บทที่ 13
วิศวกรรมซอฟต์แวร์ (Software Engineering)

บทนำ เนื้อหาในบทก่อนหน้านี้ จะเห็นได้ว่าโปรแกรมที่เราดำเนินการออกแบบและเขียนนั้นจะเป็นโปรแกรมขนาดสั้นเพื่อสนองวัตถุประสงค์เพียงอย่างเดียว ซึ่งโดยความเป็นจริงแล้วระบบโปรแกรมที่สร้างขึ้นเพื่อใช้งานมักจะเป็นโปรแกรมขนาดใหญ่ที่จะต้องสนองตอบต่อความต้องการหลายลักษณะตามความต้องการของผู้ใช้ นอกจากนี้ระบบงานที่ใช้กันทั่วไปบางระบบก็จะมีความซับซ้อนจนกระทั่งต้องปรับโปรแกรมเพื่อให้สามารถรองรับในการทำงานได้อย่างมีประสิทธิภาพ ด้วยสาเหตุดังกล่าวนี้จึงทำให้โปรแกรมนั้นๆมีขนาดใหญ่มาก (Programming in the Large) จนต้องออกแบบโดยการแบ่งออกเป็นส่วนย่อยๆที่เรียกว่าโมดุล

การออกแบบโปรแกรมขนาดใหญ่ (Program in the Large)

การออกแบบโปรแกรมขนาดใหญ่นั้น ผู้ออกแบบมืออาชีพนั้นจะใช้เครื่องมือ (Tool) และเทคนิคหลายอย่างเข้าช่วยดำเนินการ ในทุกขั้นตอนตั้งแต่ขั้นตอน ออกแบบ (Program Design) การถอดรหัส (Coding) การทดสอบ (Testing) ตลอดจนถึงการนำไปใช้งาน (Implementation)

ดังนั้นการเรียนรู้เรื่องเกี่ยวกับการใช้เครื่องมือต่างๆซชเพื่อดำเนินงานในการสร้างโปรแกรมจึงเป็นเรื่องที่จำเป็นอย่างยิ่ง และในการศึกษาเรื่องดังกล่าว ก็คือหัวข้อของวิชาการทางคอมพิวเตอร์ที่เรียกว่า วิศวกรรมซอฟต์แวร์ (Software Engineering) การศึกษายุทธวิธีต่างๆในเรื่องของ วิศวกรรมซอฟต์แวร์จะช่วยให้ผู้สร้างโปรแกรมขนาดใหญ่ได้ออกแบบ สร้าง และทดสอบโปรแกรม ได้อย่างมีประสิทธิภาพ ภายใต้เงื่อนไขของทรัพยากรที่มีอยู่อย่างจำกัด จึงมีความพยายามในการดำเนินงานเพื่อให้เกิดการพัฒนาในสิ่งต่อไปนี้ เพื่อให้การพัฒนาซอฟต์แวร์เป็นไปได้อย่างราบรื่น

- การนำโปรแกรมกลับมาใช้งานใหม่ (Reuse Program)
- การนำโปรแกรมที่ดำเนินการในงานหนึ่งมาขยายผลให้สามารถใช้งานกับอีกงานหนึ่งโดยการปรับแก้ที่น้อยที่สุด
- การออกแบบในรูปแบบของการแบ่งส่วน Modular Programming ที่จะช่วยให้การสร้างโปรแกรมง่ายขึ้น และทำให้สามารถแบ่งงานกันในทีมผู้สร้างโปรแกรม
- การนำโปรแกรมย่อยที่เรียกว่าโมดุลมาเชื่อมโยง (Software Interface) เข้าด้วยกัน
- วิธีการการทดสอบโปรแกรมขนาดใหญ่

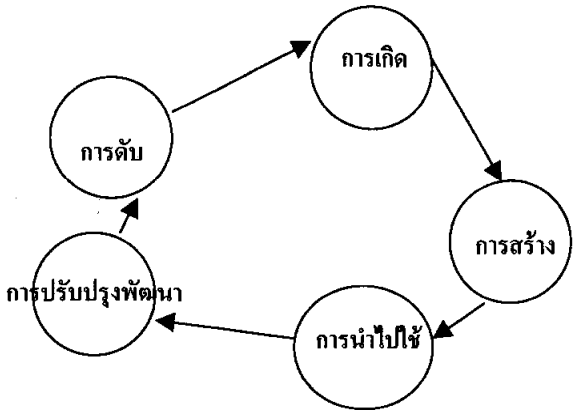
การสร้างโปรแกรมขนาดใหญ่ การสร้างโปรแกรมขนาดใหญ่นั้นผู้สร้างจะเผชิญกับปัญหาที่แตกต่างกับการเขียนโปรแกรมเดี่ยวขนาดเล็ก ทั้งนี้เพราะโปรแกรมขนาดใหญ่จะต้องมีการบริหารงาน

การติดต่อกับโปรแกรมย่อยๆนอกเหนือจากภาระกิจในการประสานงานระหว่างทีมของผู้เขียนโปรแกรม ทั้งในส่วนของการแบ่งงาน การทดสอบ เป็นต้น

ระบบซอฟต์แวร์และวัฏจักรของซอฟต์แวร์

การสร้างโปรแกรมหรือที่เรียกว่า ซอฟต์แวร์ขึ้นมาใช้นั้นระหว่างบุคลากรสองกลุ่มที่อยู่คนละสถานะคือ ถ้าเป็นผู้ที่มีอาชีพเป็นนักศึกษาเป้าหมายก็คงจะมีแต่เพียงเจตนาจะให้โปรแกรมที่ตนสร้างขึ้นมานั้นแก้ปัญหาตามที่อาจารย์กำหนดก็พอ ไม่ได้มุ่งหวังที่จะสามารถพัฒนาไปได้ตามสภาพแวดล้อมที่จะเปลี่ยนไปในอนาคต ในขณะที่ผู้ที่ประกอบอาชีพเป็นโปรแกรมเมอร์นั้น จะมีเจตนาที่จะสร้างโปรแกรมให้ตรงกับความต้องการของผู้ใช้ และในขณะเดียวกันก็มุ่งหวังที่จะให้โปรแกรมนั้นสามารถสนองตอบต่อความต้องการที่เป็นไปได้ในทุกสถานะการณ์นอกจากนั้นยังจะต้องสามารถพัฒนาให้เข้ากับสภาพแวดล้อมที่แปรเปลี่ยนไปในอนาคตที่เรียกว่าเป็น Dynamic Programming ยกเว้นในกรณีที่สภาพแวดล้อมเปลี่ยนไปจากเดิมมากจนเกินกว่าความสามารถในการพัฒนาปรับปรุงซอฟต์แวร์ได้จึงจะดำเนินการสร้างขึ้นมาใหม่ จากลำดับเหตุการณ์ที่กล่าวมานี้จะเห็นได้ว่าวัฏจักรของระบบกับวัฏจักรของซอฟต์แวร์ก็มีสภาพคล้ายๆกันคือมี เกิด การใช้ การปรับ และสุดท้ายคือการดับ

ภาพที่ 13.1 วัฏจักรของซอฟต์แวร์



สรุปวัฏจักรของการสร้างซอฟต์แวร์ (Software Life Cycle :SLC)ได้ดังนี้

1. การกำหนดความต้องการของระบบ (Requirement Specification)
   a. การกำหนดขอบเขตและวัตถุประสงค์ของความต้องการ
   b. เขียนกำหนดขอบเขตความต้องการของผู้ใช้และของนักวิเคราะห์ระบบ

2. การวิเคราะห์ (Analysis)
    a. ทำความเข้าใจกับปัญหา เริ่มตั้งแต่สิ่งที่ต้องการ รวมทั้งเรื่องของส่วนนำข้อมูลเข้าและเสนอผล (Input/Output)
    b. การเสาะหาทางเลือกในการดำเนินงาน
    c. ประเมินทางเลือกที่ดีสุด ภายใต้ข้อจำกัดของระบบและทรัพยากร
3. การออกแบบ (Design)
    a. ออกแบบโดยใช้วิธีการจากบนลงล่าง (Top Down Design)
    b. จากแต่ละโปรแกรม ย่อย (Module) ให้ทำการแยกแยะข้อมูล และการออกแบบทำงานย่อยโดยใช้ตัวแบบแบบมีโครงสร้าง (Structure Program)
4. การนำไปใช้ใช้งาน (Implementation)
    a. ลำดับขั้นตอนในการทำงานและเขียนโดยใช้เครื่องมืออย่างใดอย่างหนึ่ง เช่น Pseudo Program หรือ ผังโปรแกรม
    b . นำเอกสารในข้อ a. ไปถอดรหัสเป็นโปรแกรม (Coding)
    c. ตรวจสอบโปรแกรม (Debug The Coding)
5. ตรวจสอบและทดสอบโปรแกรม (Testing and Verification)
    a. ทดสอบโปรแกรมและแก้ไขโปรแกรม
    b. ทดสอบผู้ใช้และกลุ่มผู้ชำนาญการ
6. นำไปปฏิบัติงาน ติดตาม และให้การบำรุงดูแลรักษา
    a. นำโปรแกรมทั้งระบบไปปฏิบัติงาน
    b. ประเมินศักยภาพของระบบโปรแกรม
    c. ตรวจสอบและแก้ไขความผิดพลาดใหม่ที่เกิดขึ้นจากการพัฒนาโปรแกรม
    d. แก้ไขความต้องการโปรแกรมใหม่ให้ตรงกับความต้องการที่เปลี่ยนแปลงไป
    e. ตรวจสอบว่าการพัฒนาโปรแกรมนั้น มีผลทำให้ระบบการทำงานเปลี่ยนแปลงหรือไม่

ขั้นตอนทั้ง 6 ที่กล่าวมานี้ขั้นตอนที่ 1,3,4 จัดว่าเป็นขั้นตอนการสังเคราะห์ ส่วนขั้นตอนที่ 2 คือการวิเคราะห์ ขั้นตอนในการสังเคราะห์นั้นผู้ใช้ในระบบจะต้องร่วมมือในการช่วยดำเนินการ ระยะเวลาที่ใช้ไปค่อนข้างจะมากคือขั้นตอนที่ 1-4 ในแต่ละขั้นตอนนั้นผู้ดำเนินงานจะต้องเขียนเอกสารกำกับการทำงานที่ได้พร้อมกันไปด้วย

กรณีศึกษา เพื่อที่จะแสดงขั้นตอนการปฏิบัติงานทั้ง 6 ขั้นตอนดังกล่าวมานี้

ปัญหา ต้องการให้สร้างระบบโปรแกรมเพื่อใช้ในการสอบถามเรื่องโทรศัพท์ โดยกำหนดความต้องการดังนี้คือ ให้มีการเก็บข้อมูลของผู้ใช้โทรศัพท์แต่ละรายอยู่จำนวนหนึ่ง โดยที่ผู้ใช้แต่ละรายจะมีองค์ประกอบข้อมูลดังนี้ ชื่อและหมายเลขโทรศัพท์ โดยที่บัญชีรายชื่อของผู้ใช้โทรศัพท์สามารถ ลบ เพิ่ม แก้ไขได้

การออกแบบโปรแกรมจะมีขั้นตอนดังนี้ ลำดับแรกจะต้องจะต้องมีการกำหนดความต้องการของผู้ใช้ว่าโครงสร้างและอุปกรณ์ของส่วนนำเข้าข้อมูล และส่วนนำออกสารสนเทศ คืออะไร และในการรับข้อมูลเข้า นั้นจำเป็นจะต้องกำหนดการตรวจสอบความถูกต้องของข้อมูลหรือไม่ (Data Validation) หรือไม่ ซึ่งเราจะต้องสอบถามข้อมูลจากผู้ใช้ในระบบดังรายการต่อไปนี้

- ข้อมูลเริ่มต้นของผู้ใช้โทรศัพท์นั้นมีปรากฏอยู่แล้วหรือไม่ โครงสร้างข้อมูลเป็นเช่นใด และจัดเก็บไว้ในสื่อใด
- ในกรณีของข้อมูลที่จัดเก็บไว้แล้วในแฟ้มข้อมูลนั้นสามารถนำไปดำเนินการได้โดยสะดวกหรือไม่ หรือต้องการจะต้องมีการสร้างหรือแปลงแฟ้มข้อมูลก่อนหรือไม่ ตัวอย่างเช่นแฟ้มข้อมูลนั้นมีลักษณะเป็น Text File จะต้องมีการแปลงให้เป็นเป็น Record File เสียก่อน
- แฟ้มข้อมูลของรายชื่อผู้ใช้โทรศัพท์นั้นมีรายชื่อเจ้าของโทรศัพท์ที่ซ้ำซ้อนกันหรือไม่ หรือในกรณีที่มีผู้ใช้โทรศัพท์รายใดรายหนึ่งสามารถครอบครองโทรศัพท์ได้หลายหมายเลขหรือไม่
- ความยาวของรายการชื่อเจ้าของโทรศัพท์มีขีดจำกัดความยาวหรือไม่ และชื่อของเจ้าของโทรศัพท์มีรูปแบบอย่างไร เช่น ชื่อต้น ชื่อกลาง ชื่อท้าย หรือ เป็นรูปแบบของ ชื่อท้าย ชื่อต้น
- ลักษณะการจัดเก็บหมายเลขโทรศัพท์ เป็นรูปแบบอย่างไร เช่น 374-4331 (มีการใช้เครื่องหมาย – แบ่งตำแหน่งของตัวเลข)

ภายหลังจากที่เก็บรวบรวมข้อมูลในขั้นตอนนี้แล้ว ขั้นตอนต่อไปเราจำไปสรุปเขียนเป็นความต้องการ

ต้นแบบ (Prototype)

วิถีทางในการสร้างแบบโปรแกรมแบบดั้งเดิม (Traditional Program) นั้น เราจะนำความต้องการทั้งหลายมาสร้างเป็นเป็นต้นแบบ (Prototype) โดยที่ขั้นตอนนี้จะประกอบด้วยการสร้างโมเดลของระบบงานจริง (Actual System) มาให้ตรวจสอบ และปรับแก้ (Refine) ขั้นตอนของการสร้างต้นแบบจะใช้เวลาประมาณ 1 สัปดาห์ โดยการใช้กับข้อมูลขนาดย่อ การตรวจสอบต้นแบบนั้นจะถูกดำเนินการโดยผู้ใช้และนัก

วิเคราะห์ระบบ ว่าถูกต้อง ครบถ้วนหรือ ภายหลังเมื่อต้นแบบได้ถูกแก้ไขจนเสร็จสิ้นแล้ว ผู้สร้างโปรแกรมจะใช้ต้นแบบนั้นในการสร้างโปรแกรมจริงต่อไป

ทีมงานผู้สร้างซอฟต์แวร์ (Programming Team) การสร้างซอฟต์แวร์เชิงอุตสาหกรรมนั้น มักจะใช้ทีมงานขนาดใหญ่ในการเขียนโปรแกรม โดยการแบ่งเป็นโปรแกรมย่อย โดยการใช้ผัง HIPO เป็นเครื่องมือในการแบ่งภาระงานในการเขียนโปรแกรมให้กับผู้เขียนโปรแกรมแต่ละคนดำเนินการเขียน ภายหลังเมื่อผู้เขียนโปรแกรมดำเนินงานของตนเองเสร็จ ก็จะนำโปรแกรมไปทดสอบโดยการใช้รูปแบบของ Bottom Up หลังจากนั้นจึงนำมาเชื่อมโยงกันและทดสอบโดยใช้รูปแบบของ String Testing เพื่อศึกษาว่าการเชื่อมต่อนั้นกระทำสำเร็จหรือไม่

คลังของโปรแกรมย่อย (Module Libraries)

การออกแบบโปรแกรมโดยแบ่งเป็นส่วนที่เรียกว่าโปรแกรมย่อยหรือโมดุล หรือที่เรียกว่า Procedure or Function ในบางภาษา นั้น จะมีข้อได้เปรียบหลายประการเมื่อเทียบกับการออกแบบแบบโปรแกรมเดี่ยวดังนี้คือ

- การแบ่งเป็นโมดุลทำให้การพัฒนาโปรแกรมก็ทำได้ง่าย
- การใช้วิธีการของโมดุล ทำให้สามารถเอาชนะปัญหาและอุปสรรค ได้เช่นเดียวกับเทคนิคการปกครองที่เรียกว่า "แบ่งแยกและปกครอง" (Divide and Conquer) วิธีการออกแบบที่นั้น จัดเป็นรูปแบบของ Top Down เหมือนสายการบังคับบัญชา
- การใช้วิธีการของ Procedural Abstraction Revised (PAR) กับ โมดุล วิธีการของ PAR นั้นจะตั้งอยู่บนพื้นฐานที่ว่า เราจะสร้างการทำงานอย่างไรจึงจะบรรลุเป้าหมายที่ต้องการ (Achieve Procedure) ให้ดูตัวอย่างจาก PAR ต่อไปนี้

	Main Program	**Procedures**
WHAT	Procl : Proc2 :	Procedure Procl : begin end ; [Procl] Procedure Proc2 ; begin end, [Proc2]

จากภาพนี้ Main Program จะเรียก 2 Procedure ย่อยคือ Proc1 และ Proc2 ส่วนการที่ Proc1 และ Proc2 จะต้องดำเนินการอย่างไรเพื่อให้เป็นไปตามวัตถุประสงค์ของ Main Program น ผู้รับผิดชอบในการเขียน Proc1 และ Proc2 จะต้องไปดำเนินการออกแบบและสร้างเองต่อไป โดยที่ Main จะไม่ต้องไปรับรู้ว่า Proc1 และ Proc2 จะต้องดำเนินการอย่างไร ตามหลักการที่อธิบายมานี้เราจึงนิยามว่า Proc1 และ Proc2 ว่าเป็น PAR การดำเนินการตามลักษณะของ PAR นี้จะเปรียบได้ว่า Main นั้นจะทำหน้าที่เหมือนพ่อครัวใหญ่ที่ทำหน้าที่ปรุงอาหาร โดยที่จะมีลูกมือย่อยมาทำหน้าที่ช่วยจัดการทำความสะอาด ล้างวัตถุดิบให้ ใช้ โดยที่พ่อครัวใหญ่ไม่จำเป็นจะต้องไปรับรู้ว่าลูกมือนั้นไปจัดการอย่างไร

คุณประโยชน์ของการสร้างคลังโปรแกรม ย่อย(Library Module)

โปรแกรมย่อยที่สร้างขึ้นมานั้นนับว่ามีประโยชน์ต่อผู้สร้างและพัฒนามาก เพราะเราสามารถนำกลับมาพัฒนาใช้ใหม่ได้ ดังนั้นจึงมีการเก็บรวบรวมไว้ในคลังโปรแกรมย่อย ซึ่งก่อให้เกิดประโยชน์หลายประการด้วยกันคือ

- ใช้ทรัพยากรร่วมกันเป็นการประหยัด
- อำนวยความสะดวกในการเรียกใช้โมดุลบางอย่างที่ใช้บ่อยๆ
- เป็นการนำทรัพยากรที่ใช้แล้วกลับมาใช้ใหม่ (Reuse)
- โมดุลที่สร้างเก็บไว้ในคลังนั้นผู้ใช้จะค่อนข้างแน่ใจได้ว่าปลอดจากความผิดพลาด (Error Free) ดังนั้นจึงใช้ได้ยังสะบายใจไม่ต้องกังวล ไม่เหมือนโปรแกรมที่เขียนขึ้นมาใหม่ซึ่งจะต้องนำไปทดสอบก่อนให้แน่ใจ ซึ่งจะเสียเวลา
- ปกติผู้เขียนโปรแกรมระดับมืออาชีพ มักจะนำโปรแกรมที่เขียนขึ้นมาและผ่านการทดสอบแล้ว เก็บไว้ที่คลังโปรแกรมเพื่อจะได้นำกลับมาใช้ใหม่ได้ ตัวอย่างเช่น โปรแกรมแปลง ตัวอักขระจาก Lowercase มาเป็น Uppercase เป็นต้น
- ในบางภาษาเช่นภาษาซีจะสามารถกำหนดทางเลือกในการเชื่อมโยงการติดต่อกับคลังโปรแกรมเหล่านี้โดยการ Include เข้ามาในส่วนของ Preprocessor เพื่อจะได้ดึงโปรแกรมย่อยที่จะใช้นั้นมาเชื่อมโยงเข้าในช่วงของการ Compile เป็นต้น แต่ลักษณะนี้จะยังไม่จัดเป็น Standard ของภาษาคอมพิวเตอร์โดยทั่วไป

ตัวอย่างการใช้ประโยชน์จากโปรแกรมย่อยในคลังสำหรับภาษาปาสคาล

```
NewChar := Upcase(NextChar);
```

การทำนามธรรมข้อมูล (Data Abstraction :ADT)

กระบวนการประมวลผลข้อมูลด้วยคอมพิวเตอร์นั้นพื้นฐานที่สำคัญคือข้อมูล ซึ่งข้อมูลอาจจะเป็นวัน เดือน ปีเกิด บ้านเลขที่ หรือสัญญาณตรรกะ หรืออาจจะเป็นประวัติของสินค้าที่เราดูแลอยู่เป็นต้น ภาระกิจในการจัดการข้อมูลนั้นผู้เขียนโปรแกรมจะต้องนิยามข้อมูลทั้งหลายให้มีรูปแบบตามความต้องการของผู้ใช้ด้วยเหตุนี้จึงมีการสร้างโปรแกรมย่อยเฉพาะกิจในเรื่องการจัดการข้อมูลขึ้นมาให้ใช้โดยเรียกโปรแกรมย่อยดังกล่าวว่า การทำนามธรรมข้อมูล (Data Abstraction :ADT)

Abstract Data Type = The combination of a data type and procedures for processing that type

ตัวอย่างโปรแกรมที่เป็นลักษณะของ ADT

```
(
    Specification for abstract data type DayADT
    Structure : Day is an enumerated data type whose values represent the days of
    the week.
Operators:
ReadLnDay (var ADay :Day) :reads two data characters and stores the value
represented by the data in ADay.
Implementation
)
type .
  Day = (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday):
  Function Upcase (Ch :Char) : Char ;
    (Returns the uppercase form of its argument.
     Pre : none.
     Post : returns the uppercase equivalent of Ch if Ch is a lowercase letter :
          otherwise, returns Ch.
     )
  begin (Upcase)
     if (Ch >='a') and (Ch ,='z') then
```

```
            Upcase := Chr(Ord(Ch) - Ord('a') + Ord ('A'))
        else
            Upcase := Ch
    end ;(Upcas)

procedure ReadLnday(var ADay (output) :day):
  (
     Reads a value into ADay .
     Pre : None
     Post : ADay is assigned a value when the two characters read are SU, MO, TU,
      WE, TH ,FR, or SA:
      Value of ADay corresponds to data characters.
     Calls :Upcase
  )
    var
     DayChl ,
     DayCh2 :Char ;
     ValidDay : Boolean;
Begin
     Repeat
          Write('Enter the first two letters of the day name >');
          Readln(DayCh1, dayCh2);
          DayChl := Upcase(DayCh1) ;
          DayCh2 := Upcase(DayCh2) ;
          ValidDay := True ;
          If (DayChl = 'S') and (DayCh2 ='U') Then Aday := Sunday
          Else if (DayChl = 'M') and (DayCh2 ='O') Then Aday := Munday
          Else if (DayChl = 'T') and (DayCh2 ='U') Then Aday := Tuesday
          Else if (DayChl = 'W') and (DayCh2 ='E') Then Aday :=Wednesday
          Else if (DayChl = 'T') and (DayCh2 ='H') Then Aday := Thursday
          Else if (DayChl = 'F') and (DayCh2 ='R') Then Aday := Friday
          Else if (DayChl = 'S') and (DayCh2 ='A') Then Aday := Saturday
```

```
                Else
                ValidDay := False
        Until ValiDay
End ;

Procedure Writeday (Aday (input) :Day) :
(
display the vlaue of Aday
        Pre :Aday is defined,
        Post : Displays a string corresponding to the value of Aday
)
begin
        case Aday of
                Sunday  : Write ('Sunday') ;
                Monday  : Write ('Monday') ;
                Tuesday : Write ('Tuesday') ;
                Wednesday : Write ('Wednesday') ;
                Thursday  : Write ('Thursday') ;
                Friday : Write ('Friday') ;
                Saturday  : Write ('Saturday') ;
        End
End ;
```

ประโยชน์ที่ได้จาก ADT ก็คือการนำไปใช้งานเราจะได้ Actual Code ที่นำไปใช้เพื่อให้ได้ประเภทของข้อมูลที่ต้องการด้วย

การนำ ADT ไปใช้งาน ในModule Libraries นอกจากจะมีโปรแกรมย่อยๆที่ใช้งานบ่อยๆ ปรากฏแล้วยังมี Module ประเภท ADT Procedure ร่วมอยู่ด้วย ตัวอย่างเช่นเราสร้าง ADT มีชื่อว่า DayADT ไว้เป็นแฟ้มโปรแกรมภายใต้ชื่อว่า DayADT.PAS และเมื่อมีโปรแกรมใดที่ต้องการจะเรียกใช้ DayADT เราก็จะทำการอ่านแฟ้มดังกล่าวจากสื่อที่บันทึกโปรแกรมดังกล่าวเพื่อนำมา Compile ร่วมกับโปรแกรมที่เป็นผู้เรียก วิธีการนี้จะเรียกว่า Compiler Directives ที่จะปรากฏเป็น Header Line บนหัวของโปรแกรมที่เป็นผู้เรียกใช้ DayADT ดังนี้

% Include 'DayADT.PAS'

คุณลักษณะของ Compiler Directives (ไม่จัดว่าเป็น Standard) ก็คือมีหน้าที่รวมเอาแฟ้มโปรแกรมที่อ้างไว้ตามคำสั่ง Include มาผนวกเข้าไว้ด้วยกันก่อนกับโปรแกรมที่เรียก แล้วจึงนำไป Compile พร้อมกัน แต่สำหรับ Pascal บางตัวนั้นเราสามารถแยก Compile ออกเป็นเอกเทศจากโปรแกรมที่เรียก และภายหลังจากการ Compile แล้วจึงนำมา Link เข้าด้วยกัน ผ่านโปรแกรม Linker

การทดสอบโปรแกรมขนาดใหญ่ (Testing Large Program)

การทดสอบโปรแกรมเดี่ยวกับการทดสอบระบบโปรแกรมที่มีโปรแกรมย่อยๆหลายๆตัวเชื่อมต่อกันนั้น แต่สำหรับระบบโปรแกรมขนาดใหญ่จะมีภาระกิจเพิ่มมากว่าการทดสอบโปรแกรมเดี่ยวในส่วนที่จะต้องทำการทดสอบการเชื่อต่อระหว่างโปรแกรมย่อย (Software Interface) ว่าสามารถเชื่อมต่อกันได้อย่างถูกต้องหรือไม่

การเตรียมแผนงานในการทดสอบ (Preparing a test Plan)

การเตรรียมแผนงานในการทดสอบสำหรับโปรแกรมขนาดใหญ่เป็นสิ่งที่มีความจำเป็นอย่างยิ่งเพื่อที่จะได้ลำดับและเตรียมความพร้อมสำหรับการทดสอบ

ขั้นตอนในการวางแผนในการทดสอบจะปรากฏดังนี้

- กำหนดวิธีการทดสอบ
- กำหนดแผนการดำเนินงานในแต่ละขั้นตอน
- กำหนดกลุ่มบุคลากรที่จะทำการทดสอบ (โดยปกติผู้ที่เขียนโปรแกรมนั้นเรามักจะไม่ให้ร่วมในการเป็นทีมงานทดสอบด้วย)อันประกอบด้วย ผู้ใช้โปรแกรม นักวิเคราะห์ระบบ หัวหน้าทีมผู้สร้างระบบโปรแกรม เป็นต้น

ถึงแม้ว่าการทดสอบโปรแกรมนั้นจะเกิดขึ้นในขั้นสุดท้ายของวัฏจักรของการสร้างซอฟต์แวร์ก็ตาม แต่ความเป็นจริงแล้ว การดำเนินงานเกือบทุกขั้นตอนมักจะต้องระลึกถึงอยู่เสมอในเพื่อที่จะเอื้ออำนวยให้ขั้นของการทดสอบของ SLCเป็นไปด้วยความราบรื่น กรณีที่เราดำเนินการดังกล่าวแล้วนั้นจะส่งผลให้ Syntax Error และ Logic Error เกือบจะไม่ค่อยปรากฏ แต่จะมีปรากฏเฉพาะส่วนของ Run Time Error เท่านั้น

ข้อสังเกตุของการเขียนโปรแกรมนั้นเรามักจะสร้างความสามารถของการปกป้องตนเอง (Defensive Programming) ซึ่งหมายถึงความสามารถในการตรวจจับข้อมูลที่ผิดพลาดที่อาจจะหลงเข้ามาในระบบได้ ดังตัวอย่างคำสั่งต่อไปนี้

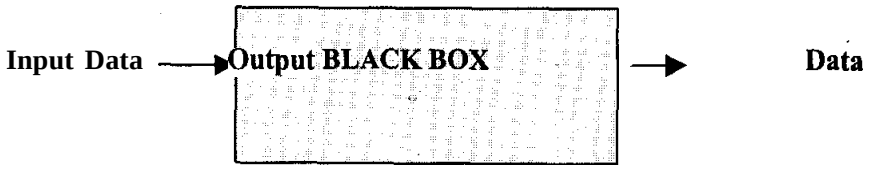
```
if ( N < = 0) Then
        Writeln('Invalid Value for parameter N' , N:l ) ;
```

Structure Walkthrough เป็นเทคนิควิธีหนึ่งที่ใช้กันอย่างแพร่หลายในการตรวจสอบระบบรวมทั้งนำมาใช้ในการทดสอบระบบโปรแกรม โดยที่ในช่วงของการทำ Walkthrough นั้น ทีมงานจะพยายามค้นหาความผิดพลาด (Error or Bug) ซึ่งยังมีปรากฏในโปรแกรมนั้นอันเกิดจากการมองข้ามไปของผู้เขียนโปรแกรม

Black Box versus White Box Testing

การทดสอบโมดุลย่อยๆในระบบโปรแกรมนั้นสามารถดำเนินการได้ทั้ง 2 วิธี ด้วยกันคือ

1. **Black Box Testing (Specification Based Testing)** เป็นวิธีการทดสอบที่ตั้งอยู่บนพื้นฐานที่ว่าจะไม่รับรู้เกี่ยวกับการใช้คำสั่งใดๆที่ปรากฏในโมดุลย่อย โดยจะทำการทดสอบว่าโมดุลนั้นๆสามารถดำเนินการสนองตอบตามรายการที่แจ้งไว้ (Specification) หรือไม่ ดังนั้นการทดสอบจึงต้องดำเนินการสร้างข้อมูลขึ้นมาให้ครบทุกกรณีที่จะเป็นไปได้ในทางเลือกของการดำเนินงานของโปรแกรม การสร้างข้อมูลขึ้นมาทดสอบนั้นถ้าเป็นงานลักษณะทั่วๆไปทางธุรกิจจะไม่ยุ่งยากเท่าไรและมีความสามารถที่จะสร้างข้อมูลได้ครบถ้วนทุกทางที่เป็นไปได้ แต่ถ้าเป็นกรณีของงานทางคณิตศาสตร์นั้นค่อนข้างจะมีปัญหาในแง่ที่ว่าเราไม่อาจจะทราบได้ว่ามีข้อมูลทั้งหมดที่เป็นไปได้คืออะไร เช่นการเขียนโปรแกรมย่อยเพื่อหา Determinant ของ Matrices เป็นต้น สภาวะที่ผู้ทดสอบสามารถทราบข้อมูลที่เป็นไปได้ทั้งหมด เราจะเรียกว่า System Boundaries



ภาพที่ 13.2 กรรมวิธีของ BLACK BOX

2. **White Box Testing** เป็นวิธีการทดสอบที่ตกแต่งกับวิธีแรกโดยสิ้นเชิง เพราะทีมงานที่ทดสอบจะศึกษาทุกคำสั่งที่เขียนในโมดุลย่อยต่างๆ และทำการสอบกับคำสั่งย่อยๆ เหล่านั้นเพื่อดูว่ามีข้อผิดพลาด และสามารถปฏิบัติงานตามที่ต้องการหรือไม่ การดำเนินงานเช่นนี้ จะต้องทราบถึงสาระขอบเขตของระบบที่เรียกว่า System Boundaries ของตัวแปรทุกตัวในโมดุลด้วย

Integration Testing เป็นนำแผนการทดสอบต่างๆคือ Top Down Testing และ Bottom Up Testing มาใช้ร่วมกัน วิธีการของ Integration Testing นั้นผู้ทดสอบจะเป็นผู้จำแนกว่าในระบบโปรแกรมนั้นส่วนใดจะทำการทดสอบแบบ Top Down Testing (Testing System) และส่วนใดจะต้องทำการทดสอบแบบ Bottom Up testing ภายหลังการทดสอบเสร็จสิ้นลงจึงจะมีการทดสอบทั้งระบบ (Whole System)

Formal Methods of Program Verification

Formal Methods เป็นทางเลือกอีกทางหนึ่งที่พัฒนาขึ้นในสายงานทางวิทยาการคอมพิวเตอร์ การทดสอบวิธีนี้จะสร้างกฎในการทดสอบ (Formal Rules) เพื่อทดสอบว่าโปรแกรมที่สร้างขึ้นมานั้นสามารถทำงานตรงตามวัตถุประสงค์หรือไม่ แนวทางเช่นนี้จะคล้ายกับการพิสูจน์ทฤษฎีทางคณิตศาสตร์ โดยมีองค์ประกอบคือ Definition ,Axioms และ Theory ที่เคยพิสูจน์มาก่อนหน้านี้แล้ว ในส่วนที่จะกล่าวถึงในตอนนี้จะสรุปเฉพาะแนวคิดในการดำเนินงานตามวิธีของ Formal Method จะประกอบด้วย

- Assertion
- Loop Invariant

Assertion จะหมายถึงกรรมวิธีการตรวจสอบ Logical Statement ในโปรแกรม ซึ่งยอมรับว่าเป็นจริงเสมอ (Assertion) คำสั่งใดที่มีลักษณะเช่นนี้จะเขียน Comment กำกับไว้ด้วย ดังตัวอย่างต่อไปนี้

```
A := 5 ; {assert : A is 5}
X := A ; {assert : X is 5}
Y := X + A ; {assert : Y is 10}
```

Assertion แรก คือ A = 5 และลำดับถัดมาคือ Assertion ที่สอง X จะมีค่าเท่ากับ 5 ด้วย ส่วน Assertion สามก็จะเป็นไปตามผลของสอง Assertion ที่เกิดก่อนหน้านี้ นั่นคือเมื่อ A = 5 และ X = 5 จะยังผลให้ Assertion ที่สามคือ Y มีค่าเป็น 10 จากทั้งสาม Assertion นี้จะสรุปผลได้ว่า การเปลี่ยนค่าของ A จะส่งผลให้ค่าของ X และ Y เปลี่ยนแปลงไปด้วย

วัตถุประสงค์ของ Formal Verification ก็เพื่อจะพิสูจน์ว่า การแตกย่อยของโปรแกรมเป็นส่วนๆนั้นจะยังให้ผลตรงตามวัตถุประสงค์หรือไม่ ดังตัวอย่างที่แสดงมาให้ดูนี้เราจัดว่า A เป็น Precondition

การแตกโปรแกรมเป็นส่วนย่อย (Program Fragment) แล้วนั้น คำสั่งที่มีลักษณะเป็น **Assignment Rule** จะเป็นคำสั่งที่มีผลมากที่สุดในการทำงานของโปรแกรม ดังเช่นคำสั่งที่ กำหนดให้ A มีค่าเป็น 5 นั้นจะจัดเป็น Assignment Rule เราอาจจะเขียนโครงสร้างของโปรแกรมทั้งสามคำสั่งได้ดังนี้

(P(A)) ;

X := A ;

(P(X)) ;

ซึ่งจะอธิบายได้ดังนี้

If P(A) is a logical statement (assertion) about A , the same statement will be true of X after the assignment statement X := A execution .

เจตนาของเราก็คือ ต้องการที่จะใช้ Assertion ให้เป็นรูปแบบของ Document Tool เพื่อใช้พิสูจน์ความเข้าใจในวิถีทางของการเขียนโปรแกรมแทนที่จะใช้วิธีการของ Formally Proving เพื่อจะได้แก้ไขความผิดพลาดของการเขียนโปรแกรม การเขียน Assertion โดยการใช้รูปแบบภาษาอังกฤษไม่จำเป็นต้องใช้รูปแบบของ Formal Language ในการเขียน นอกจากนี้เรายังสามารถใช้ Assertion ในการเขียนเอกสารอธิบายถึงผลที่เกิดจากการทำงานของโปรแกรมย่อย (Procedure) ได้ด้วย

Preconditions and Postconditions Procedure ' s Precondition จะหมายถึง Logical Statement ที่เกี่ยวข้องกับ Input Parameters ส่วน Procedure Postconditions จะหมายถึง ถึง Logical Statement ที่เกี่ยวข้องกับ Output Parameters หรืออาจจะหมายถึง Logical Statement ที่อธิบายถึงการเปลี่ยนแปลงที่เกิดใน Program State ซึ่งเกิดจากการปฏิบัติงานในโปรแกรมย่อย (Procedure Execution) โดยที่กิจกรรมที่เปลี่ยนแปลงในโปรแกรม เช่นการเปลี่ยนแปลงมูลค่าของตัวแปร หรือการสั่ง ให้แสดงผล Output หรือการรับข้อมูลเข้าสู่ระบบ

ตัวอย่างต่อไปนี้จะแสดง Precondition และ Postcondition สำหรับ Procedure EnterInt

```
Procedure EnterInt(MinN, MaxN,input) : Integer ;
                        Var N (output) : Integer ):
(
   Reads an integer between MinN and MaxN into N.
   Pre : MinN and MaxN are assigned values
```

Post : Returns in N the first data value between MinN and MaxN if

MinN <= MaxN is true : otherwise, N is not defined

)

จากตัวอย่างที่แสดงมานี้เห็นได้ว่า Precondition จะระบุว่า Input Parameter MinN และ MaxN เพื่อเป็นการแจ้งลักษณะของตัวแปรที่จะนำข้อมูลเข้าไปปฏิบัติการส่วน Postcondition จะบ่งว่า ภายหลังเมื่อโปรแกรมปฏิบัติงานเสร็จค่าของข้อมูลระหว่าง MinN จะถูกกำหนดเป็นค่าของ N เป็น Output Parameter ถ้าหากว่า MinN น้อยกว่าค่าของ MaxN

Loop Invariants โปรแกรมที่มีส่วนของการใช้ Loop นั้นจะมีข้อยุ่งยากในการตัดสินว่าสิ่งที่จะดำเนินการภายใน Body ของ Loop นั้นถูกต้องหรือไม่ อาทิเช่นการเปลี่ยนแปลงมูลค่าของตัวแปรภายใน Loop ที่เกิดจากการวน Loop แต่ละรอบ Assertion พิเศษที่เกิดขึ้นจากการทำงานของ Loop จะเรียกว่า Loop Invariant

ลักษณะของ Loop Invariant ก็คือ Logical Statement ที่เกี่ยวเนื่องกับตัวแปรต่างๆที่ปรากฏภายใน Loop ที่เป็นจริงก่อนที่จะเข้าไปวนใน Loop และค่าของตัวแปรดังกล่าวที่เกิดขึ้นภายหลังในการทำงานของแต่ละ Loop จะต้องยังจริงอยู่ด้วย ลักษณะดังกล่าวจะเรียกว่าคุณสมบัติของการไม่เปลี่ยนแปลง (Invariant)

ตัวอย่างต่อไปนี้จะแสดงลักษณะของ Loop Invariant ของโปรแกรมที่คำนวณหาค่าผลบวกของ 1 + 2+ 3+...+ N โดยที่ N คือเลขจำนวนเต็มบวกและ SUM , I ก็เป็นเลขจำนวนเต็มบวกด้วย

(

Accumulate the sum of integers I through N in SUM .

Assert : N >= 1

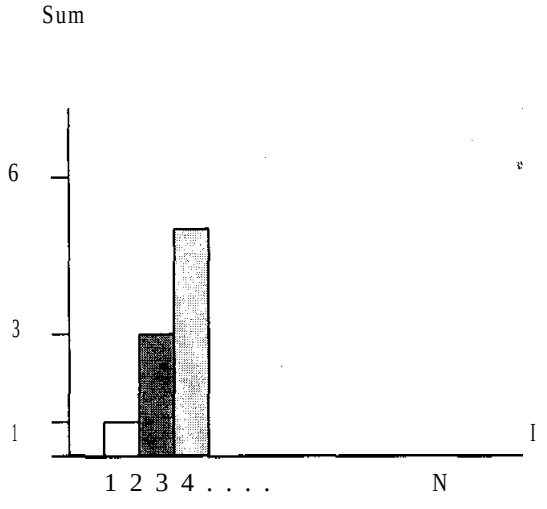
```
Sum := 0 ;
I : = 1;

While (I <=N) do
        Begin
                Sum := Sum + I ;
                I := I+1
        End ; (While)
```

(asserte : Sum is I + 2 + 3 + . ..t +N-1 t N)

Assertion แรก (N >=1) ก็คือ Precondition ของ Loop และ Assertion สุดท้ายคือ *Postcondition* คือ {Sum is 1 + 2 + 3 + ...+ N-1 + N} จากโปรแกรมนี้ Loop Invariant จะเป็นจริงก่อนที่ Loop นั้นจะดำเนินการปฏิบัติงานตามคำสั่งใน Loop การวนเวียนทำงานนั้นจะมีการตรวจสอบตัวแปรที่ควบคุม Loop โดยที่ loop Invariant ก็จะคงมีสภาพเป็น Logical Statement ภาพแสดงการหาค่า Sum ที่ Loop เมื่อ I มีค่าเป็น 4 Sum = 1=2+3 = 6 (The sum of all integer less than 4) เงื่อนไขนี้จะยังคงเป็นจริง เมื่อ I = N+1 แล้ว Sum = 1+2+3+...+N เราจะเขียนเป็น Invariant ได้ดังนี้

(Invariant : I < = N+1 and Sum is 1+2+3+..+N)

ภาพที่ 13.3 Sketch of Summation Loop



สรุปได้ว่า Assertion ของ Loop Invariant นี้ได้ความว่า การทำซ้ำนั้น I จะมีค่าน้อยกว่าหรือเท่ากับ N+1 โดยที่ Sum จะเป็นผลบวกของเลขจำนวนเต็มซึ่งมีค่าน้อยกว่า I เงื่อนไขที่เขียนนี้อาจจะอ่านแล้วสงสัยว่า ส่วนแรกของ Loop Invariant ที่เขียนนั้นเขียนว่า I <= N+1 ทำไมถึงไม่เขียนเป็น I <=N เหตุผลที่เป็นเช่นนี้ก็เพราะว่า loop Invariant นี้จะเป็นจริงหลังจากการทำซ้ำครั้งสุดท้ายสิ้นสุดลง ดังนั้นถ้าคำสั่งสุดท้ายยังเป็นคำสั่งที่ปรากฏภายใน Loop (loop body) เพื่อที่จะขยับค่า I ให้เพิ่มขึ้น 1 และการทำงานของ Loop While ก็จะสิ้นสุดลงออกจาก Loop เมื่อ ค่า N+1 ลักษณะของ Loop Invariant หมายความว่าก่อนที่จะทำงานใน Loop ค่าของ I จะเป็น 1 และ I <= N+1 จะเป็นจริงต่อเมื่อ N >= 1 (เงื่อนไขของ Precondition)

โปรแกรมเมอร์มักจะใช้ Loop Invariant เป็นเครื่องมือช่วยในการตรวจสอบก่อนที่จะนำไปเขียนโปรแกรม (Coding) ต่อไป สำหรับโปรแกรมที่มีการใช้ Loop ร่วมด้วย เพราะ Loop Invariantจะช่วยในส่วนของการนำทาง การกำหนดค่าเริ่มต้น และการกำหนดเงื่อนไขการทำงานซ้ำๆ ตลอดจนการกำหนดวิธีการทำงานภายใน Loop Body ดังตัวอย่างต่อไปนี้

(Invariant :

Count < N and Sum is the Sum of all data read so far

จากลักษณะของ loop Invariant เราจะแบ่งออกมาเป็นส่วนๆดังนี้คือ

- The loop initialization is

  sum :=0.0 ;

  Count : = 0;

- The loop repetition test is

  Count < N

- The loop body is

  Read (Next) ;

  Sum := Sum + Next ;

  Count := count + 1 ;

จาก Information ที่กำหนดมานี้จะเห็นว่าช่วยในการเขียนโปรแกรมได้ง่ายขึ้น

Invariant กับคำสั่ง For จากที่ได้อธิบายมาแล้วเกี่ยวกับความหมายของ Loop Invariant ว่า เราจะต้องทราบข้อเท็จจริงก่อนที่จะเข้า loop และหลังจากการทำงานเสร็จสิ้นลงในแต่ละครั้งของการทำงานใน Loop เราก็สามารถที่จะเขียน Invariant ของคำสั่ง For ได้เช่นเดียวกับการเขียนในคำสั่ง While การเขียน Loop Invariant Document ของ For Loop เราระทำได้ดังนี้

(assert N >=1)

Sum :=0 ;

For I := 1 TO N do

(Invariant ; I <= N+1 and Sum is 1+2+3+..+ N

Sum ;= Sum + I ;

(assert ; Sum is 1+2+3+..+N-1+N)

ตัวอย่างต่อไปนี้จะแสดงวิธีการของ Sentinel – Controlled ของ While loop โดยที่การทำงานจะออกจาก Loop หลังจากที่ Sentinel Value ถูกอ่านเข้ามา Loop Invariant ที่ปรากฏนี้จะชี้แนะถึงการหาค่า Product ของข้อมูลที่รับเข้ามา

ตัวอย่างการใช้ Sentinel Loop with Invariant

```
( Compute the product of a sequence of data values ,
        assert : Sentinel is a constant.
)
```

ตัวอย่างโปรแกรมต่อไปนี้จะใช้เทคนิคของ Sentinel – Controlled While Loop โดยที่การทำงานจะออกจาก Loop หลัง หลังจากที่ Sentinel Value จะถูกอ่านเข้ามา Loop Invariant ที่ปรากฏนี้จะชี้แนะถึงกาหาค่า Product ของข้อมูลที่รับเข้ามา

```
(
        Compute the product of sequence of data values,
        Assert : Sentinel is a constant.
)
Product := 1;
Writeln ('When done ,enter a sequence of data values .
Writeln ('Enter the first number >') ;
Readln (Num) ;
While Num <> Sentinel do
        ( invariant :
        Product is the product of all prior values read into Num and no prior value of
        Num and no prior value of Num was the sentinel
        )
        begin
                Product := Product + Num ;
                Writeln ('Enter the next number >') ;
                Readln (Num)
        End ;
        (assert :
                Product is the product of all numbers
                Read into Num before the sentinel. )
```

จริยธรรมและความรับผิดชอบของผู้ทำงานในสายวิชาชีพคอมพิวเตอร์

วิชาชีพต่างๆในทางคอมพิวเตอร์ ไม่ว่าจะเป็นสายงานทางด้านวิศวกรรมซอฟต์แวร์ หรือโปรแกรมเมอร์ หรือ นักวิเคราะห์ระบบ หรือผู้ใช้ในระบบ ก็สมควรที่จะมีจริยธรรมในงานที่ทำเช่นเดียวกับบุคลากรในสายงานอื่นเช่นแพทย์ ทนายความ วิศวกร คือจะต้องมีความซื่อสัตย์ต่องานอาชีพ เช่นจะต้องไม่นำเอาสารสนเทศที่เป็นความลับขององค์กรไม่ว่าจะเป็นระดับ Top Secret หรือ Secrete ไปดำเนินการเปิดเผยเพื่อหาผลประโยชน์เข้าตน รวมทั้งการก่ออาชญากรรมอื่นๆโดยนำเอาระบบคอมพิวเตอร์เป็นเครื่องมือ เช่นการเปิดเผยข้อมูลของลูกค้าให้กับบริษัทที่เป็นคู่แข่งขันทางการค้า การทุจริตในการยักยอกโอนเงินของลูกค้าธนาคารมาเป็นของตน การแก้ไขแฟ้มข้อมูลในทางทุจริต ตัวอย่างเช่นที่พบกันบ่อยในระบบคอมพิวเตอร์คือการขโมยข้อมูลที่เรียกว่า Computer Hackers หรือพฤติกรรมการแพร่ไวรัสออกไปทำลายระบบคอมพิวเตอร์ หรือการทำสำเนาโปรแกรมซอฟต์แวร์ที่มีลิขสิทธิ์ออกแจกจ่ายใช้โดยมิชอบ พฤติกรรมที่กล่าวมานี้เป็นเพียงตัวอย่างส่วนหนึ่งที่เกิดขึ้นจากแวดวงของผู้ประกอบอาชีพที่เกี่ยวข้องกับคอมพิวเตอร์

สาระสำคัญโดยสรุปสำหรับความผิดพลาดที่เกิดจากในโปรแกรม

(Common Programming Errors)

ความผิดพลาดที่เกิดขึ้นในโปรแกรมมักจะมีสาเหตุต่อไปนี้

- การเชื่อมต่อของโปรแกรมย่อยผิดพลาด (Sub System Interface)
- การส่งพารามิเตอร์ระหว่าโปรแกรมย่อยผิดพลาดเช่น การเรียงลำดับผิดพลาด การกำหนดประเภทของ Formal Parameter ผิดพลาด
- การกำหนด ข้อมูลในส่วนของ Precondition ของโปรแกรมย่อยผิดพลาด
- การใช้ ADT ผิดพลาด ตัวอย่างเช่นบนระบบ Turbo Pascal จะสร้าง ADT เก็บไว้บน Unit ในขณะที่ VAX Pascal จะเก็บไว้ใน Module ดังนั้นการใช้Client Program เพื่อเรียกโปรแกรมย่อยประเภท ADT จึงต้องระลึกอยู่ว่าโปรแกรมย่อย ADT ของระบบนั้นเป็นรูปแบบใด มิฉะนั้นจะเกิดความผิดพลาดในการนำไปใช้งาน

คำถามท้ายบท

1.จงอธิบายความหมายของคำต่อไปนี้

- Testing and Verification
- Loop Invariant
- Black Box Testing
- Library Procedure

2. จงเขียนโปรแกรมย่อย (Procedure) เพื่อหาค่าเฉลี่ยของจำนวนอักขระแต่ละตัว (A-Z) ที่นับได้จากการป้อนข้อความใดที่รับจากแป้นพิมพ์ โดยที่ข้อความดังกล่าวจะยาวไม่เกิน 60 ตัวอักขระ

3. ขั้นตอนใดในการสร้างโปรแกรมที่เกี่ยวข้องกับผู้ใช้และนักวิเคราะห์ระบบมากที่สุดในกระบวนการสร้างซอฟต์แวร์

4. จงอธิบายปัญหาที่เกิดจาก Procedure Interface มากที่สุดเมื่อเราใช้ White Box Testing

5. จงอภิปรายถึงข้อดีและข้อเสียของการใช้ Black Box Testing

6. จากส่วนหนึ่งของโปรแกรมต่อไปนี้

```
Product : =1;
Counter :=2 ;
While Counter < =5 do
        Begin
                Product := Product + Counter ;
                Counter := Counter + 1
        End:
```

ให้ดำเนินการเขียนFormal Method กับ Loop Invariant