr. Ponsak Sea-Kooce
Advisor	DR. Ouen Pinngern
Level of Study	Master of Science in Information Technology
Major	Information Science
Year	1997

ABSTRACT

This article of System Development Project explains about the idea, steps in development project through referable theory, results from measuring project and analyzing complexity of software by system development. For measuring complexity of data in EASYTRIEVE PLUS programs. The parameters from measuring are calculated for complexity values by using McCabe's Theory which measures complexity of program by observing the Controlling Program Graph (CFG) in those programs. The results of this project presents those programs have rather high complexity value and the median can defines as standard value for developing programs of the organizations

สารบัญ

หน้า

สารบัญตาราง

หน้า

ตารางที่

สารบัญภาพ

หน้า

ภาพที่

บทที่ 1

บทนำ

1.1 ความเป็นมาของปัญหา

การพัฒนาระบบงานคอมพิวเตอร์ เป็นกระบวนการที่ยุ่งยากซับซ้อน มีขั้นตอนปลีกย่อยมากมาย อาจมีผู้ร่วมพัฒนาตั้งแต่หนึ่งคนจนถึงหลายร้อย หลายพันคน อาจใช้เวลาในการพัฒนาระบบตั้งแต่หนึ่งเดือนจนถึงเป็นสิบๆปีเลยทีเดียว ดังนั้นการควบคุมการจัดการกระบวนการที่ใช้ในการพัฒนาระบบ จึงเป็นเรื่องใหญ่และมีความสำคัญมาก ได้มีความพยายามในการพัฒนากระบวนการ วิธีการ เทคนิค ตลอดจนเครื่องมือต่างๆที่เข้ามาช่วยให้การพัฒนาระบบให้เป็นไปอย่างมีประสิทธิภาพ และบรรลุดังเป้าหมายที่ตั้งไว้ สามารถปฏิบัติงานได้ตามความต้องการของระบบ โดยส่วนใหญ่แล้วในปัจจุบันการพัฒนาระบบจะคำนึงถึงผลลัพธ์ของระบบที่ต้องสามารถปฏิบัติงานได้ตามวัตถุประสงค์ในการพัฒนาระบบ โดยลืมนึกถึงคุณสมบัติข้ออื่นๆของซอฟต์แวร์ไปเช่น ความง่ายต่อการทดสอบ ความง่ายต่อการดูแลรักษา ซึ่งในขั้นตอนการพัฒนาระบบงานคอมพิวเตอร์นั้นเราอาจต้องใช้เวลาในขั้นตอนของการทดสอบถึงครึ่งหนึ่งของเวลาที่ใช้ในการพัฒนาทั้งหมดและเราอาจต้องใช้งบประมาณในการปรับปรุง ดูแลรักษาระบบมากกว่างบประมาณที่ใช้ในการพัฒนาระบบในตอนแรกเสียอีก เนื่องจากซอฟต์แวร์มีคุณสมบัติพิเศษคือสามารถนำมาปรับปรุงพัฒนาขยายขีดความสามารถออกไปได้เรื่อยๆ โดยไม่จำเป็นต้องเริ่มพัฒนาใหม่ตั้งแต่ต้น ดังนั้นซอฟต์แวร์ที่มีการออกแบบให้สามารถทำความเข้าใจได้ง่าย, สามารถทำการปรับปรุงเปลี่ยนแปลงได้ง่ายจึงช่วยประหยัดทั้งเวลาและงบประมาณที่ใช้ในการพัฒนาระบบหรือขยายความสามารถของระบบได้เป็นอย่างมาก

ในปัจจุบันขั้นตอนในการทดสอบหรือตรวจสอบซอฟต์แวร์ ส่วนมากจะทำการทดสอบว่าซอฟต์แวร์สามารถทำงานได้ถูกต้องตามความต้องการที่กำหนดไว้หรือไม่ โดยการจำลองหรือเอาข้อมูลที่ใช้จริงมาผ่านเข้าไปในระบบแล้วตรวจสอบดูว่าผลลัพธ์ที่ได้ออกมานั้นตรงกับความต้องการหรือสอดคล้องกับข้อมูลที่นำเข้าไปหรือไม่ โดยไม่ได้มีการตรวจวัดหรือทดสอบคุณสมบัติอื่นๆของซอฟต์แวร์เลย อาจจะเนื่องจากว่าในปัจจุบันยังไม่มีเครื่องมือหรือวิธีการที่ใช้ตรวจวัดคุณสมบัติอื่นๆของซอฟต์แวร์มากนัก ดังนั้นจึงควรมีการพัฒนาเครื่องมือที่ใช้วัด

คุณสมบัติอื่นๆที่สำคัญของซอฟต์แวร์ โดยเฉพาะค่าความซับซ้อนของซอฟต์แวร์ซึ่งจะมีผลโดยตรงกับความง่ายในการทดสอบของซอฟต์แวร์ และความง่ายในการบำรุงรักษาซอฟต์แวร์ เพื่อให้สามารถระบุได้ว่าโปรแกรมใดๆ หรือโมดูลใดๆต้องใช้ความพยายามหรือระยะเวลาในการทดสอบมากกว่าโปรแกรมหรือโมดูลอื่นๆ หรือต้องใช้งบประมาณและความพยายามในการดูแลรักษาสูงกว่า โดยเราสามารถกำหนดเป็นค่ามาตราฐานการยอมรับได้ของค่าความซับซ้อนของการพัฒนาซอฟต์แวร์ภายในองค์กร ค่าความซับซ้อนของการพัฒนาซอฟต์แวร์ภายในองค์กรนั้นขึ้นอยู่กับสภาพแวดล้อมในการพัฒนา เครื่องมือที่ใช้และลักษณะงานของแต่ละองค์กรนั้นๆ โดยสามารถทำการตรวจวัดและทำการจัดเก็บผลลัพธ์ที่ได้เป็นระยะๆ เพื่อใช้ในการปรับปรุงระดับของค่าความซับซ้อนของซอฟต์แวร์ให้เหมาะสมกับแต่ละองค์กรได้ และยังสามารถนำเครื่องมือที่ได้จากการพัฒนามาใช้ในการประเมินขีดความสามารถของผู้พัฒนาระบบเอง เพื่อสามารถทำการบริหารและเพิ่มประสิทธิภาพให้แก่ผู้พัฒนาระบบได้อย่างถูกต้อง และยังสามารถนำมาใช้ในการทำนายระยะเวลาที่ใช้ในขั้นตอนการทดสอบ, และงบประมาณที่ต้องใช้ในการดูแลรักษาหรือปรับปรุงเปลี่ยนแปลงระบบได้อีกด้วย

1.2 ขั้นตอนการดำเนินโครงการ

1.2.1 ศึกษาทฤษฎีของมาตรวัดคุณภาพของซอฟต์แวร์แบบต่าง ๆ

1.2.2 ศึกษาโครงสร้างและการทำงานของ ภาษาที่ใช้ในการพัฒนาซอฟต์แวร์ที่จะทำการวัด

1.2.3 วิเคราะห์และเลือกชนิดของคุณภาพและแบบจำลองที่จะนำมาใช้ในการวัด

1.2.4 ออกแบบระบบ

1.2.5 พัฒนาโปรแกรม

1.2.6 ทำการวัดคุณภาพของซอฟต์แวร์ตามชนิดของคุณภาพและแบบจำลองที่ได้จาก 1.2.3

1.2.7 สรุปผลของโครงการ และงานที่ควรทำในอนาคต

1.3 ขอบเขตของโครงการ

1.3.1 พัฒนาโปรแกรมโดยใช้ภาษา C++ ในการพัฒนา

1.3.2. พัฒนาโปรแกรมบนสภาพแวดล้อมของระบบปฏิบัติการ Windows95

1.3.3 โปรแกรมที่จะนำมาทำการวัดต้องเขียนด้วยภาษา EASYTRIEVE PLUS

1.3.4 โปรแกรมที่จะนำมาทำการวัดต้องเป็นโปรแกรมที่มีความถูกต้องทางไวยากรณ์ของภาษา EASYTRIEVE PLUS แล้วเท่านั้น

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1.4.1 ได้ระบบที่ใช้ในการวัดคุณภาพของซอฟต์แวร์

1.4.2 สามารถกำหนดค่ามาตรฐานของ คุณภาพของซอฟต์แวร์

1.4.3 ได้เครื่องมือในการประเมินขีดความสามารถของผู้พัฒนาระบบ

1.4.4 ได้เครื่องมือในการทำนายและบริหารทรัพยากรที่ใช้ในการดูแลรักษาระบบซอฟต์แวร์

1.4.5 ได้เครื่องมือในการกะประมาณค่าใช้จ่ายในการดูแลรักษาระบบซอฟต์แวร์

1.4.6 ได้เครื่องมือในการติดตามควบคุมการดำเนินงานในการพัฒนาระบบซอฟต์แวร์

1.5 การเลือกกลุ่มตัวอย่างในโครงการ

จากการที่ภายในองค์กรที่ทำการทดลองมีการจัดโครงสร้างการบริหารแยกเป็นหน่วยงานย่อยๆประมาณ 20 หน่วยงาน โดยมีภาระและความรับผิดชอบที่แตกต่างกันไป และภายในองค์กรที่ทำการทดลองมีการใช้งานในภาษา EASYTRIEVE PLUS ในการพัฒนาโปรแกรมที่เป็นงาน Offline เป็นภาษาหลักในทุกๆหน่วยงาน โดยใช้ในการพัฒนาโปรแกรมเพื่อจัดพิมพ์รายงาน และจัดการกับแฟ้มข้อมูลเป็นส่วนใหญ่ ลักษณะโครงสร้างของภาษา EASYTRIEVE PLUS มีลักษณะที่คล้ายคลึงกับภาษา COBOL แต่จะมีโครงสร้างของภาษาที่ง่ายกว่า

ในการเลือกทำการสุ่มข้อมูลเพื่อนำมาทำการทดสอบได้ทำการจัดเก็บข้อมูลจาก 15 หน่วยงาน หน่วยงานละ 10 โปรแกรม โดยให้มีความยาวของโปรแกรมตั้งแต่ 200 บรรทัดขึ้นไป และให้เป็นโปรแกรมที่มีการใช้งานอยู่ในปัจจุบัน เพื่อให้ได้ชุดตัวอย่างที่สามารถใช้เป็นตัวแทนข้อมูลทั้งหมดได้ใกล้เคียงความเป็นจริง

1.6 การสรุปผลโครงการ

ภายหลังจากทำการทดสอบ และจัดเก็บข้อมูลจากชุดตัวอย่างทั้งหมดแล้วจะได้นำข้อมูลที่ได้มาทำการวิเคราะห์โดยใช้กระบวนการวิเคราะห์ทางสถิติ เพื่อหาค่าที่สามารถใช้เป็นตัวแทนค่ากลาง

ของข้อมูลได้ โดยจะดูจากลักษณะการกระจายของข้อมูล และค่าความน่าเชื่อถือของข้อมูล โดยใช้โปรแกรมทางสถิติช่วยในการวิเคราะห์ผล

บทที่ 2

ทฤษฎีการวัดค่าความซับซ้อนของซอฟต์แวร์

วิธีการวัดความซับซ้อนของซอฟต์แวร์หรือตัววัดซอฟต์แวร์ (Software Metrics) คือวิธีการมาตรฐานที่ใช้ในการวัดคุณสมบัติบางอย่างของซอฟต์แวร์ เช่น ขนาด ความยากง่าย ของซอฟต์แวร์

ความซับซ้อนของซอฟต์แวร์คือความยากต่อการเข้าใจ การแก้ไขเปลี่ยนแปลง และการทดสอบซอฟต์แวร์

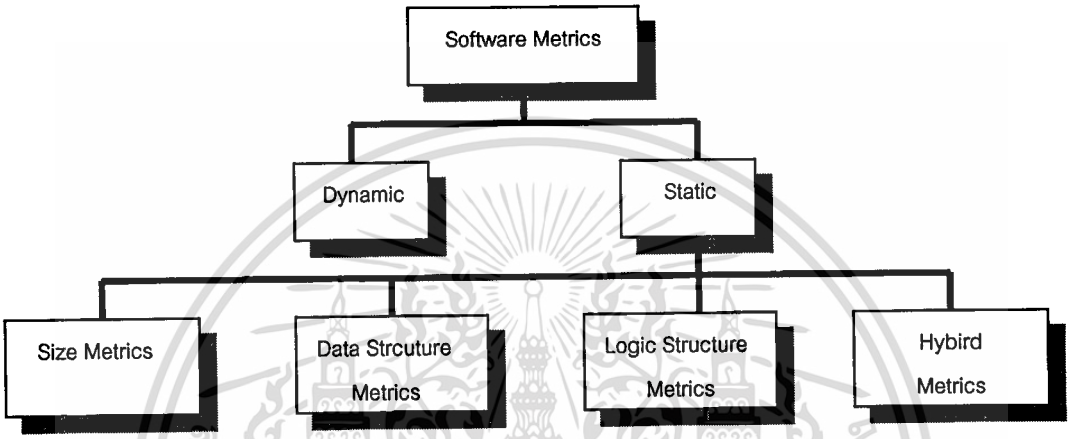
2.1 ชนิดของตัววัดซอฟต์แวร์

ตัววัดซอฟต์แวร์แบ่งออกได้เป็น 2 รูปแบบคือ

1. การวัดแบบสแตติก (Static) คือการวัดค่าความวับซ้อนจากตัวโปรแกรม สามารถแบ่งตามลักษณะได้ 4 แบบ คือ
   - 1.1 ตัววัดขนาด (Size Metrics) เป็นทฤษฎีการวัดที่ถูกสร้างขึ้นเป็นตัวแรก เป็นตัววัดที่เข้าใจง่ายและนิยมใช้มากที่สุดส่วนใหญ่เป็นการประมาณขนาดของโปรแกรมโดยการนับคำตัววัดที่จัดอยู่ในประเภทนี้ได้แก่การวัดปริมาตร ความยาวของโปรแกรมของฮอลสตีด
   - 1.2 ตัววัดโครงสร้างข้อมูล (Data Structure Metrics) เป็นวิธีที่คำนวณค่าความซับซ้อนโดยการพิจารณาจาก ปริมาณ โครงสร้าง การสื่อสาร และการใช้ข้อมูลร่วมกันระหว่างโมดูล การจัดการและการกระจายของข้อมูล ค่าคงที่ และตัวแปรต่างๆ ภายในโปรแกรม
   - 1.3 ตัววัดโครงสร้างตรรกะ (Logical Structure Metrics) เป็นการวัดความซับซ้อนจากโปรแกรมกราฟ หรือที่เรียกอีกชื่อหนึ่งว่า กราฟกระแสควบคุม (Control Flow Graph : CGF) ตัววัดแบบนี้ได้แก่ ตัววัดเชน ตัววัดแมคคูล ตัววัดไซโคลเมติกของแมคเคบ

1.4 ตัววัดแบบผสม (Hybird Metrics) เป็นตัววัดที่หาค่าความซับซ้อนจากกระแสข้อมูล (data flow) และกระแสการควบคุม (control flow) ตัววัดประเภทนี้ได้แก่ ตัววัดโอวิโด (Oviedo's Metrics)

2. การวัดแบบไดนามิก (Dynamic) คือการวัดค่าความซับซ้อนในขณะที่โปรแกรมทำงานอยู่



ภาพที่ 2.1 การแบ่งประเภทของตัววัดซอฟต์แวร์

วิธีวัดความซับซ้อนของโปรแกรม

1. Line of Code

เป็นการนับจำนวนบรรทัดของโปรแกรม ซึ่งเชื่อกันว่าโปรแกรมที่ยาวกว่าน่าจะมีความซับซ้อนมากกว่า มีการกำหนดการนับที่แตกต่างกันไป

- นับแต่ประโยคที่มี การ execute เท่านั้น
- นับทั้งประโยคที่มีการ execute และประโยคที่เป็นการกำหนดข้อมูล
- หรืออาจนับทุกประโยคที่อยู่ในตัวโปรแกรม รวมถึงประโยคที่อธิบายการทำงานของโปรแกรม

2. Halstead 's Metrics

Halstead ถือเป็นผู้ให้กำเนิดวิทยาการวัดความซับซ้อนของซอฟต์แวร์ โดยพิจารณาจากตัวดำเนินการ (operators) และตัวถูกดำเนินการ (operands) ที่มีอยู่ในโปรแกรม วิธีของ Halstead มีพารามิเตอร์ที่เกี่ยวข้องดังนี้

n_1 : จำนวนตัวดำเนินการทั้งหมดที่ไม่ซ้ำกันในโปรแกรม ได้แก่ ชื่อของฟังก์ชัน (function name) และพวกอักขระคั่น (delimiter) ต่าง ๆ รวมถึงคำหลักต่าง ๆ นักวิจัยบางท่านจะถือเป็นตัวดำเนินการแบบหนึ่งด้วย

n_2 : จำนวนตัวถูกดำเนินการทั้งหมดที่ไม่ซ้ำกันในโปรแกรม หมายถึง ตัวแปร และค่าคงที่ที่ใช้ในโปรแกรม

N_1 : จำนวนตัวดำเนินการทั้งหมดในโปรแกรม

N_2 : จำนวนตัวถูกดำเนินการทั้งหมดในโปรแกรม

จากพารามิเตอร์ทั้ง 4 สามารถนำไปใช้ในการวัดความซับซ้อนของโปรแกรม โดยพิจารณาจากสิ่งต่อไปนี้

2.1 ความยาวของโปรแกรม (Program Length)

Halstead เชื่อว่าโปรแกรมใดที่ยาวกว่าย่อมมีแนวโน้มที่จะมีความซับซ้อนมากกว่าโปรแกรมที่สั้นกว่า และความยาวนี้เป็นการนับ statement ของโปรแกรมที่มีการ execute โดยให้ N แทนความยาวของโปรแกรม

$$N = N_1 + N_2$$

หรือประมาณค่าได้จากสูตร

$$N = n_1 \times \log_2 n_1 + n_2 \times \log_2 n_2$$

2.2 ปริมาตรของโปรแกรม (Program Volume)

ปริมาตรของโปรแกรม หมายถึง จำนวนบิตที่ต้องการ encode โปรแกรมนั้น ๆ เช่น ถ้าโปรแกรมมีความยาว N = 4 และต้องการ encode โปรแกรมนั้น 2 บิต

ดังนั้นจะต้อง encode 2 บิตต่อ 1 operator หรือ 1 operand

จะได้ปริมาตรของโปรแกรมเท่ากับ 2 + N = 2 + 4 = 8

จากที่เราทราบอยู่แล้วว่าโปรแกรมหนึ่ง ๆ จะประกอบด้วยค่าหรือสัญลักษณ์ที่แตกต่างกันไป $n_1 + n_2$ ชนิด ดังนั้นจะต้อง encode $\log_2(n_1 + n_2)$ บิต ต่อ 1 operator หรือ 1 operand เช่น $n_1 + n_2 = 16$ คือในโปรแกรมนั้นมีค่าหรือสัญลักษณ์ทั้งหมด 16 ชนิด แต่ละชนิดจะ encode ด้วย $\log_2 16$ เท่ากับ 4 บิต

ดังนั้นสามารถคำนวณปริมาตรปริมาตรได้ดังนี้

$$V = N \times \log_2(n_1 + n_2)$$

2.3 ระดับของโปรแกรม (Level of abstraction)

เป็นระดับของโปรแกรมในทางนามธรรม คือ โปรแกรมที่ใช้ภาษาระดับต่ำกว่าจะให้ค่าน้อยกว่าโปรแกรมที่ใช้ภาษาชั้นสูงกว่า สามารถคำนวณได้จากสูตร

$$L = (2 \times n_2) / (n_1 \times n_2)$$

2.4 ความฉลาดของโปรแกรม (Intelligence Content)

เนื่องจากอัลกอริทึมหนึ่ง อาจสามารถเขียนโปรแกรมได้ในลักษณะที่แตกต่างกัน ทำให้ระดับและปริมาตรของโปรแกรมแตกต่างกัน ซึ่ง Halstead กำหนดให้ I แทนความฉลาดของโปรแกรม และคำนวณได้ดังนี้

$$I = L \times V$$

2.5 ความพยายามของโปรแกรม (Mental Effort)

ความพยายามของโปรแกรม หมายถึง ค่าที่แสดงถึงความพยายามของจิตใจในการเขียนโปรแกรมนั้น ๆ หรือเป็นความพยามที่ต้องการใช้สำหรับอ่านและเข้าใจโปรแกรม Halstead นิยามค่า E แทนความพยายามของโปรแกรม ดังนี้

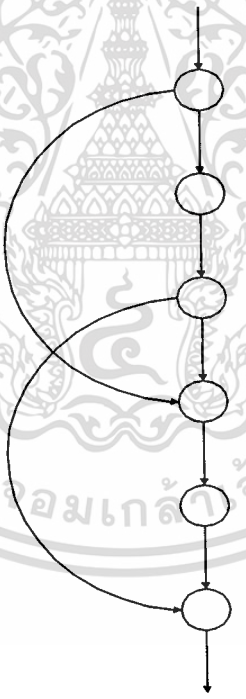
$$E = (n_1 * n_2(n_1+n_2) * \log_2(n_1+n_2)) / (2*n_2)$$

ซึ่งเชื่อว่าโปรแกรมที่มีความซับซ้อนมากน่าที่จะต้องการใช้ความพยายามมาก

3. ตัววัดปมของโปรแกรม (Knot Complexity)

Woodward Hennell และ Hedley ได้นำเสนอตัววัดปมของโปรแกรม นำมาใช้เป็นเกณฑ์แสดงถึงความซับซ้อน โดยพิจารณาจากโครงสร้างของโปรแกรม ซึ่งโปรแกรมจะเกิดปมก็ต่อเมื่อ โฟลร์กราฟสองเส้นทางตัดกันดังแสดงในรูป ถ้าโปรแกรมใดมีปมมากโปรแกรมนั้นจะถือว่าซับซ้อนเข้าใจยาก หรืออาจจะหาปมได้โดยการลากเส้นจากคำสั่งให้แยกทุกคำสั่งไปยังจุดหมายปลายทาง แล้วดูจุดตัดที่เกิดขึ้นมีกี่ที่ก็จะได้จำนวนปมที่เกิดขึ้นดังตัวอย่างในรูป

ตัวอย่างปมนี้นำเสนอครั้งแรกสำหรับภาษาฟอร์แทน แต่สำหรับการหาปมภาษชั้นสูงสมัยใหม่ เช่น ภาษาซี หรือ ปาสคาลอาจจะหาไม่ได้ เนื่องจากภาษาสมัยใหม่มักไม่ใช้ goto ดังนั้นตัววัดจำนวนปมจึงค่อนข้างล้าสมัยไม่เป็นที่นิยมใช้



ภาพที่ 2.2 แสดงการเกิดปมในโฟลร์กราฟ

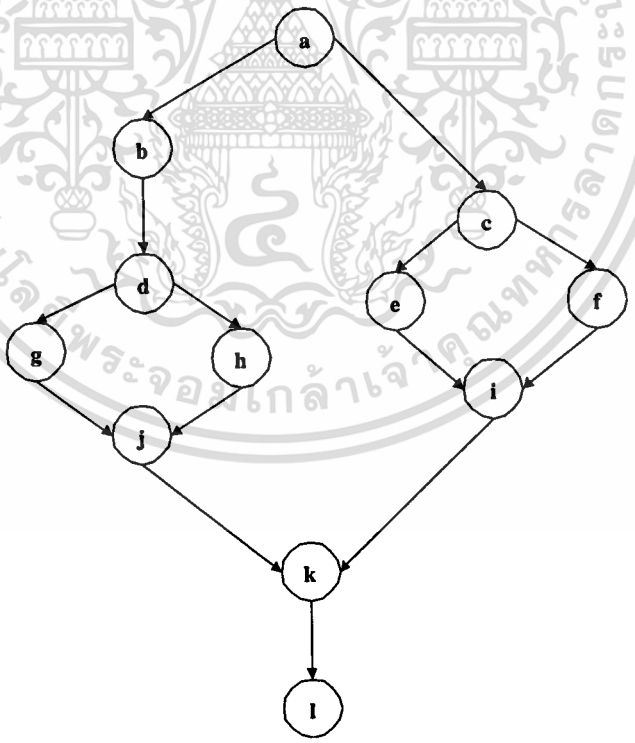
4. ตัววัดสโคป (Scope Complexity)

ตัววัดสโคปนี้เป็นแนวความคิดของ Harrison และคณะ คิดขึ้นเพื่อแก้ไขข้อเสียของตัววัดของ McCabe เนื่องจากว่าค่าไซโคลเมติกไม่สามารถบ่งบอกอะไรเกี่ยวกับโครงสร้างการซ้อนใน และขนาดของโปรแกรม

<u>นิยาม</u> ถ้าให้ G เป็นกราฟย่อยของโฟลร์กราฟ

1. ขอบเขตล่าง (lower bound : lb) ของ G คือโหนดหนึ่งใน G ที่ทุก ๆ โหนดใน G สามารถไปถึงได้

2. ขอบเขตล่างที่มากที่สุด (Greatest lower blound : glb) ของ G คือ ค่าขอบเขตล่างที่อยู่เหนือขอบเขตล่างอื่น ๆ ของ G หรือกล่าวคือ โหนดที่อยู่บนสุดในขอบเขตล่างนั้น ตัวอย่างดังรูป



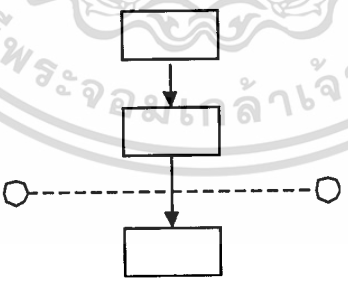
ภาพที่ 2.3 แสดงการหาค่าจากตัววัดสโคป

ถ้าเอาโหนด a เป็นหลัก จากโหนด a จะมีกราฟย่อย 2 กราฟย่อย 2 กราฟ ตั้งต้นที่โหนด b และโหนด c ทุก ๆ โหนดของกราฟทั้งสองนี้ไปถึงโหนด k และโหนด 1 ได้ ดังนั้น k และ 1 เป็น lower bound ของโหนด a และ k เป็น greatest lower bound ของโหนด a จากนั้นนับจำนวนโหนดทั้งหมดที่อยู่ระหว่าง a กับ k จะได้ค่าความซับซ้อนของโหนด a ในกรณีนี้คือ 10 โหนดสุดท้ายค่าความซับซ้อนจะเท่ากับ 0 สำหรับค่าความซับซ้อนของกราฟนี้ คือ ผลรวมค่าความซับซ้อนทั้งหมด ซึ่งเท่ากับ 25

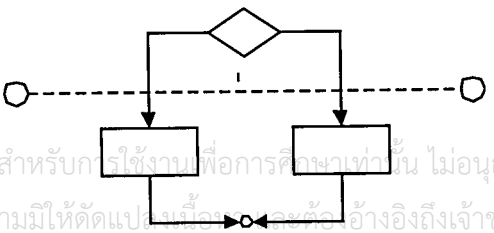
5. Chen's Metrics

Chen ได้เสนอระบบที่เรียกว่า Maximal Intersect Number หรือ MIN ซึ่งเป็นอีกวิธีที่หาค่าความซับซ้อนจากโฟลร์กราฟ โดยจะทำการวัดโครงสร้างการซ้อนใน (nesting structure) ของคำสั่งประเภทการตัดสินใจและวนซ้ำ การคำนวณตัววัดแบบนี้ โหนดเข้า (entry) และโหนดออก (exit) ของโฟลร์กราฟต้องเชื่อมโยงกันเป็นกราฟต่อเนื่อง (connected graph) และสามารถแบ่งออกเป็นบริเวณได้หลาย ๆ บริเวณ จากนั้นลากเส้นตัดผ่านเข้าไปยังแต่ละบริเวณ ค่า MIN ย่อยแต่ละบริเวณจะเท่ากับจำนวนจุดตัดที่มากที่สุด เมื่อลากเส้นตัดผ่านจากซ้ายไปขวาของโฟลร์กราฟย่อยนั้น ตัวอย่างเช่น

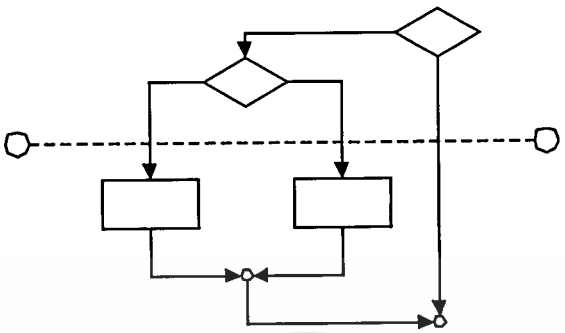
(1) MIN = 1



(2) MIN = 2



(3) MIN = 3



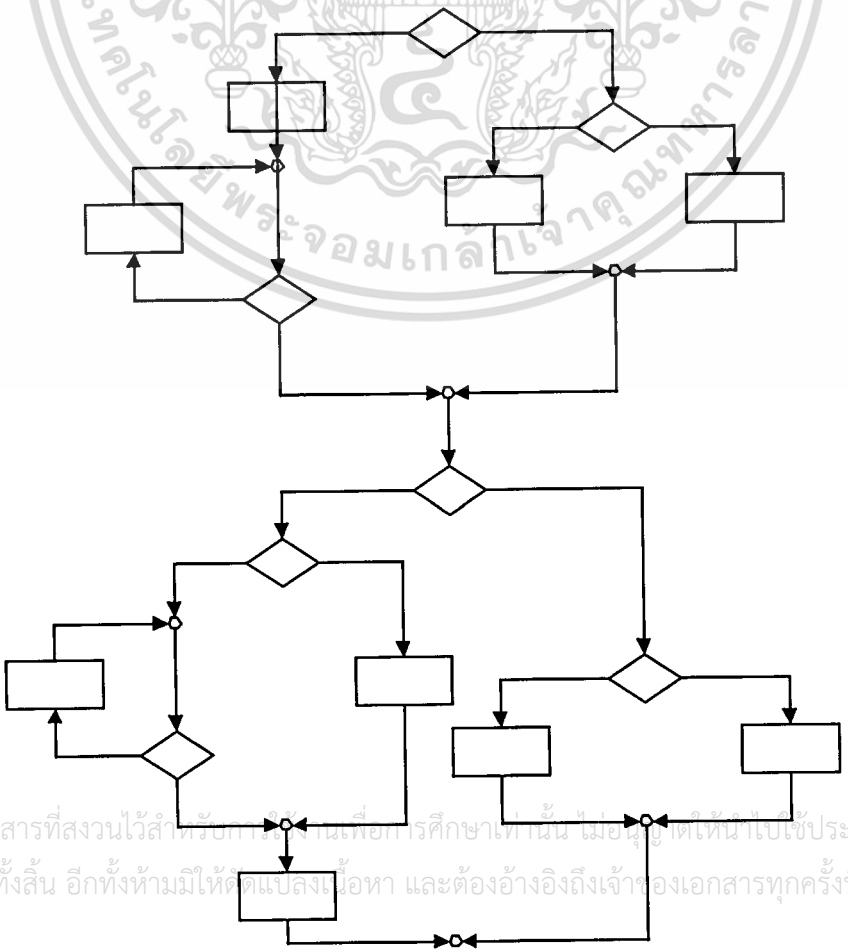
ค่า MIN รวมทั้งหมดหาได้จากสูตร

MIN = c – (2 * s) + 2

โดยที่ c คือ ผลรวมของ MIN ย่อยในแต่ละบริเวณ

s คือ จำนวนของบริเวณย่อยที่แบ่งได้ทั้งหมดใน CFG

ตัวอย่างจากรูป



จะได้ C = 4 + 5 = 9

S = 2 $\Rightarrow$ (1), (2)

$\therefore$ MIN รวม = 9 - (2 * 2) + 2 = 7

6. McClue's Metrics

ตัววัดของ McClue จะพิจารณาจากจำนวนการเปรียบเทียบและจำนวนตัวแปรควบคุมที่ถูกอ้างอิงภายในโมดูล แล้วนำไปคำนวณค่าความซ้อนได้จากสูตร

C(m) = C + V

โดยที่ C คือ จำนวนของการเปรียบเทียบภายในโมดูล m

V คือ จำนวนของตัวแปรควบคุมที่ถูกอ้างอิงภายในโมดูล m

ตัวอย่างการพิจารณาจำนวนของการเปรียบเทียบและตัวแปรควบคุม

```
Do While ((A == B) AND ( C == 1))
```

จะได้ # ครั้งการเปรียบเทียบ = 2 คือ (A = B) กับ (C = 1)

ตัวแปรควบคุม = 3 คือ A, B และ C (ค่าคงที่ 1 ไม่นับ)

7. Oviedo's Metrics

ตัววัด Oviedo พิจารณาจากโฟลร์กราฟและวิธีการใช้ข้อมูล ซึ่งเป็นวิธีที่วัดการไหลของข้อมูล (data flow) ด้วย Oviedo นิยามการวัดความซับซ้อนไว้ดังนี้

C = cf + df

โดยที่ cf คือ จำนวน edge ของโฟลร์กราฟ

df คือ ค่าการไหลของข้อมูล (data flow)

โดยนิยามค่า df เท่ากับผลบวกของค่า df พิจารณาตัวแปรในทุก ๆ โหนด นั่นคือ

$$df = \sum_{i=1}^{/v/} df_i$$

โดยที่ /v/ คือ จำนวนโหนดในโฟลร์กราฟ

df_i คือ จำนวนการนิยามก่อนของตัวแปรที่ได้รับการตีแผ่นเฉพาะถิ่นในโหนด n_j ที่ไปถึง n_i ได้ หรือจำนวนนิยามของตัวแปรที่ไปถึงโหนดที่พิจารณาได้

<u>ตัวแปรที่ได้รับการนิยาม</u> เมื่อได้รับค่าจากคำสั่งรับค่า หรือมีการรับค่าจากคำสั่งกำหนดว่า (assignment) หรือจากโปรแกรมย่อย เช่น

```
Scanf ("%d", X),   ⟹ (ภาษา C)  ┐
X = 10                          ├── การนิยามของตัวแปร X
Interchange (X, Y)              ┘
```

<u>ตัวแปรได้รับการใช้หรืออ้างอิง</u> หมายถึง เมื่อมีการใช้ค่าตัวแปรนั้นในคำสั่งประเภทกำหนดค่าหรือคำสั่งแสดงผลลัพธ์ เช่น

```
print x %
y = x + 10
If  x = 10  Then
```

<u>ตัวแปรจะมีค่าใช้ได้เฉพาะถิ่น</u> (locally available variable) ถ้าตัวแปรได้รับการนิยามในโหนดนั้น

<u>ตัวแปรจะได้รับการตีแผ่นเฉพาะถิ่น</u> (locally exposed variable) ถ้าใช้ตัวแปรนั้นในโหนดโดยที่ไม่มีการนิยามตัวแปรนี้ก่อน เช่น

```
y  =  10;
z  =  x+y;
x  =  z*2;
```

y, z เป็นตัวแปรใช้ได้เฉพาะถิ่น

x เป็นตัวแปรที่ได้รับการตีแผ่นเฉพาะถิ่น

บทที่ 3

การวัดค่าความซับซ้อนโดยใช้ทฤษฎีของแมคเคบ (A Complexity Measure)

3.1. การวัดค่าความซับซ้อนโดยใช้ทฤษฎีของแมคเคบ (A Complexity Measure) [2]

A Complexity Measure เสนอโดย Thomas J. McCabe ทฤษฎีนี้ทำการวิเคราะห์และควบคุมจำนวนทางเดิน (path) ของโปรแกรมโดยการหาค่าตัวเลขที่ใช้บอกความซับซ้อนของโปรแกรมว่ามีมากหรือน้อยเพียงใด เรียกว่า ค่าความซับซ้อนไซโคลเมติก (Cyclomatic Complexity Number) หรือ ค่าความซับซ้อนของซอฟต์แวร์ ซึ่งค่าความซับซ้อนไซโคลเมติกนี้ก็คือตัวเลขที่บอกจำนวนเส้นทางเดินพื้นฐานทั้งหมด (basis path) ที่โปรแกรมนั้นจะสามารถเลือกเส้นทางเดินหรือเลือกทำงานได้ การที่จะหาค่าความซับซ้อนไซโคลเมติกนี้ แมคเคบเสนอว่าให้นำโปรแกรมสร้างเป็นกราฟแสดงกระแสควบคุมการทำงาน (Control Flow Graph) โดยอาศัยทฤษฎีกราฟ (Graph Theory) มาช่วย กราฟกระแสการควบคุมนี้จะแสดงให้เห็นการควบคุมการทำงานของโปรแกรมว่ามีแนวทางใดได้บ้าง โดยแมคเคบเห็นว่าค่าความซับซ้อนของซอฟต์แวร์นั้นไม่ได้ขึ้นกับขนาดของซอฟต์แวร์ ที่ว่าถ้าโปรแกรมมีขนาดใหญ่มากก็จะเป็นซอฟต์แวร์ที่ซับซ้อนมากหรือถ้าโปรแกรมมีขนาดเล็กจะเป็นซอฟต์แวร์ที่ง่าย ไม่ซับซ้อนแต่จะขึ้นอยู่กับจำนวนคำสั่งที่ใช้ในเรื่องการตัดสินใจเลือกทางเดินของโปรแกรม (Decision Structure) จำพวกคำสั่ง if คำสั่ง case หรือคำสั่งเกี่ยวกับการวนซ้ำการทำงาน เช่น คำสั่ง while คำสั่ง until ว่ามีจำนวนมากน้อยเพียง ใด ถ้ามีจำนวนมากก็จะทำให้ได้ค่าความซับซ้อนไซโคลเมติกสูงแสดงว่าโปรแกรมมีความซับซ้อนมาก แต่ถ้าคำสั่งเกี่ยวกับการตัดสินใจมีใช้น้อย ค่าความซับซ้อนไซโคลเมติกก็จะต่ำโปรแกรมมีโอกาสเลือกทางเดินไม่มากมีความซับซ้อนน้อย ค่าความซับซ้อนไซโคลเมติกนี้สามารถหาได้ 3 วิธีคือ

3.1.1 หาได้โดยการแทนค่าในสมการ ซึ่งแทนค่าความซับซ้อนไซโคลเมติกด้วย v(G) หรือ v

$$v(G) = e - n + 2p$$

โดย G หมายถึง กราฟแสดงกระแสการควบคุมของโปรแกรมที่ต้องการหาค่า

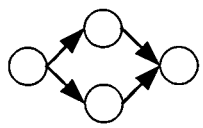
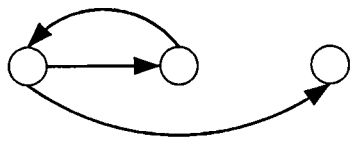
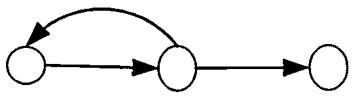
e " จำนวนเส้นลูกศรที่เชื่อมระหว่างโหนด ในกราฟ

n " จำนวนโหนดที่มีภายในกราฟ

p หมายถึง จำนวนโมดูล หรือ ฟังก์ชันการทำงานในกราฟ

p เป็นจำนวนทั้งหมดที่เราทำการคำนวณหาค่าถ้าเราหาค่าความซับซ้อนของโมดูลที่มีการเรียกทำงานโมดูลอื่น ด้วย ค่า p ก็จะมีค่าเท่ากับจำนวนโมดูลทั้งหมด ซึ่งก็คือ p จะมีค่ามากกว่า 1 แต่ถ้าหาค่าโมดูลนั้นๆ โดยไม่คำนึงถึงโมดูลอื่นที่ถูกเรียกใช้ด้วย ค่า p ก็จะเท่ากับ 1

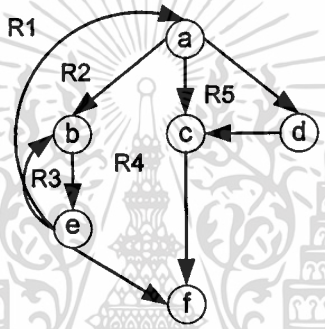
ในรูปภาพที่ 3.1 แสดงตัวอย่างของคำสั่งต่างๆ ว่าจะมีลักษณะโครงสร้างของกราฟอย่างไรและมีผลต่อค่าความซับซ้อนไซโคลเมติกอย่างไร ซึ่งเมื่อพิจารณาโครงสร้างของคำสั่งจะเห็นได้ว่า คำสั่งที่เป็นการทำงานแบบเรียงตามลำดับ (sequence) แม้จะมีจำนวนมากก็ไม่ทำให้ค่าความซับซ้อนไซโคลเมติกเพิ่มต่างกับคำสั่งที่เกี่ยวกับการตัดสินใจหรือวนซ้ำการทำงาน ที่ถ้าคำสั่งมีมากขึ้นก็จะทำให้ค่าความซับซ้อนไซโคลเมติกสูง ขึ้นด้วย

Control	Structure	Cyclomatic Complexity
SEQUENCE		V = 1 - 2 + 2(1) = 1
IF THEN ELSE		V = 4 - 4 + 2(1) = 2
W H I L E		V = 3 - 3 + 2(1) = 2
U N T I L		V = 3 - 3 + 2(1) = 2

ภาพที่ 3.1 แสดงตัวอย่างการหาค่าความซับซ้อนของคำสั่งที่เป็นเงื่อนไข หรือคำสั่งวนลูป

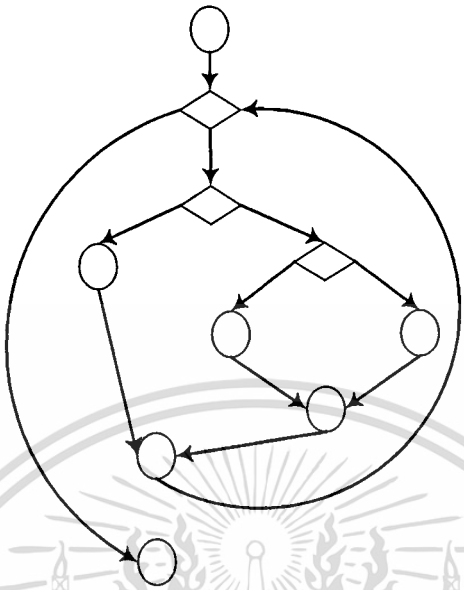
3.2 พิจารณาจากจำนวนพื้นที่ที่ถูกแบ่งเป็นส่วนๆ (region) ในกราฟ โดยรวมพื้นที่ที่อยู่ในเขตและนอกเขตกราฟ

ดังรูปภาพที่ 3.2 ให้ R แทนพื้นที่ที่จะนับ จะได้ว่าค่าความซับซ้อนไซโคลเมติกเท่ากับจำนวนของ R = 5



ภาพที่ 3.2 แสดงการหาค่าความซับซ้อนโดยนับจำนวนพื้นที่ (Region)

3.3 ค่าความซับซ้อนไซโคลเมติก จะเท่ากับจำนวน คำสั่งเงื่อนไข (predicate) + 1 ดังรูปภาพที่ 3.3 ให้ '<>' แทน predicate จะได้ค่าความซับซ้อนไซโคลเมติกเท่ากับจำนวน predicate + 1 เท่ากับ 3 + 1 = 4

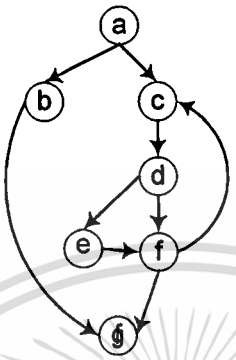


ภาพที่ 3.3 แสดงการหาค่าความซับซ้อนโดยใช้วิธีนับจำนวนคำสั่งเงื่อนไข (Predicate)

```
JOB INPUT NULL
  IF CUSTOMER-ID > 256000
     DISPLAY "CUSTOMER-ID IN BRANCH"
  ELSE
       DO WHILE INDX1 LE 2
          IF INDEX2 LE 3
             DISPLAY "INDEX2"
          END-IF
       END-DO
    END-IF
END
```

ภาพที่ 3.4 ตัวอย่างโปรแกรมภาษา EASYTRIEVE PLUS ประกอบด้วย 1 โมดูล

จากภาพที่ 3.4 จากโปรแกรมนี้ ซึ่งจะมีเพียง 1 โมดูล เราสามารถแทนด้วยกราฟกระแสการควบคุมได้ตามภาพที่ 3.5



ภาพที่ 3.5 ภาพกระแสการควบคุมที่ได้จากตัวอย่างโปรแกรมในรูปที่ 3.4

จากภาพที่ 3.5 จะได้ว่าค่าความซับซ้อนของโปรแกรมนี้ เท่ากับ 9 - 7 + 2(1) = 4

```
JOB INPUT NULL
    DO WHILE  INDX1  LE  2
          PERFORM  READLINE
          DISPLAY  "PERFORM"
    END-DO
READLINE PROC.
    DO WHILE  INDX2  LE  2
          DISPLAY  "READLINE"
    END-DO
END-PROC
END
```

ภาพที่ 3.6 ตัวอย่างโปรแกรมภาษา EASYTRIEVE PLUS ที่มีมากกว่า 1 โมดูล

จากรูปภาพที่ 3.6 ซึ่งเป็น โปรแกรมที่มีการเรียกฟังก์ชันให้ทำงาน ซึ่งวาดเป็นกราฟกระแสการควบคุมได้เป็น 2 กราฟย่อยดังรูปภาพที่ 3.7

กราฟของฟังก์ชัน main กราฟของฟังก์ชัน read_line

ภาพที่ 3.7 แสดงกราฟกระแสการควบคุมที่ได้จากโปรแกรมในรูปภาพที่ 3.6

จากภาพที่ 3.7 สามารถหาค่าความซับซ้อนไซโคลเมติก ได้โดยแทนค่าสมการให้ e = 4 n = 4 และ p = 2 จะได้ 4 - 4 + 2(2) = 4 หรือแยกหาของแต่ละกราฟแยกกัน จะได้ค่าความซับซ้อนไซโคลเมติก

ฟังก์ชัน main = 2 - 2 + 2(1) = 2 และ
ฟังก์ชัน read_line = 2 - 2 + 2(1) = 2

ซึ่งจะได้ค่าความซับซ้อนไซโคลเมติก รวมเท่ากับค่าความซับซ้อนไซโคลเมติก ของฟังก์ชัน main + ค่าความซับซ้อนไซโคลเมติก ของฟังก์ชัน read_line เท่ากับ 2 + 2 = 4

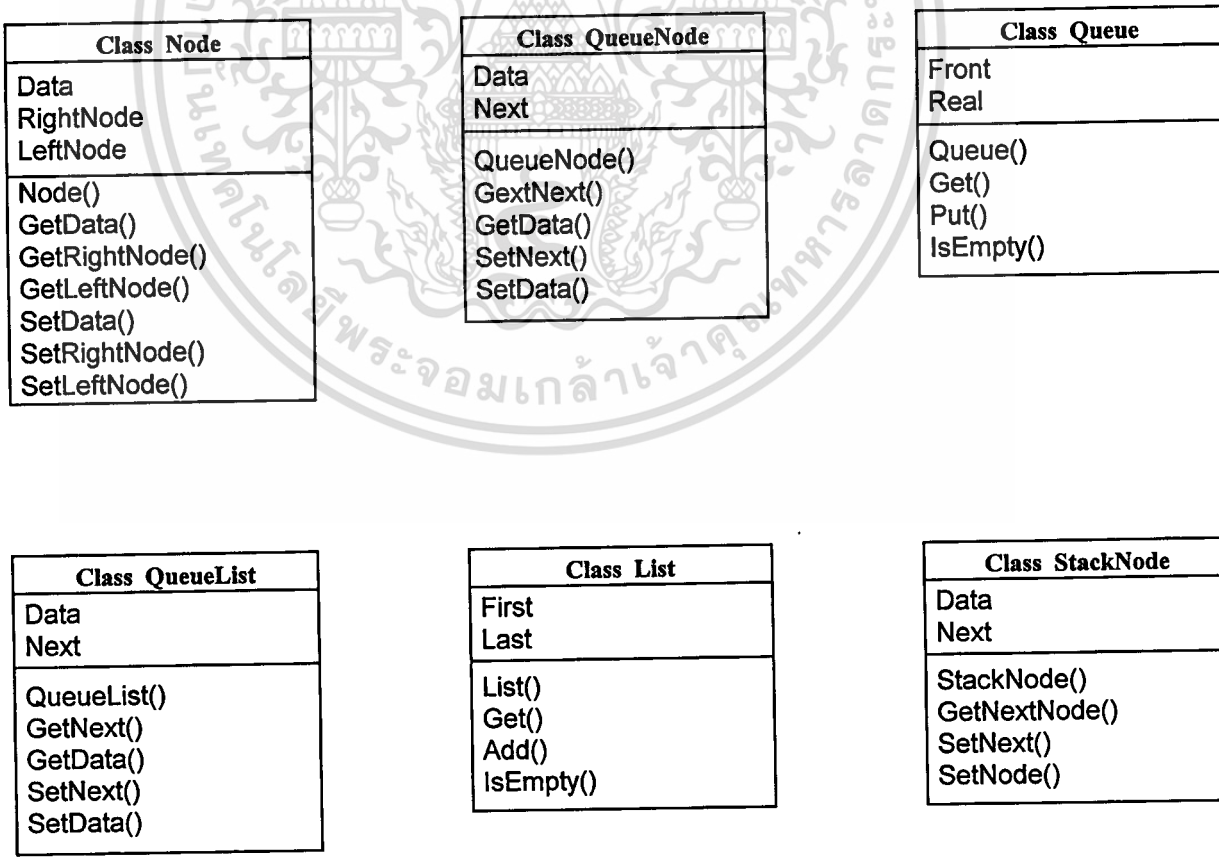
จะเห็นได้ว่า ยิ่งค่าความซับซ้อนไซโคลเมติกมากขึ้น แสดงว่าซอฟต์แวร์มีการทำงานที่เกี่ยวกับการตัดสินใจมาก ทำให้โปรแกรมมีความซับซ้อนยิ่งขึ้น การจะทดสอบโปรแกรมก็จะต้องมีกรณีทดสอบหลายกรณีเพิ่มขึ้น ทำให้การทดสอบได้ลำบาก ดังนั้น แมคเคบจึงได้เสนอให้จำกัดขนาดของซอฟต์แวร์ด้วยค่าวัดความซับซ้อนไซโคลเมติกแทนการวัดจากขนาดของซอฟต์แวร์ โดยแนะนำว่าค่าความซับซ้อนไซโคลเมติกควรมีค่าไม่เกิน 10 โดยทั่วๆไป ซอฟต์แวร์ที่มีการเขียนเป็นโครงสร้างที่ดีจะมีค่าความซับซ้อนอยู่ระหว่าง 3-7 เท่านั้น ดังนั้นเมื่อใดที่ นักเขียนโปรแกรมเขียนโปรแกรมแล้ว ก็ควรจะตรวจสอบซอฟต์แวร์ที่เขียนว่ามีค่าความซับซ้อนไซโคลเมติกเท่าไหร่

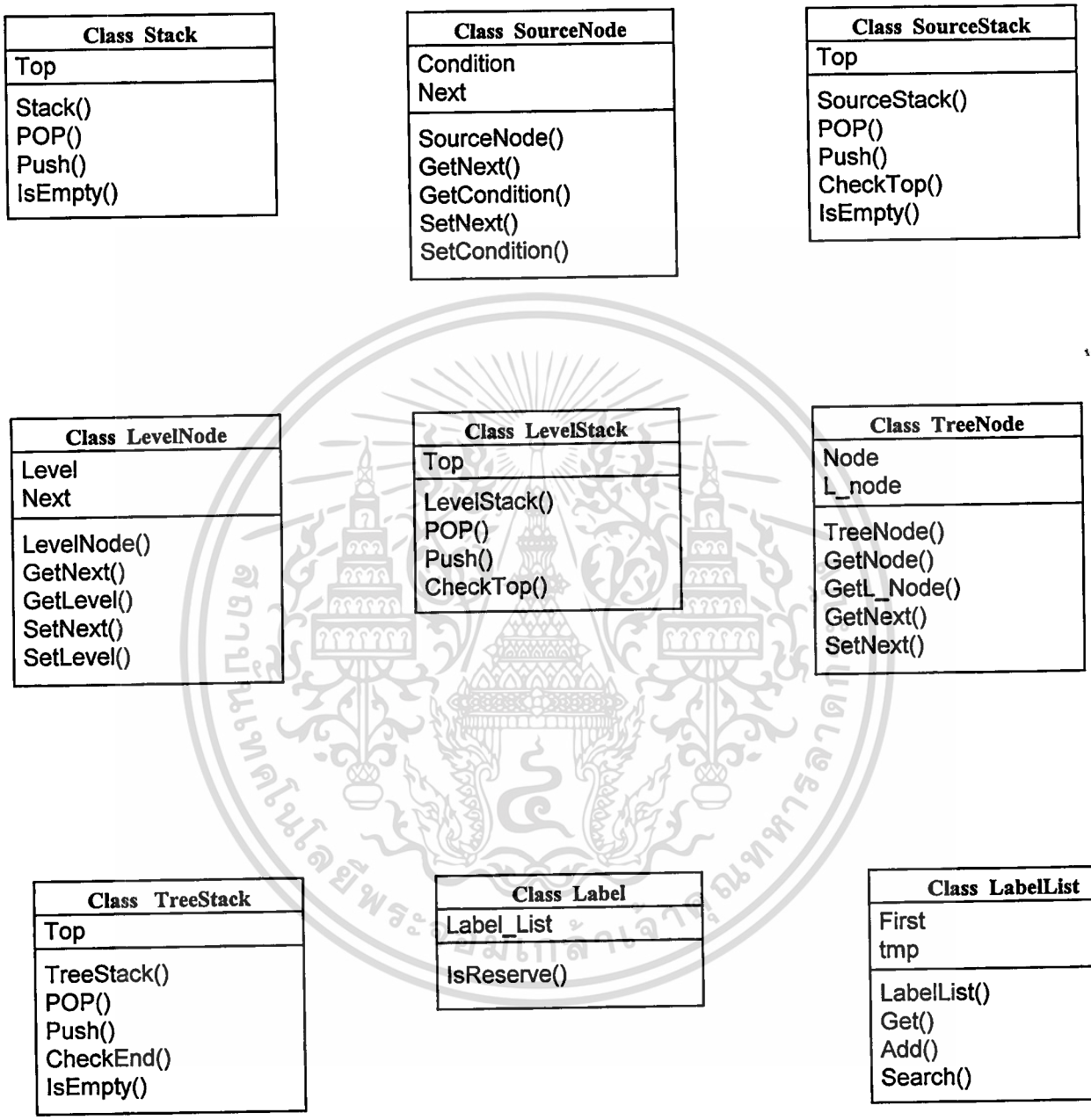
ถ้าเกิน 10 ก็แสดงว่าควรจะทำการแบ่งหรือแยกย่อยออกเป็นโมดูลหรือฟังก์ชันย่อยๆ หลายๆส่วน เพื่อให้การทำงานในโปรแกรมเดิมมีความซับซ้อนน้อยลง และเพื่อให้ขนาดของซอฟต์แวร์อยู่ในระดับที่สามารถดำเนินการทดสอบในทุกๆแนวทางได้ครบถ้วนโดยใช้กรณีทดสอบน้อยและประหยัดเวลา ทำให้ได้ซอฟต์แวร์ที่มีประสิทธิภาพมากขึ้น และสามารดูแลรักษาได้ดียิ่งขึ้น

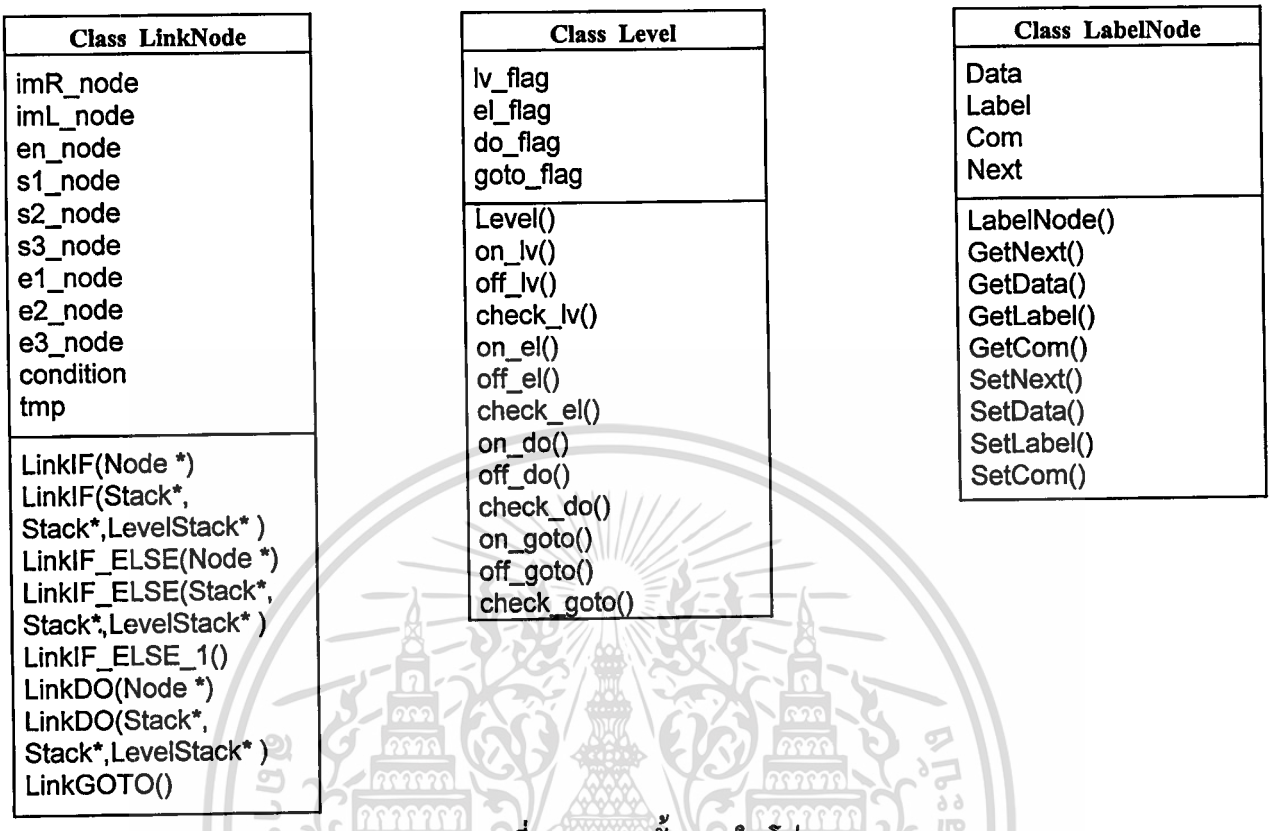
บทที่ 4

การออกแบบระบบ (System Design)

ในบทนี้จะนำเอาทฤษฎีในบทที่ 3 และขอบเขตของโครงการในบทที่ 1 มาทำการออกแบบระบบเพื่อพัฒนาโปรแกรม วัดค่าความซับซ้อนของโปรแกรมตามทฤษฎีการวัดค่าความซับซ้อนของซอฟต์แวร์ของแมคเคบ โดยการออกแบบระบบนี้เป็นการออกแบบระบบแบบ Object-Oriented Design ตามเทคนิคและวิธีการของ Edward Yourdon จากรูปภาพที่ 4.1 จะแสดงถึง Class ทั้งหมดในระบบ







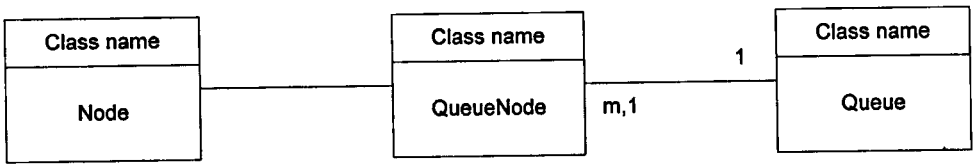
ภาพที่ 4.1 Class ทั้งหมดในโปรแกรม

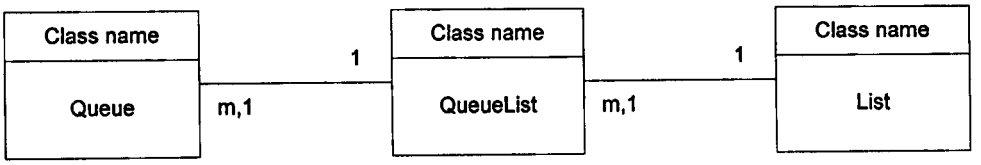
รายละเอียดการทำงานของแต่ละ Class

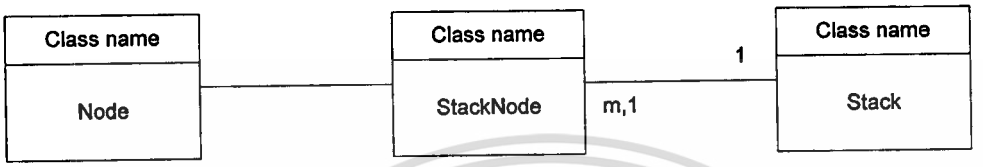
- Class Node เป็น Class ของ Node ที่ใช้แทน Node ใน Direct Graph โดยแต่ละ Node จะมีตัวชี้ชี้ไปที่โหนดที่อยู่ภายใต้
- Class Queue Node เป็น Class ของ Datatype item ภายใน Queue โดยนำ Class Node มาสร้างเป็น Class Queue Node เพื่อใช้เป็น item ในการสร้าง Queue ของ Node นั้น
- Class Queue เป็น Class ที่นำ Queue Node มาทำงานร่วมกันในลักษณะโครงสร้างการทำงานของ Queue เพื่อใช้แสดงถึงว่าในแต่ละ Node ใน Graph G มีโหนดใดอยู่ภายใต้โหนดนั้นๆบ้าง
- Class QueueList เป็น Class ที่นำ Class Queue มาสร้างเป็น Datatype item ใน LinkList เพื่อใช้เป็น item ในการสร้าง LinkList ของ Class Queue
- ClassList เป็น Class ที่นำ Class QueueList มาเชื่อมต่อกันในลักษณะโครงสร้างการทำงานแบบ LinkList เพื่อใช้แสดงถึงโหนดทุกๆโหนด และการเชื่อมต่อกันระหว่างโหนดที่อยู่ภายใต้ทั้งหมดภายใน Graph G
- Class StackNode เป็น Class ที่ใช้เป็น Datatype item ภายใน stack โดยการนำเอา Class Node มาใช้เป็น Datatype item

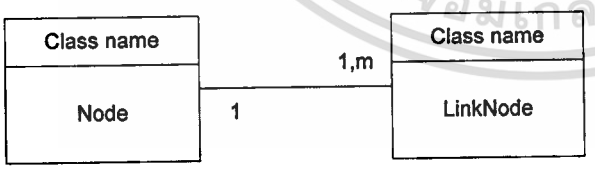
- Class Stack เป็น Class ที่นำ Class Stack Node มาสร้างเป็นโครงสร้างการทำงานแบบ stack โดยใช้ในการเก็บที่อยู่ของโหนดเริ่มต้นและโหนดสิ้นสุดในแต่ละกราฟ ย่อยภายใน Graph G
- Class SourceNode เป็น Class ที่ใช้เป็น Datatype item ภายใน Class SourceStack ใช้ในการเก็บ stack ของคำสั่งของภาษา " EASYTRIEVE PLUS " ที่เกี่ยวกับการตัดสินใจหรือการทำซ้ำ เช่น IF เป็นต้น
- Class SourceStack เป็น Class ที่นำเอา Class SourceNode มาเป็น Datatype item ในการทำงานแบบ stack โดยใช้ในการจัดเก็บคำสั่งที่เกี่ยวข้องกับการตัดสินใจหรือมีผลกระทบต่อโครงสร้างของโปรแกรม
- Class LevelNode เป็น Class ที่ใช้เป็นความลึกหรือลำดับขั้นของกราฟย่อยภายในโปรแกรม
- Class LevelStack เป็น Class ที่นำเอา Class LevelNode มาเป็น Datatype item ในการทำงานแบบ stack ใช้ในการเก็บความลึกหรือลำดับขั้นของกราฟย่อยภายใน Graph G โดยใช้คู่กับ Class Stack
- Class Label เป็น Class ที่ใช้เก็บคำสงวนที่ไม่สามารถใช้เป็น Label ภายในโปรแกรมทั้งหมด เพื่อใช้ทดสอบว่า คำนั้นๆมีคุณสมบัติเป็น Label หรือไม่
- Class LabelList เป็น Class ที่ใช้เก็บตำแหน่งของ Label ทั้งหมดและเก็บตำแหน่งของ Node ที่เป็นจุดชี้ออกของคำสั่ง GOTO ทั้งหมดโดยมีโครงสร้างแบบ LinkList
- Class LinkNode เป็น Class หลักในการเชื่อมกราฟย่อยๆภายในกราฟ G ทั้งหมดเข้าด้วยกัน โดยพิจารณาตามลำดับขั้นของแต่ละกราฟย่อย และเงื่อนไขของคำสั่งที่ใช้ในการ Link เช่น IF-ELSE เป็นต้น
- Class Level เป็น Class ที่ใช้เก็บของชนิดของคำสั่งต่างๆภายในโปรแกรม เช่น IF-ELSE-END-IF , DO-END-DO เป็นต้น
- Class Label Node เป็น Class ที่ใช้เก็บชื่อของ Label และตำแหน่งของ Label นั้นๆและเงื่อนไขก่อนการเกิด Label ใช้ในการทำการเชื่อมโยงจากจุดออกของคำสั่ง GOTO กับตำแหน่งของ Label นั้นๆ

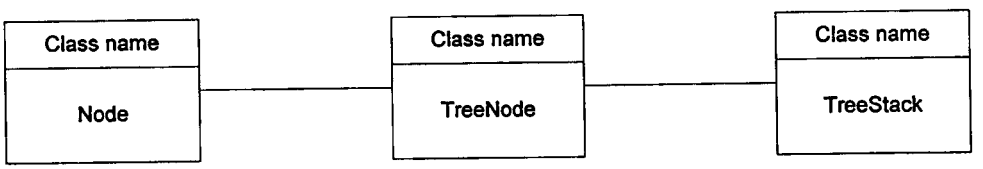
จากภาพที่ 4.2 จะแสดงถึงความสัมพันธ์ของวัตถุของแต่ละ Class ในโปรแกรม

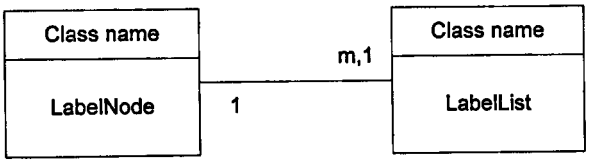


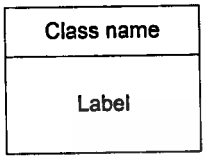


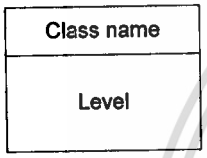






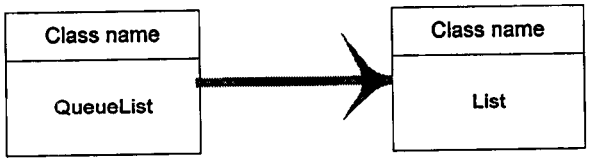


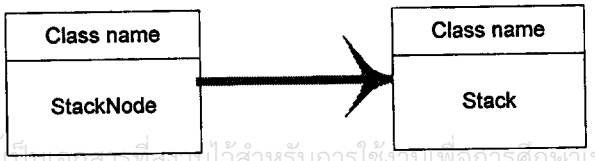


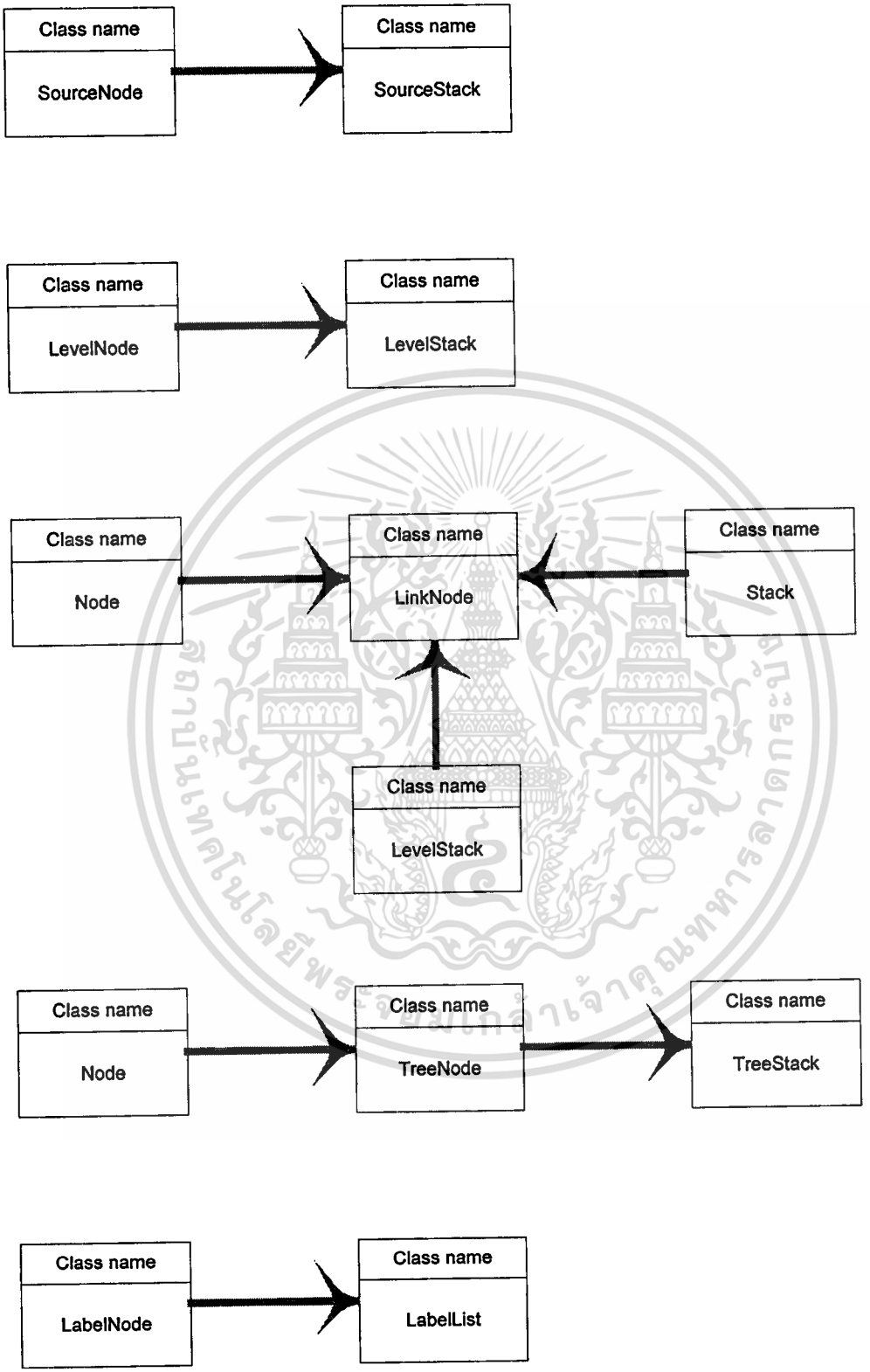


ภาพที่ 4.2 ความสัมพันธ์ระหว่างวัตถุของแต่ละ Class

จากภาพที่ 4.3 จะแสดงให้เห็นถึงความสัมพันธ์ของวัตถุของแต่ละ Classในแง่ของการเป็นผู้รับ และผู้ส่งข้อความระหว่างกัน



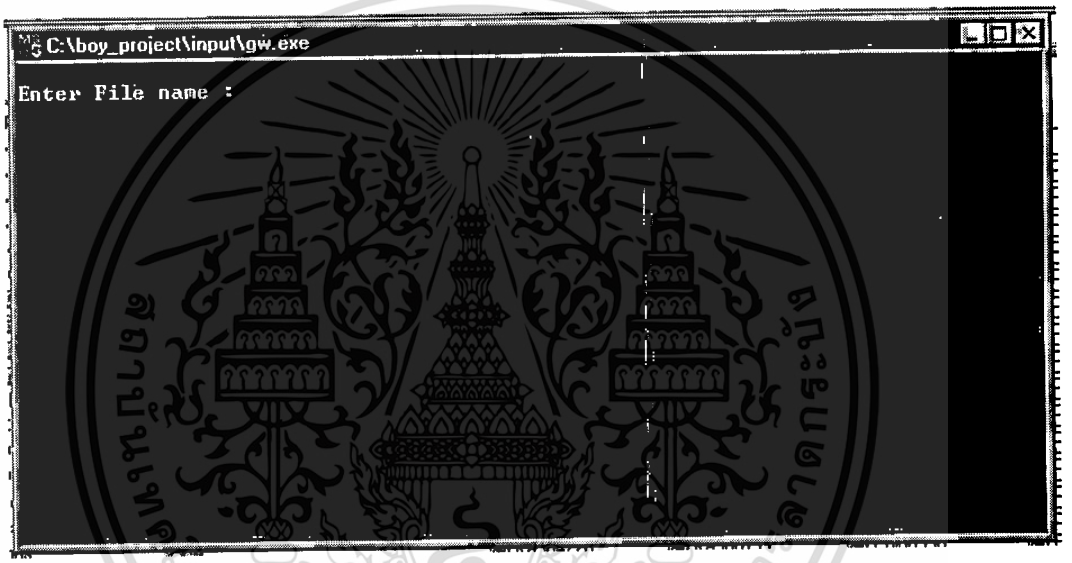




ภาพที่ 4.3 ความสัมพันธ์ของวัตถุของแต่ละ Class ในการรับส่งข้อมูลระหว่างกัน

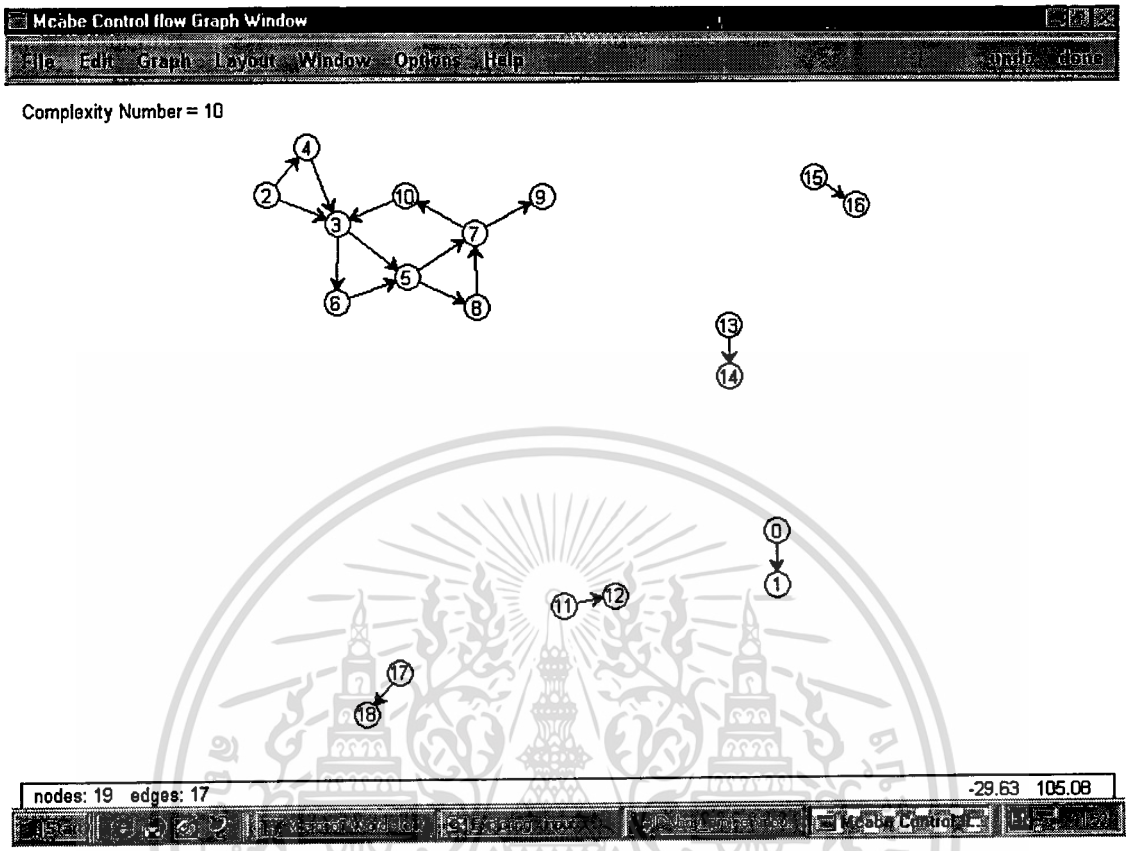
ภายหลังจากการสร้างโครงสร้างแบบต้นไม้จากข้อมูลที่นำเข้ามาในรูปแบบของ EASYTRIEVE PLUS โปรแกรมแล้วจะนำมาสร้างเป็นวัตถุของ Class Graph โดยเก็บรายละเอียด 2 ส่วน คือ โหนดทั้งหมดในกราฟ และรายละเอียดของการเชื่อมต่อระหว่างโหนดใดๆในกราฟ

โปรแกรมส่วนที่ใช้ในการแสดงผลกราฟ ได้นำเอา LEDA Library มาใช้ในการแสดงผล โดยการส่งวัตถุของ Class Graph ที่ได้จากโปรแกรมไปให้ จากภาพที่ 4.4 จะเป็นตัวอย่างหน้าจอในการรับข้อมูลชื่อแฟ้มข้อมูลที่จะทำการวิเคราะห์เข้าสู่ระบบ



ภาพที่ 4.4 หน้าจอในการรับข้อมูลแฟ้มข้อมูล

จากภาพที่ 4.5 จะเป็นหน้าจอที่แสดงผลลัพธ์ของระบบในรูปแบบของ Graph และแสดงตัวเลข Complexity Number ของแฟ้มข้อมูลที่ทำการวิเคราะห์



ภาพที่ 4.5 หน้าจอแสดงผลลัพธ์ของระบบในรูปแบบของ Graph

บทที่ 5
ผลการศึกษา

จากการที่ได้ทำการพัฒนาระบบในการตรวจวัดค่าความซับซ้อนของโปรแกรม โดยใช้ทฤษฎีการวัดค่าความซับซ้อนของแมคเคบและใช้ภาษา EASYTRIEVE PLUS ในการตรวจวัด และได้ทำการตรวจวัดโปรแกรมภาษา EASYTRIEVE PLUS จำนวน 150 โปรแกรม ซึ่งได้ผลดังตารางที่ 5.1

Value	Frequency	Percent	Valid Percent	Cum Percent
2.00	2	1.3	1.3	1.3
3.00	1	.7	.7	2.0
4.00	2	1.3	1.3	3.3
5.00	1	.7	.7	4.0
6.00	2	1.3	1.3	5.3
7.00	2	1.3	1.3	6.7
8.00	1	.7	.7	7.3
9.00	1	.7	.7	8.0
10.00	2	1.3	1.3	9.3
12.00	3	2.0	2.0	11.3
13.00	2	1.3	1.3	12.7
14.00	5	3.3	3.3	16.0
15.00	2	1.3	1.3	17.3
16.00	3	2.0	2.0	19.3
17.00	3	2.0	2.0	21.3
18.00	1	.7	.7	22.0
19.00	7	4.7	4.7	26.7

20.00	2	1.3	1.3	28.0
21.00	2	1.3	1.3	29.3
22.00	2	1.3	1.3	30.7
23.00	7	4.7	4.7	35.3
24.00	2	1.3	1.3	36.7
25.00	2	1.3	1.3	38.0
26.00	1	.7	.7	38.7
27.00	1	.7	.7	39.3
28.00	4	2.7	2.7	42.0
29.00	1	.7	.7	42.7
30.00	4	2.7	2.7	45.3
31.00	1	.7	.7	46.0
32.00	3	2.0	2.0	48.0
33.00	1	.7	.7	48.7
34.00	2	1.3	1.3	50.0
35.00	4	2.7	2.7	52.7
36.00	4	2.7	2.7	55.3
38.00	2	1.3	1.3	56.7
39.00	3	2.0	2.0	58.7
42.00	1	.7	.7	59.3
45.00	1	.7	.7	60.0
47.00	1	.7	.7	60.7
48.00	2	1.3	1.3	62.0
49.00	2	1.3	1.3	63.3
50.00	2	1.3	1.3	64.7
52.00	4	2.7	2.7	67.3
53.00	1	.7	.7	68.0
54.00	2	1.3	1.3	69.3
56.00	1	.7	.7	70.0

57.00	2	1.3	1.3	71.3
58.00	3	2.0	2.0	73.3
59.00	3	2.0	2.0	75.3
60.00	2	1.3	1.3	76.7
61.00	1	.7	.7	77.3
62.00	1	.7	.7	78.0
64.00	1	.7	.7	78.7
65.00	2	1.3	1.3	80.0
66.00	1	.7	.7	80.7
67.00	3	2.0	2.0	82.7
68.00	1	.7	.7	83.3
69.00	1	.7	.7	84.0
70.00	1	.7	.7	84.7
71.00	1	.7	.7	85.3
75.00	1	.7	.7	86.0
77.00	1	.7	.7	86.7
78.00	1	.7	.7	87.3
79.00	1	.7	.7	88.0
81.00	2	1.3	1.3	89.3
82.00	2	1.3	1.3	90.7
84.00	1	.7	.7	91.3
87.00	2	1.3	1.3	92.7
91.00	2	1.3	1.3	94.0
92.00	1	.7	.7	94.7
93.00	2	1.3	1.3	96.0
98.00	2	1.3	1.3	97.3
121.00	1	.7	.7	98.0
142.00	1	.7	.7	98.7
149.00	1	.7	.7	99.3

182.00	1	.7	.7	100.0
Total	150	100.0	100.0	100.0

ตารางที่ 5.1 ค่าความซับซ้อนของโปรแกรม

จากตารางที่ 5.1 จะได้ค่าทางสถิติดังนี้

Mean 42.280 Median 34.500

Mode 19.000 Sum 6342.000

และนอกจากจะได้ทำการตรวจวัดค่าความซับซ้อนของโปรแกรมแล้วยังได้ทำการตรวจวัดขนาดความยาวของโปรแกรมในหน่วยของจำนวนบรรทัดทั้งหมดของโปรแกรม ได้ผลดังตารางที่ 5.2

Value	Frequency	Percent	Valid Percent	C u m Percent
30.00	1	.7	.7	.7
51.00	1	.7	.7	1.3
58.00	1	.7	.7	2.0
76.00	1	.7	.7	2.7
78.00	1	.7	.7	3.3
93.00	2	1.3	1.3	4.7
99.00	1	.7	.7	5.3
100.00	1	.7	.7	6.0
104.00	1	.7	.7	6.7
131.00	1	.7	.7	7.3
146.00	1	.7	.7	8.0
168.00	1	.7	.7	8.7
171.00	1	.7	.7	9.3
175.00	2	1.3	1.3	10.7
183.00	1	.7	.7	11.3

184.00	1	.7	.7	12.0
185.00	1	.7	.7	12.7
197.00	1	.7	.7	13.3
200.00	1	.7	.7	14.0
202.00	1	.7	.7	14.7
203.00	1	.7	.7	15.3
209.00	1	.7	.7	16.0
212.00	1	.7	.7	16.7
217.00	1	.7	.7	17.3
220.00	1	.7	.7	18.0
225.00	1	.7	.7	18.7
226.00	1	.7	.7	19.3
229.00	1	.7	.7	20.0
232.00	1	.7	.7	20.7
234.00	1	.7	.7	21.3
246.00	1	.7	.7	22.0
261.00	1	.7	.7	22.7
264.00	1	.7	.7	23.3
267.00	1	.7	.7	24.0
282.00	1	.7	.7	24.7
284.00	1	.7	.7	25.3
289.00	1	.7	.7	26.0
293.00	1	.7	.7	26.7
297.00	2	1.3	1.3	28.0
301.00	1	.7	.7	28.7
303.00	1	.7	.7	29.3
308.00	1	.7	.7	30.0
315.00	1	.7	.7	30.7
316.00	1	.7	.7	31.3

317.00	2	1.3	1.3	32.7
325.00	1	.7	.7	33.3
328.00	1	.7	.7	34.0
336.00	1	.7	.7	34.7
337.00	1	.7	.7	35.3
344.00	1	.7	.7	36.0
348.00	1	.7	.7	36.7
358.00	1	.7	.7	37.3
365.00	1	.7	.7	38.0
370.00	1	.7	.7	38.7
371.00	1	.7	.7	39.3
376.00	1	.7	.7	40.0
380.00	1	.7	.7	40.7
403.00	1	.7	.7	41.3
404.00	2	1.3	1.3	42.7
413.00	1	.7	.7	43.3
421.00	1	.7	.7	44.0
423.00	1	.7	.7	44.7
425.00	1	.7	.7	45.3
429.00	1	.7	.7	46.0
446.00	1	.7	.7	46.7
448.00	1	.7	.7	47.3
450.00	1	.7	.7	48.0
452.00	1	.7	.7	48.7
460.00	1	.7	.7	49.3
463.00	1	.7	.7	50.0
470.00	1	.7	.7	50.7
474.00	1	.7	.7	51.3
480.00	1	.7	.7	52.0

483.00	2	1.3	1.3	53.3
484.00	2	1.3	1.3	54.7
485.00	1	.7	.7	55.3
488.00	2	1.3	1.3	56.7
491.00	1	.7	.7	57.3
493.00	1	.7	.7	58.0
494.00	1	.7	.7	58.7
495.00	4	2.7	2.7	61.3
503.00	2	1.3	1.3	62.7
505.00	1	.7	.7	63.3
506.00	1	.7	.7	64.0
510.00	1	.7	.7	64.7
515.00	1	.7	.7	65.3
517.00	1	.7	.7	66.0
528.00	1	.7	.7	66.7
544.00	1	.7	.7	67.3
545.00	1	.7	.7	68.0
554.00	1	.7	.7	68.7
566.00	1	.7	.7	69.3
568.00	1	.7	.7	70.0
571.00	1	.7	.7	70.7
572.00	1	.7	.7	71.3
578.00	1	.7	.7	72.0
579.00	1	.7	.7	72.7
588.00	1	.7	.7	73.3
594.00	1	.7	.7	74.0
608.00	1	.7	.7	74.7
612.00	1	.7	.7	75.3
615.00	1	.7	.7	76.0

621.00	1	.7	.7	76.7
633.00	1	.7	.7	77.3
640.00	1	.7	.7	78.0
647.00	1	.7	.7	78.7
651.00	1	.7	.7	79.3
655.00	1	.7	.7	80.0
676.00	1	.7	.7	80.7
680.00	1	.7	.7	81.3
685.00	1	.7	.7	82.0
698.00	2	1.3	1.3	83.3
699.00	1	.7	.7	84.0
725.00	1	.7	.7	84.7
738.00	1	.7	.7	85.3
744.00	1	.7	.7	86.0
767.00	1	.7	.7	86.7
771.00	1	.7	.7	87.3
788.00	1	.7	.7	88.0
793.00	1	.7	.7	88.7
798.00	1	.7	.7	89.3
802.00	1	.7	.7	90.0
863.00	1	.7	.7	90.7
879.00	1	.7	.7	91.3
884.00	1	.7	.7	92.0
892.00	2	1.3	1.3	93.3
894.00	1	.7	.7	94.0
913.00	1	.7	.7	94.7
925.00	1	.7	.7	95.3
1043.00	1	.7	.7	96.0
1074.00	1	.7	.7	96.7

1230.00	1	.7	.7	97.3
1834.00	1	.7	.7	98.0
1957.00	1	.7	.7	98.7
2409.00	1	.7	.7	99.3
2813.00	1	.7	.7	100.0
Total	150	100.0	100.0	100.0

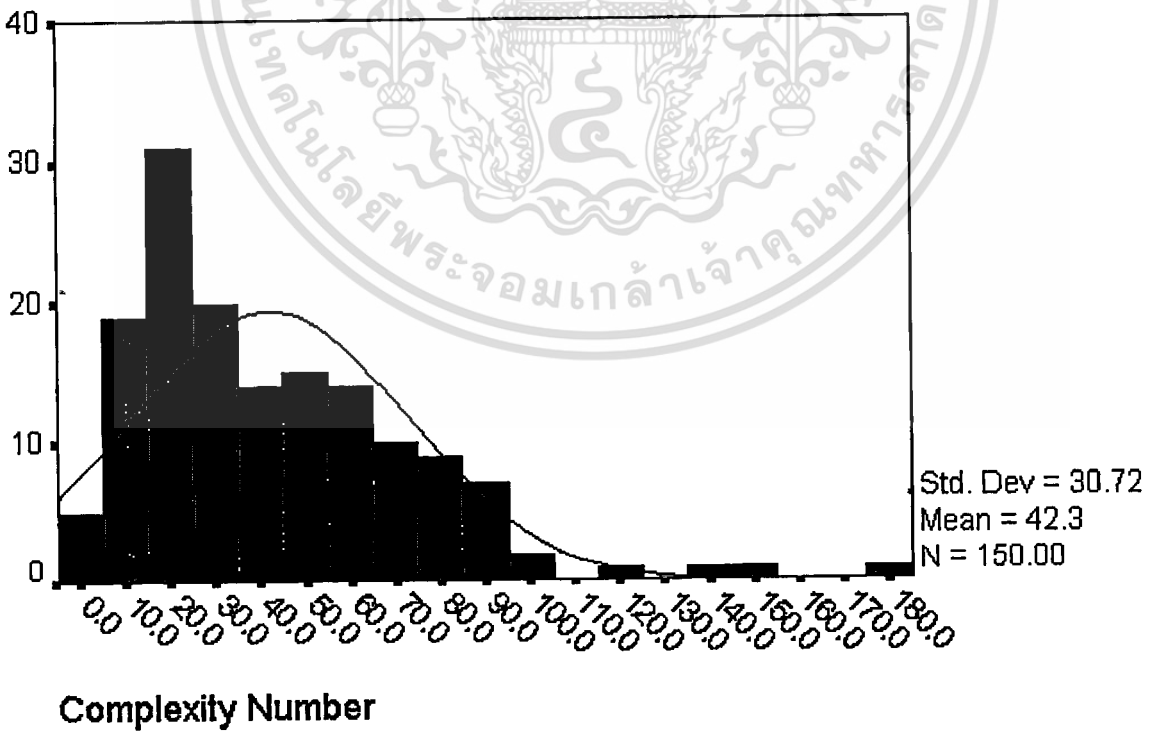
ตารางที่ 5.2 แสดงค่าความยาวของโปรแกรม

จากตารางที่ 5.2 จะได้ค่าทางสถิติดังนี้

Mean 501.407 Median 466.500

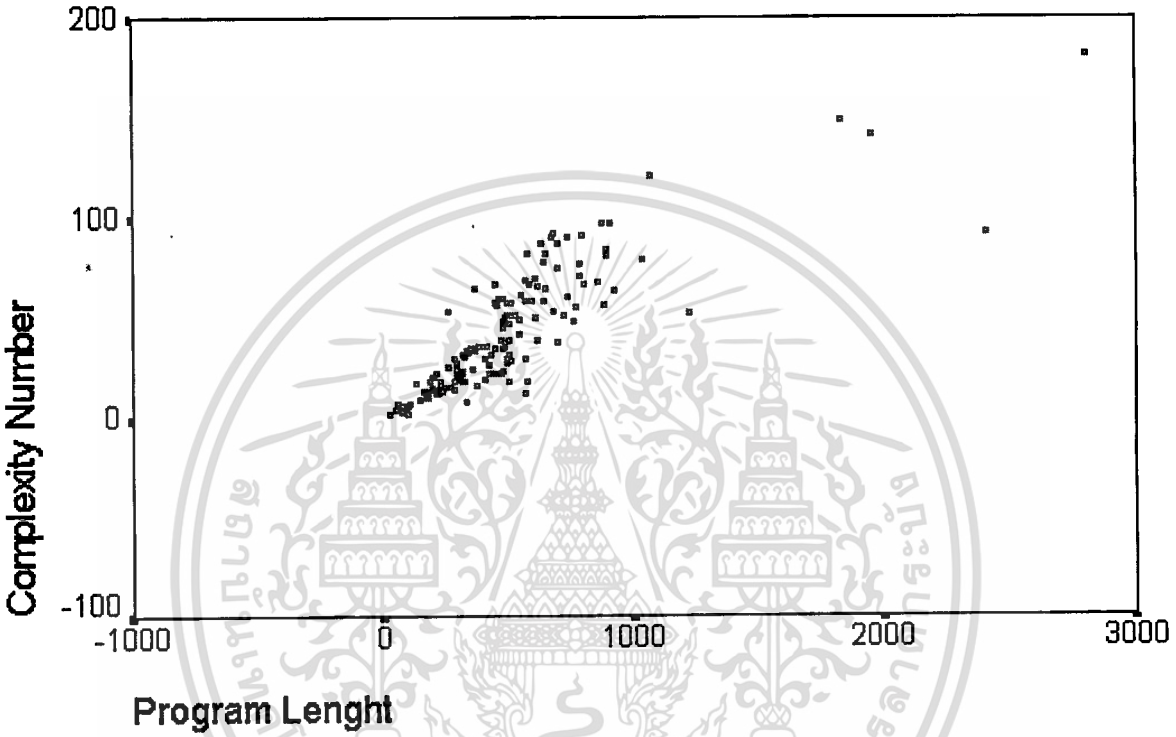
Mode 495.000 Sum 75211.000

และเมื่อนำข้อมูลค่าความซับซ้อนของโปรแกรมที่ทำการตรวจวัดได้มาสร้างฮีสโตแกรมกราฟจะได้กราฟดังภาพที่ 5.1



รูปภาพที่ 5.1 ฮีสโตแกรมกราฟ

และได้นำข้อมูลที่ได้ของค่าความซับซ้อนและขนาดของโปรแกรมที่จัดเก็บได้มาสร้างกราฟ เพื่อหาความสัมพันธ์ระหว่างข้อมูลทั้งสองตัวนี้ ได้กราฟดังภาพ 5.2



ภาพที่ 5.4 ความสัมพันธ์ระหว่างค่าความซับซ้อนและขนาดของโปรแกรม

บทที่ 6
บทสรุปและข้อเสนอแนะ

6.1 สรุปผลโครงการ

จากผลการศึกษาที่ได้และเห็นว่าค่าเฉลี่ยของค่าความซับซ้อนที่ได้มีค่าที่ค่อนข้างสูง (42.280) ซึ่งเกินกว่าค่าของแมคเคบที่เสนอไว้ว่าควรอยู่ในช่วง 7 ถึง 10 แต่เนื่องจากค่าความซับซ้อนของซอฟต์แวร์ในแต่ละองค์กรย่อมขึ้นอยู่กับสภาพแวดล้อมในการพัฒนา เครื่องมือที่ใช้ในการพัฒนา ลักษณะของงานในแต่ละองค์กร และความสามารถหรือประสิทธิภาพของผู้พัฒนา จากลักษณะของของภาษาและลักษณะของการทำงานของโปรแกรมที่ได้นำมาทดสอบจะเป็นลักษณะของการออกรายงาน และการจัดแฟ้มข้อมูลซึ่งจะมีลักษณะของการทดสอบเงื่อนไขและการวนรอบซ้ำ จึงทำให้ค่าความซับซ้อนที่ได้มีค่าสูงไปด้วย

แต่อย่างไรก็ตามค่าเฉลี่ยของค่าความซับซ้อนที่ได้จากการทดลองนี้สามารถนำมาใช้เป็นค่าเริ่มต้นของค่าความซับซ้อนมาตราฐานในการยอมรับได้ของการพัฒนาซอฟต์แวร์ แล้วจึงปรับเปลี่ยนค่าเริ่มต้นนี้ให้เหมาะสมกับองค์กร และเมื่อได้ค่าดังกล่าวแล้วจึงนำมาใช้เป็นเกณฑ์ของค่าความซับซ้อนของซอฟ์ตแวร์ก่อนที่จะทำการส่งไปให้ฝ่ายตรวจสอบทำการทดสอบซอฟ์ตแวร์

6.2 ข้อเสนอแนะ

เนื่องจากการใช้ค่าความซับซ้อนเพียงอย่างเดียวในการวัดคุณภาพของซอฟต์แวร์หรือการประเมินขีดความสามารถของผู้พัฒนาระบบอาจจะไม่เพียงพอ ควรมีการวัดค่าของคุณสมบัติด้านอื่นๆของซอฟต์แวร์แล้วนำมาพิจารณาประกอบด้วย เช่น ค่าที่แสดงถึงความสามารถของซอฟต์แวร์ เป็นต้น และเนื่องจากการพัฒนาระบบในการตรวจวัดค่าความซับซ้อนของซอฟต์แวร์ เราจะได้โปรแกรมที่สามารถสร้าง CFG ได้ ดังนั้นจึงสามารถนำมาพัฒนาเพื่อหากรณีทดสอบอย่างน้อย ในการทำการทดสอบระบบ และนำมาสร้าง Program Flow Chart จาก Source Code ได้

บรรณานุกรม

Barkakati, Nabajyoti. Object-Oriented Programming in C++. A Division of Macmilan Computer Publishing, Carmel, Indiana, 1991

Fenton, Norman E. and Lawrence Pfleeger. Software Metrics. PWS Publishing Company, London, 1997

Gansner, Enden R. , and others, A Technique for Drawing Directed Graphs. IEEE Transaction Engineering. Vol 19 (March 1993): 214-229

Gorewich, Nathan and Ori Gorewich. Mastering C++ From C to C++ in 2 weeks. Tech Publication PTE Ltd., Singapore, 1994

Horstmann, Cay S. Mastering C++. John Wiley & Son, New York, 1996

McCabe, Thomas J. A Complexity Measure. IEEE Transaction on Software Engineering. SE-2, (December 1976): 308-320

Sedgewick, Robert. Algorithms in C++. Addision-Wesley Publishing Company, Menlo Park, California, 1992

ปราโมทย์ ลือนาม, การพัฒนาโปรแกรมเพื่อวัดความซับซ้อนของซอฟต์แวร์. ปริญญานิพนธ์วิศวกรรมศาสตร์มหาบัณฑิต. กรุงเทพ:บัณฑิตวิทยาลัย จุฬาลงกรณ์มหาวิทยาลัย, 2536

ประวัติผู้เขียน

ประวัติส่วนตัว

ชื่อผู้เขียน	นายพรศักดิ์ แซ่ก๊วย
วันที่เกิด	15 มกราคม 2514
สถานที่เกิด	กรุงเทพมหานคร

ประวัติการศึกษา

มัธยมศึกษา	โรงเรียนทวีทาภิเษก
อุดมศึกษา	วท.บ.(สถิติ) มหาวิทยาลัยเกษตรศาสตร์

ประวัติการทำงาน

ธนาคารกรุงเทพ จำกัด (มหาชน) ตำแหน่ง SYSTEM ANALYST