ชื่อเรื่อง **เกณฑ์การประเมินความครอบคลุมของการทดสอบซอฟต์แวร์แบบโครงสร้าง**

Title **Structural Test Coverage Criteria of Software**

ชื่อผู้เขียน ดร. ภาณุชาติ บุณยเกียรติ
คณะวิทยาศาสตร์และเทคโนโลยี มหาวิทยาลัยหอการค้าไทย
E-Mail: panuchart_Bun@utcc.ac.th
นางสาวศศิธร มงคลศรีพัฒนา
ศูนย์วิจัยมหาวิทยาลัยชิคาโก-มหาวิทยาลัยหอการค้าไทย

บทคัดย่อ

การเพิ่มความเชื่อมั่นว่าซอฟต์แวร์นั้นมีข้อผิดพลาดน้อยที่สุด วิศวกรซอฟต์แวร์จะต้องทำการทดสอบการทำงานของซอฟต์แวร์ให้ครอบคลุมมากที่สุด บทความนี้กล่าวถึงเกณฑ์การประเมินความครอบคลุมในการทดสอบซอฟต์แวร์ด้วยวิธีการทดสอบโครงสร้างของโปรแกรม (Structural Testing) หรือที่เรียกกันว่า การทดสอบแบบกล่องขาว (White-box Testing) โดยทำการเปรียบเทียบความครอบคลุมของการทดสอบซอฟต์แวร์ออกเป็น 6 ระดับ ประกอบด้วย 1.ระดับครอบคลุมคำสั่ง (Statement Coverage) 2.ระดับครอบคลุมการตัดสินใจ (Decision Coverage) 3.ระดับครอบคลุมเส้นทางการทำงาน (Path Coverage) 4.ระดับครอบคลุมเงื่อนไขและการตัดสินใจ (Condition/Decision Coverage) 5.ระดับครอบคลุมเงื่อนไขแบบหลายกรณี (Multiple Condition Coverage) และ 6.ระดับครอบคลุมเงื่อนไขแบบแก้ไข (Modified Condition/Decision Coverage) จากนั้นจะกล่าวถึงประเด็นปัญหาที่อยู่ในความสนใจของนักวิจัยทางด้านวิศวกรรมซอฟต์แวร์ เกี่ยวกับการประเมินความครอบคลุมในการทดสอบซอฟต์แวร์ ซึ่งจะได้นำไปเป็นแนวทางในการวิจัยต่อไปในอนาคต

คำสำคัญ : การทดสอบซอฟต์แวร์ ความครอบคลุมของการทดสอบซอฟต์แวร์

Abstract

Software engineers can ensure that a piece of software is tested properly by monitoring test coverage of program. This article examines six test coverage criteria of white-box, structural testing including Statement Coverage (SC), Decision Coverage (DC), Path Coverage (PC), Condition/Decision Coverage (CDC), Multiple Condition Coverage (MCC) and Modified Condition/Decision Coverage (MC/DC). This article then discusses issues related to test coverage criteria, particularly, to MC/DC and identifies open problems and current lines of research in test coverage analysis.

Keywords: Software Testing, Test Coverage Criteria

บทนำ

Dijkstra (1969) ได้อธิบายเกี่ยวกับการทดสอบซอฟต์แวร์ซึ่งได้รับการยอมรับโดยทั่วกันไว้ว่า “เราสามารถที่จะแสดงถึงการมีอยู่ของข้อผิดพลาดในซอฟต์แวร์ แต่ไม่สามารถที่จะแสดงว่าซอฟต์แวร์นั้นปราศจากข้อผิดพลาดได้” หนทางหนึ่งซึ่งวิศวกรซอฟต์แวร์สามารถเพิ่มความเชื่อมั่นว่าซอฟต์แวร์ที่ได้ทำการทดสอบนั้น มีข้อผิดพลาดน้อยที่สุดคือ การทำการทดสอบให้ครอบคลุมการทำงานของซอฟต์แวร์ให้มากที่สุด ซึ่งย่อมจะเพิ่มความเป็นไปได้ที่จะตรวจพบข้อผิดพลาดในซอฟต์แวร์นั้นๆ ให้มากขึ้นตามไปด้วย

บทความนี้กล่าวถึงเกณฑ์การประเมินความครอบคลุมในการทดสอบซอฟต์แวร์ด้วยวิธีการทดสอบโครงสร้างของโปรแกรม (Structural Testing) หรือที่เรียกว่า การทดสอบแบบกล่องขาว (White-box Testing) แนวคิดพื้นฐานของการทดสอบแบบนี้ คือการที่ความผิดพลาดของซอฟต์แวร์จะถูกตรวจพบได้นั้น เราต้องทำการทดลองเรียกใช้คำสั่งต่างๆ ในโปรแกรมให้มากที่สุด การทดสอบโครงสร้างของโปรแกรมเช่นนี้มีความสำคัญมาก เนื่องจากจะสามารถช่วยวิศวกรซอฟต์แวร์ในการวิเคราะห์การทำงานของโปรแกรมเพื่อค้นหาข้อบกพร่องที่อาจจะเกิดขึ้นจากการทำงานของคำสั่งต่างๆ ในโปรแกรมได้ อย่างไรก็ดี เนื่องจากการทดสอบด้วยวิธีนี้ไม่ได้พิจารณาข้อกำหนดความต้องการ (Requirement Specification) ของซอฟต์แวร์ เราจึงไม่สามารถระบุความสัมพันธ์ระหว่างระดับความครอบคลุมในการทดสอบโครงสร้างของโปรแกรมกับคุณภาพของซอฟต์แวร์เมื่อพิจารณาจากข้อกำหนดความต้องการได้ (Page และคนอื่นๆ, 2009 น. 116-117)

การทดสอบโครงสร้างของโปรแกรมสามารถแบ่งออกเป็น 6 ระดับ ตามระดับของความเข้มข้น หรือที่เรียกว่า ความครอบคลุมรหัสโปรแกรม (Code Coverage) ประกอบด้วย 1.ระดับครอบคลุมคำสั่ง (Statement Coverage - SC) 2.ระดับครอบคลุมการตัดสินใจ (Decision Coverage - DC) 3.ระดับครอบคลุมเส้นทางการทำงาน (Path Coverage - PC) 4.ระดับครอบคลุมเงื่อนไขและการตัดสินใจ (Condition/Decision Coverage - C/DC) 5.ระดับครอบคลุมเงื่อนไขแบบหลายกรณี (Multiple Condition Coverage - MCC) และ 6.ระดับครอบคลุมเงื่อนไขแบบแก้ไข (Modified Condition/Decision Coverage - MC/DC)

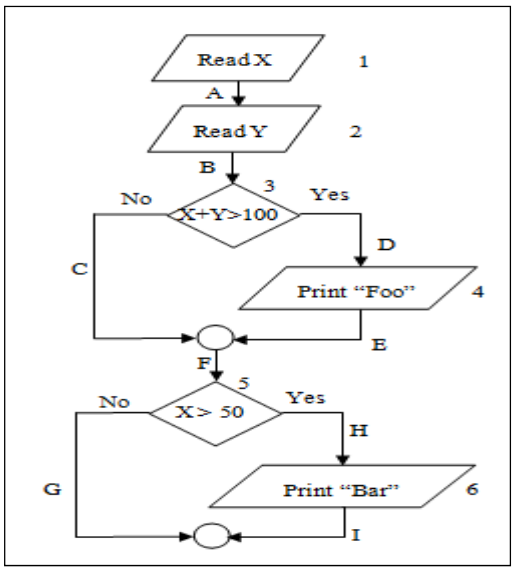
1. ระดับครอบคลุมคำสั่ง (Statement Coverage - SC)

ระดับครอบคลุมคำสั่ง ถือเป็นรูปแบบพื้นฐานที่สุดของความครอบคลุมรหัสโปรแกรม (Code Coverage) ซึ่งถูกใช้ในการวัดร้อยละของคำสั่งที่มีการทำงานระหว่างการทดสอบ เป้าหมายของระดับครอบคลุมคำสั่ง คือ ต้องการให้ทุกคำสั่งในโปรแกรมได้รับการทดสอบอย่างน้อยหนึ่งครั้ง ซึ่งสามารถคำนวนค่าความครอบคลุมในระดับคำสั่งของการทดสอบ T สำหรับโปรแกรม P ได้ โดยการหาสัดส่วนระหว่างจำนวนคำสั่งที่ถูกเรียกทดสอบในการทดสอบ T นั้นกับจำนวนคำสั่งทั้งหมดในโปรแกรม (Pezze และ Young, 2008, น. 215) แสดงดังสูตรด้านล่าง

$$\text{ค่าความครอบคลุมในระดับคำสั่ง} = \frac{\text{จำนวนคำสั่งที่ถูกเรียกทดสอบ}}{\text{จำนวนคำสั่งทั้งหมดในโปรแกรม}}$$

```
1.  read (X)
2.  read (Y)
3.  if (X+ Y> 100)
4.  {
5.      Print “Foo”
6.  }
7.  If (X>50)
8.  {
9.      Print “Bar”
10. }
```

รูปที่ 1 แสดงชุดคำสั่งที่ 1



รูปที่ 2 แสดงFlow chart ของชุดคำสั่งที่ 1

ตัวอย่างที่ 1 แสดงโปรแกรมรับค่า X และ Y มาทำการคำนวณและแสดงผลลัพธ์ที่ได้ทางหน้าจอ จากรูปที่ 2 กำหนดกรณีทดสอบ คือ x=60 และ y=50 เพียงชุดเดียวก็ครอบคลุมทุกคำสั่ง ซึ่งจะได้จำนวนที่สั้นที่สุดของเส้นทางซึ่งครอบคลุมทุกคำสั่งได้ คือ 1A-2B-3D-4E-F-5H-6I ซึ่งสามารถครอบคลุมทุกคำสั่ง คือ 123456

ความครอบคลุมในระดับคำสั่งนี้มีประโยชน์ในการทดสอบคำสั่งที่มักจะไม่ได้รับการทดสอบ เช่นการทำงานกับคำสั่งประเภททางเลือก (switch-case) และการดำเนินการกับเอกเซ็บชั่น (exception handling) (Page และคนอื่นๆ, 2009 น. 123-125)

ตัวอย่างที่ 2 จากรูปที่ 3 แสดงชุดคำสั่งที่ 2 การทดสอบที่ครอบคลุมในระดับคำสั่ง ต้องการกรณีทดสอบ 3 กรณี คือ a=1, a=2 และ a=3 จึงจะครอบคลุมการทำงานของทุกคำสั่ง

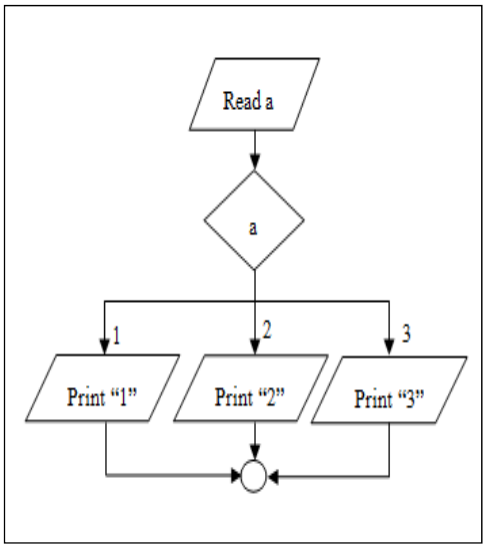
กรณีที่ 1 กำหนดให้ a=1 จะทำการทดสอบคำสั่งในบรรทัดที่ 1, 2, 4, 5 และ 6

กรณีที่ 2 กำหนดให้ a=2 จะทำการทดสอบคำสั่งในบรรทัดที่ 1, 2, 7, 8 และ 9

กรณีที่ 3 กำหนดให้ a=3 จะทำการทดสอบคำสั่งในบรรทัดที่ 1, 2, 10, 11 และ 12

```
1. read (a)
2. switch (a):
3. {
4. case 1:
5.          print “1”
6.          break
7. case 2:
8.          print “2”
9.          break
10. case 3:
11.         print “3”
12.         break
13. }
```

รูปที่ 3 แสดงชุดคำสั่งที่ 2



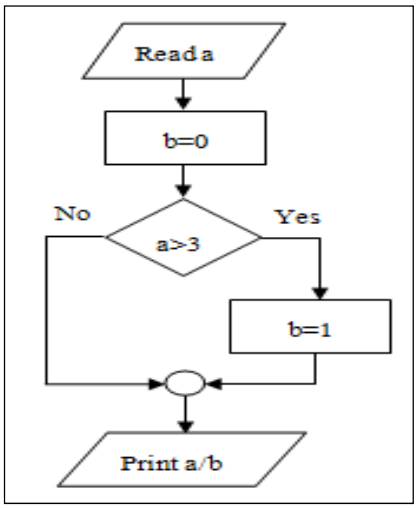
รูปที่ 4 แสดง flow chart ของชุดคำสั่งที่ 2

ในบางกรณีการทดสอบที่ครอบคลุมทุกคำสั่งนั้นไม่สามารถตรวจพบข้อผิดพลาดทั้งหมดได้ เพราะบางครั้งความผิดพลาดกลับเกิดจากการทำงานในบางครั้งที่บางคำสั่งไม่ถูกเรียกใช้งาน ยกตัวอย่างเช่น

ตัวอย่างที่ 3 จากรูปที่ 5 แสดงชุดคำสั่งที่ 3 การทดสอบชุดคำสั่งนี้ให้ครอบคลุมที่ระดับคำสั่งที่ 100% สามารถทำโดยการทดสอบเพียง 1 กรณี คือการกำหนดให้เงื่อนไข a>3 เป็นจริง เช่น กำหนดให้ a=5 ส่งผลให้ผลลัพธ์ในบรรทัดที่ 6 เท่ากับ 5 แต่ถ้ากำหนดให้เงื่อนไข a>3 เป็นเท็จ เช่น กำหนดให้ a=2 จะส่งผลให้เกิดความผิดพลาดในบรรทัดที่ 6 เนื่องจากการหารด้วย 0 ซึ่งความผิดพลาดนี้จะไม่ได้รับการตรวจพบ หากการทดสอบถูกทำที่ระดับความครอบคลุมของคำสั่งเท่านั้น

```
1. read (a)
2. b=0
3. if( a>3 ){
4.          b=1
5. }
6. print (a/b)
```

รูปที่ 5 แสดงชุดคำสั่งที่ 3



รูปที่ 6 แสดง flow chart ของ ชุดคำสั่งที่ 3

2. ระดับครอบคลุมการตัดสินใจ (Decision Coverage - DC)

เป็นการทดสอบความครอบคลุมในระดับที่สูงขึ้น ซึ่งต้องพิจารณาการตัดสินใจ (Decision) ของชุดคำสั่งทั้ง 2 กรณี คือ กรณีที่การตัดสินใจเป็นจริงและกรณีที่การตัดสินใจเป็นเท็จ เช่น จากชุดคำสั่งที่ 2 ต้องพิจารณาทั้งกรณีที่เงื่อนไข a>3 เป็นจริงและกรณีที่เงื่อนไข a>3 เป็นเท็จ

เป้าหมายของระดับครอบคลุมการตัดสินใจคือ ต้องการให้ทุกการตัดสินใจในโปรแกรมได้รับการทดสอบอย่างน้อยหนึ่งครั้งทั้งในกรณีที่เป็นจริงและในกรณีที่เป็นเท็จ โดยสามารถคำนวนค่าความครอบคลุมในระดับการตัดสินใจในการทดสอบ T สำหรับโปรแกรม P ได้จากการหาสัดส่วนระหว่างจำนวนทางเลือกที่ถูกทดสอบในการทดสอบ T กับจำนวนทางเลือกทั้งหมดในโปรแกรม (Pezze และ Young, 2008, น. 217) ดังสูตรด้านล่าง

$$\textit{ค่าความครอบคลุมในระดับการตัดสินใจ (DC)} = \frac{\textit{จำนวนทางเลือกที่ถูกเรียกทดสอบ}}{\textit{จำนวนทางเลือกทั้งหมดในโปรแกรม}}$$

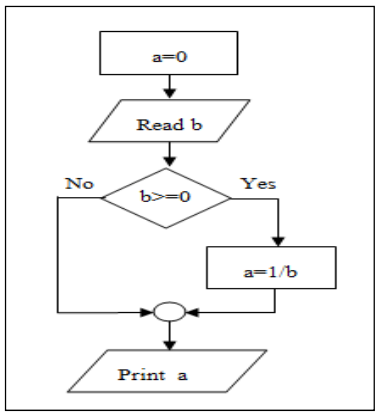
จากชุดคำสั่งที่ 1 หากใช้กรณีทดสอบ คือ x=60 และ y=50 จะได้เส้นทางคือ1A-2B-3D-4E-F-5H-6I ซึ่งครอบคลุมในกรณีที่โปรแกรมเป็นจริง แต่ยังไม่ครอบคลุมในกรณีที่เป็นเท็จ (เส้น B และ F ยังไม่ได้รับการทดสอบ) ดังนั้นจึงต้องการชุดทดสอบอีก 1 ชุด กำหนดให้ x=30 และ y=60 ทำให้ได้เส้นทาง คือ 1A-2B-3C-F-5G ซึ่งครอบคลุมในกรณีที่เป็นเท็จ ดั้งนั้นการจะครอบคลุมในระดับการตัดสินใจต้องใช้ชุดทดสอบทั้งสิ้น 2 ชุด

บางกรณี การทดสอบที่ระดับครอบคลุมการตัดสินใจยังไม่สามารถครอบคลุมการทดสอบที่น่าสนใจได้ในทุกกรณี ดังตัวอย่างด้านล่าง

ตัวอย่างที่ 4 ชุดคำสั่งที่ 4 จากรูปที่ 7 เงื่อนไข b >= 0 หากการทดสอบมี 2 กรณีคือกรณีที่ b>0 และกรณีที่ b<0 ก็เพียงพอที่จะครอบคลุมที่ระดับการตัดสินใจ DC ที่ 100% แล้ว ซึ่งการทดสอบนี้ยังไม่ได้ทำการทดสอบกรณี b=0 ซึ่งเป็นกรณีที่สำคัญอีกกรณีหนึ่งที่จะทำให้เกิดความผิดพลาดในบรรทัดที่ 4 จากการหารด้วย 0 ซึ่งความผิดพลาดนี้จะไม่ได้รับการตรวจพบหากการทดสอบถูกทำที่ระดับครอบคลุมการตัดสินใจเท่านั้น

```
1. a=0
2. read (b)
3. if( b >= 0){
4.     a=1/b
5. }
6. print (a)
```

รูปที่ 7 แสดงชุดคำสั่งที่ 4



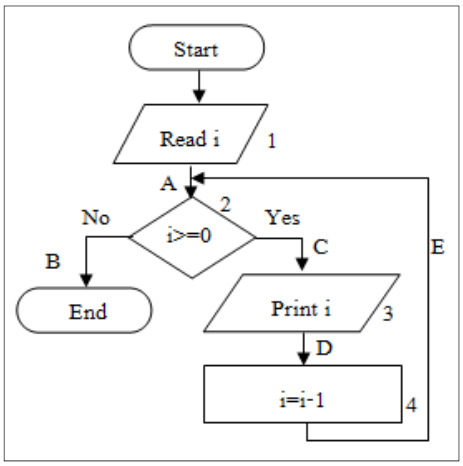
รูปที่ 8 แสดง flow chart ของชุดคำสั่งที่ 4

3. ระดับครอบคลุมเส้นทางการทำงาน (Path Coverage - PC)

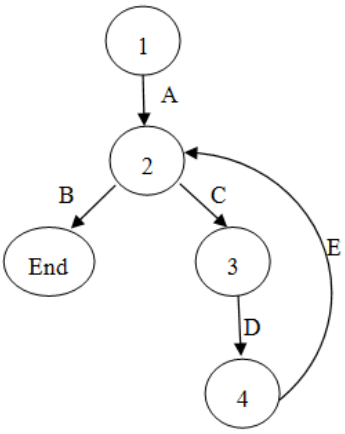
หากพิจารณาชุดคำสั่งในลักษณะของกราฟควบคุมการทำงาน (Control Flow Graph) การทดสอบที่ระดับครอบคลุมคำสั่ง สามารถเทียบเคียงได้กับการครอบคลุมโหนดในกราฟ (Node Coverage) ในขณะที่ระดับครอบคลุมการตัดสินใจ สามารถเทียบเคียงได้กับการครอบคลุมเอดก์ในกราฟ (Edge Coverage) (Ammann และ Offutt, 2008 น. 52-54) จากรูปที่ 11 (1) แสดงกราฟของการควบคุมการทำงานที่มี 5 โหนด ซึ่งโหนด 2 เป็นโหนดที่มีการวนรอบ รูปที่ 11 (2) (3) และ (4) แสดงกราฟเส้นทางย่อยที่เป็นไปได้ของชุดคำสั่งที่ 5

```
1.  read i
2.  while (i>=0)
3.  {
4.          print i
5.          i=i-1
6.  }
```

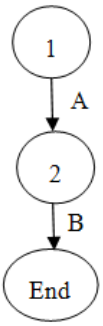
รูปที่ 9 แสดงชุดคำสั่งที่ 5



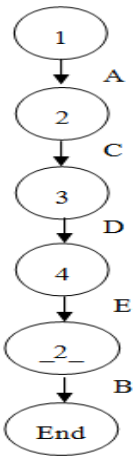
รูปที่ 10 แสดง flow chart ของชุดคำสั่งที่ 5



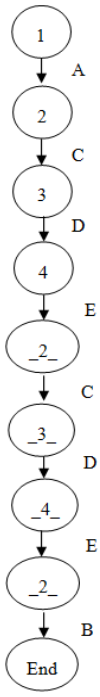
(1)



(2)



(3)



(4)

รูปที่ 11 แสดงกราฟของการควบคุมการทำงานของชุดคำสั่งที่ 5

ตัวอย่างที่ 5 จากชุดคำสั่งที่ 5 แสดงชุดคำสั่งที่มีการรับข้อมูลเข้า แล้วทำการตรวจสอบเงื่อนไขและทำงานไปจนกว่าเงื่อนไขจะเป็นเท็จ จึงจะหยุดการทำงาน เรียกการทำงานในลักษณะนี้ว่า การวนรอบ (loop) ชุดทดสอบที่นำมาทดสอบแสดงดังตารางที่ 1

ตารางที่ 1 แสดงชุดทดสอบที่ใช้ทดสอบชุดคำสั่งที่ 5

ชุดทดสอบ	i	การวนรอบ	เส้นทาง
1	-1	ไม่มี	1A-2B
2	0	1 รอบ	1A-2C-3D-4E-2B
3	1	2 รอบ	1A-2C-3D-4E-2C-3D-4E-2B

การทดสอบที่ระดับความครอบคลุมนี้จะทดสอบการทำงานของชุดคำสั่งในทุก ๆเส้นทางการทำงานในทางปฏิบัตินั้น หากชุดคำสั่งที่ทำการทดสอบมีการวนรอบ (loop) ที่มีจำนวนรอบได้ไม่จำกัดจำนวน (∞) หรือมีทางเลือกจำนวนมาก (astronomical) ย่อมจะไม่สามารถครอบคลุมเส้นทางการทำงานทั้งหมดได้ ในสถานะการณ์เช่นนี้ อาจทำการทดสอบการวนรอบได้ใน 3 กรณี คือ 1.ไม่มีการวนรอบเลย 2.วนรอบหนึ่งครั้ง และ 3.วนรอบหลายครั้ง ในกรณีที่สามนี้ เราไม่อาจกำหนดตายตัวไปได้ว่าจำนวนการวนรอบเท่าไรจึงจะเพียงพอ ในการนี้ McCabe (1982) ได้เสนอวิธีการทดสอบด้วยการพิจารณา Cyclomatic Complexity Metric ซึ่งรายละเอียดจะอยู่นอกเหนือจากขอบเขตของบทความนี้ ผู้อ่านที่สนใจสามารถดูรายละเอียดเพิ่มเติมได้ (Jorgensen, 2002, น.146-153)

การคำนวนค่าความครอบคลุมเส้นทางการทำงานสามารถหาได้จากสูตร (Pezze และ Young, 2008, น. 222) ซึ่งสูตรนี้จะใช้ได้ก็ต่อเมื่อสามารถหาเส้นทางการทำงานทั้งหมดได้เท่านั้น

$$\textit{ค่าความครอบคลุมเส้นทางการทำงาน} = \frac{\textit{จำนวนเส้นทางที่ถูกเรียกทดสอบ}}{\textit{จำนวนเส้นทางทั้งหมด}}$$

จากชุดคำสั่งที่ 1 ข้างต้น หากต้องการครอบคลุมเส้นทางการทำงานทั้งหมดจะต้องใช้ชุดทดสอบ 4 ชุด ดังตารางที่ 2 ด้านล่าง

ตารางที่ 2 แสดงชุดทดสอบที่ต้องใช้ทั้งหมดหากต้องการครอบคลุมเส้นทางการทำงานของชุดคำสั่งที่ 1

ชุดทดสอบ	X	Y	เส้นทาง
1	60	50	1A-2B-3D-4E-F-5H-6I
2	50	60	1A-2B-3D-4E-F-5G
3	70	20	1A-2B-3C-F-5H-6I
4	30	60	1A-2B-3C-F-5G

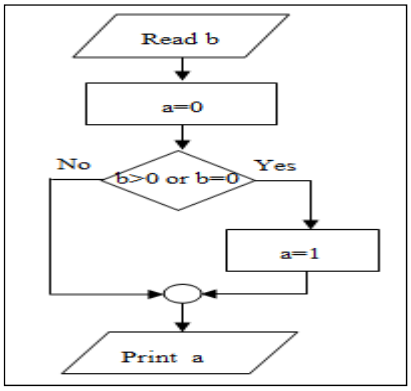
4. ระดับครอบคลุมเงื่อนไขและการตัดสินใจ (Condition/Decision Coverage - C/DC)

จากชุดคำสั่งที่ 4 ในหัวข้อที่สอง เพื่อให้สามารถทำการทดสอบได้ครอบคลุมมากยิ่งขึ้น เราสามารถกระจายเงื่อนไข (b >= 0) เป็นสองส่วน คือ (b>0) or (b=0) ซึ่งจะทำให้สามารถเพิ่มความครอบคลุมให้อยู่ที่ระดับ**เงื่อนไข** นั่นคือ ในแต่ละเงื่อนไขย่อยของการตัดสินใจจะต้องมีการทดสอบสองกรณี คือกรณีที่เงื่อนไขนั้นเป็นจริงและกรณีที่เงื่อนไขนั้นเป็นเท็จ นอกจากนั้น การตัดสินใจจะต้องถูกทดสอบครบทั้งสองกรณี คือกรณีที่การตัดสินใจนั้นเป็นจริงและกรณีที่การตัดสินใจนั้นเป็นเท็จ ซึ่งการคำนวนค่าความครอบคลุมเงื่อนไขและการตัดสินใจ สามารถหาได้จากสูตร (Pezze และ Young, 2008, น. 219)

$$\textit{ค่าความครอบคลุมเงื่อนไขและการตัดสินใจ} = \frac{\textit{จำนวนค่าความจริงทั้งหมดที่ถูกกำหนดให้แต่ละเงื่อนไขย่อย}}{\textit{จำนวนเงื่อนไขย่อยทั้งหมด} * 2}$$

```
1. read (b)
2. a=0
3. if( b>0 or b=0 ){
4.          a=1
5. }
6. print (a)
```

รูปที่ 12 แสดงชุดคำสั่งที่ 6



รูปที่ 13 แสดง flow chart ของชุดคำสั่งที่ 6

เมื่อพิจารณาชุดคำสั่งที่ 6 การทดสอบให้ครอบคลุมที่ระดับเงื่อนไขนั้น อาจมองว่าทำได้ง่าย โดยการทดสอบเพียงสองกรณี คือกรณีที่เงื่อนไข b>0 และ b=0 เป็นจริงทั้งคู่ และกรณีที่เงื่อนไขทั้งสองเป็นเท็จทั้งคู่ แต่เมื่อพิจารณาต่อไป กลับพบว่าเงื่อนไขทั้งสองจะเป็นจริงพร้อมกันไม่ได้ เนื่องจากค่าของ b นั้นขัดแย้งกัน Utting และ Legeard (2004, น. 219-220) จึงได้อธิบายวิธีการสร้างชุดทดสอบเพื่อให้ครอบคลุมที่ระดับเงื่อนไข โดยเริ่มต้นที่การกำหนดการทดสอบแรกให้ทุกเงื่อนไขมีค่าความจริงเป็นเท็จ (ในกรณีที่นิพจน์การตัดสินใจนั้นเป็น Disjunctive Normal Form) จากนั้นจึงเปลี่ยนค่าความจริงของแต่ละเงื่อนไขให้เป็นจริง โดยในการเปลี่ยนแต่ละครั้งก็จะเป็นการสร้างกรณีทดสอบใหม่ โดยกำหนดให้เงื่อนไขอื่นๆ มีค่าเป็น "ไม่สนใจ" เพื่อป้องกันไม่ให้มีความขัดแย้งกันระหว่งเงื่อนไข ในทางกลับกัน ในกรณีที่นิพจน์นั้นเป็น Conjunctive Normal Form เราต้องกำหนดการทดสอบแรกให้ทุกเงื่อนไขมีค่าความจริงเป็นจริง จากนั้นจึงเปลี่ยนค่าความจริงของแต่ละเงื่อนไขให้เป็นเท็จสำหรับแต่ละกรณีทดสอบ และกำหนดให้เงื่อนไขอื่นๆ มีค่าเป็น "ไม่สนใจ"

ดังนั้น ชุดคำสั่งด้านบนจะต้องประกอบไปด้วยการทดสอบ 3 กรณีด้วยกัน ในแถวแรก เงื่อนไข b>0 และเงื่อนไข b=0 นั้น เป็นเท็จทั้งคู่ จึงสรุปได้ว่าในกรณีแรกนี้ ค่า b<0 กรณีที่สองค่า b>0 และกรณีที่สามค่า b=0

ตารางที่ 3 แสดงการทดสอบตามทฤษฎีของ Utting และ Legeard

b>0	b=0	ค่า b
เท็จ	เท็จ	<0
จริง	ไม่สนใจ	>0
ไม่สนใจ	จริง	0

5. ระดับครอบคลุมเงื่อนไขแบบหลายกรณี (Multiple Condition Coverage - MCC)

จะเห็นได้ว่าระดับครอบคลุมเงื่อนไขนั้นยังไม่ได้พิจารณาความสัมพันธ์ระหว่างค่าของเงื่อนไขแต่ละค่าที่อาจมีความสัมพันธ์ระหว่างกันได้ การทดสอบที่ครอบคลุมในระดับเงื่อนไขแบบหลายกรณี จะพิจารณาความสัมพันธ์ระหว่างค่าของเงื่อนไขแต่ละค่าในทุกกรณี ซึ่งจะทำให้มีกรณีที่ต้องพิจารณาเท่ากับ 2^n เมื่อ n คือจำนวนของเงื่อนไขย่อย และค่าความเป็นไปได้ของแต่ละเงื่อนไขย่อยคือจริงหรือเท็จ จากตัวอย่างข้างต้น เมื่อมีเงื่อนไข 3 เงื่อนไข จึงต้องมีกรณีทดสอบที่ต้องพิจารณา 8 กรณี จะเห็นได้ว่าการทดสอบที่ระดับความครอบคลุมนี้จะทำได้กับการตัดสินใจที่มีจำนวนเงื่อนไขไม่มากนักเท่านั้น

ตารางที่ 4 แสดงตัวอย่างการทดสอบที่ครอบคลุมในระดับเงื่อนไขแบบหลายกรณี

a	B	C	(a or b) & c
จริง	จริง	จริง	จริง
จริง	จริง	เท็จ	เท็จ
จริง	เท็จ	จริง	จริง
จริง	เท็จ	เท็จ	เท็จ
เท็จ	จริง	จริง	จริง
เท็จ	จริง	เท็จ	เท็จ
เท็จ	เท็จ	จริง	เท็จ
เท็จ	เท็จ	เท็จ	เท็จ

ทั้งนี้ เราสามารถลดจำนวนกรณีทดสอบลงได้ด้วยการลัดวงจร (Short Circuit) โดยการจัดเรียงนิพจน์การตัดสินใจใหม่ เช่น c & (a or b) ซึ่งจะทำให้สามารถลดจำนวนกรณีทดสอบลงได้เหลือเพียง 4 กรณีเท่านั้น แสดงดังตารางที่ 5

ตารางที่ 5 แสดงการลดจำนวนกรณีทดสอบลงได้ด้วยการลัดวงจร (Short Circuit)

c	a	B	c & (a or b)
จริง	จริง	-	จริง
จริง	เท็จ	จริง	จริง
จริง	เท็จ	เท็จ	เท็จ
เท็จ	-	-	เท็จ

6. ระดับครอบคลุมเงื่อนไขแบบแก้ไข (Modified Condition/Decision Coverage - MC/DC)

MC/DC (Hayhurst และคนอื่นๆ, 2001) เป็นระดับของการครอบคลุมที่ถูกใช้อย่างแพร่หลาย และได้ถูกกำหนดไว้เป็นมาตรฐานในการทดสอบซอฟต์แวร์ทางการบิน เช่น (RTCA Inc., 1992) MC/DC เพิ่มความเข้มข้นในการทดสอบขึ้นจาก C/DC โดยกำหนดให้แต่ละเงื่อนไขในการตัดสินใจถูกแสดงว่าส่งผลต่อการตัดสินใจอย่างเป็นอิสระจากเงื่อนไขอื่นๆ คือ เงื่อนไขย่อยๆ แต่ละเงื่อนไขจะต้องปรากฏค่าความจริงทั้งจริงและเท็จในชุดทดสอบ ซึ่งกรณีทดสอบกรณีหนึ่งในนั้นจะต้องส่งผลต่อการตัดสินใจเป็นจริงและกรณีทดสอบอีกกรณีหนึ่งจะต้องส่งผลต่อการตัดสินใจเป็นเท็จ การทดสอบที่ระดับ MC/DC มีข้อดี คือ จะใช้จำนวนกรณีทดสอบเพียง n+1 กรณีเท่านั้น เมื่อ n คือจำนวนของเงื่อนไขย่อย (Pezze และ Young, 2008, น. 221) ซึ่งทำให้มีกรณีทดสอบน้อยกว่าที่ระดับ MCC อย่างมาก

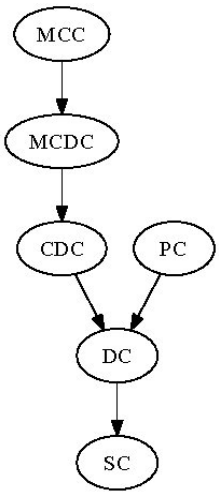
ตัวอย่างที่ 6 กรณีที่การตัดสินใจหนึ่งประกอบด้วยสามเงื่อนไขย่อยคือ (a or b) & c เราต้องการกรณีทดสอบทั้งสิ้น 4 กรณี โดยพิจารณาได้จาก ค่าความจริงในตารางที่ถูกขีดเส้นใต้เป็นเงื่อนไขที่ส่งผลต่อผลลัพธ์จากการตัดสินใจอย่างเป็นอิสระจากเงื่อนไขอื่นๆ กรณีทดสอบที่ 1 เทียบกับกรณีทดสอบที่ 2 ค่าความจริงของ c เปลี่ยนจากจริงเป็นเท็จส่งผลให้ค่าการตัดสินใจเปลี่ยนจากจริงเป็นเท็จ กรณีทดสอบที่ 1 เทียบกับกรณีทดสอบที่ 3 ค่าความจริงของ a เปลี่ยนจากจริงเป็นเท็จส่งผลให้ค่าการตัดสินใจเปลี่ยนจากจริงเป็นเท็จ และ กรณีทดสอบที่ 3 เทียบกับกรณีทดสอบที่ 4 ค่าความจริงของ b เปลี่ยนจากเท็จเป็นจริงส่งผลให้ค่าการตัดสินใจเปลี่ยนจากเท็จเป็นจริง ตามไปด้วย

ตารางที่ 6 แสดงตัวอย่างการทดสอบที่ระดับ MC/DC

กรณีทดสอบที่	A	b	c	(a or b) & c
1	จริง	เท็จ	จริง	จริง
2	จริง	เท็จ	เท็จ	เท็จ
3	เท็จ	เท็จ	จริง	เท็จ
4	เท็จ	จริง	จริง	จริง

บทสรุป

เกณฑ์การประเมินความครอบคลุมในการทดสอบซอฟต์แวร์แบบโครงสร้างทั้ง 6 ระดับ มีข้อดีและข้อเสียแตกต่างกัน โดยทั่วไปแล้วเกณฑ์ที่เข้มข้นน้อยก็ย่อมจะใช้กรณีทดสอบน้อยกว่า แต่ก็จะให้ความเชื่อมั่นในชุดคำสั่งที่ได้ทำการทดสอบนั้นน้อยลงไปด้วยเช่นกัน อย่างไรก็ตาม อาจกล่าวได้ว่า การทดสอบซอฟต์แวร์ไม่ว่าจะทดสอบที่ระดับความครอบคลุมใดๆ ก็มีเป้าหมายเดียวกันอยู่สามประการ คือ ต้องการที่จะครอบคลุมการทำงานของชุดคำสั่งให้ครบถ้วนตามระดับความครอบคลุมที่มุ่งหวัง ต้องการลดความซ้ำซ้อนของการทดสอบที่ไม่จำเป็น และการทดสอบนั้นต้องช่วยในการหาตำแหน่งของความผิดพลาด (fault localization – Zeller และ Hildebrandt, 2002) ของชุดคำสั่งด้วยนั่นคือ กรณีทดสอบก็จะต้องครอบคลุม ไม่มีความซ้ำซ้อน และละเอียดเพียงพอที่วิศวกรซอฟต์แวร์จะได้ข้อมูลจากการทดสอบไปหาข้อผิดพลาดของโปรแกรม



รูปที่ 14 ความสัมพันธ์ระหว่างความครอบคลุมในการทดสอบซอฟต์แวร์แบบโครงสร้างทั้ง 6 ระดับ

ความสัมพันธ์ระหว่างระดับความครอบคลุมในการทดสอบซอฟต์แวร์แบบโครงสร้างทั้ง 6 ระดับ อยู่ในลักษณะที่ระดับความครอบคลุมที่เข้มข้นมากกว่าจะรวมความครอบคลุมที่เข้มข้นน้อยกว่าเข้าไปด้วย (subsume) นั่นคือ หากระดับ DC เข้มข้นกว่า SC สามารถอธิบายได้ว่า DC รวม SC ไว้ด้วย (DC subsume SC) หรือ DC ⟶ SC จึงสามารถสรุปตามรูปที่ 12 ได้ดังนี้

PC ⟶ DC ⟶ SC และ

MCC ⟶ MC/DC ⟶ C/DC ⟶ DC ⟶ SC

ในระดับ MC/DC Rajan และคนอื่นๆ (2008) ได้แสดงให้เห็นว่า สำหรับชุดคำสั่งที่มีความหมายเดียวกันที่ถูกเขียนขึ้น 2 แบบ การทดสอบที่ครอบคลุมชุดคำสั่งแบบที่ 1 ที่ระดับ MC/DC นั้น อาจไม่ครอบคลุมชุดคำสั่งอีกชุดหนึ่งก็ได้ ซึ่งหมายความว่า MC/DC มีความอ่อนไหวต่อรูปแบบการเขียนโปรแกรมของนักพัฒนาซอฟต์แวร์ การศึกษานี้ใช้กรณีศึกษาจำนวน 6 ชิ้นจากซอฟต์แวร์ในอุตสาหกรรมการบิน พบว่าในบางกรณีการเขียนโปรแกรมแบบหนึ่งอาจต้องการกรณีทดสอบมากกว่าการเขียนโปรแกรมอีกวิธีหนึ่งถึง 20 เท่า เพื่อที่จะให้ครอบคลุมที่ระดับ MC/DC ประเด็นนี้จึงต้องการการศึกษาต่อไปในการที่จะออกแบบระดับความครอบคลุมที่พิจารณาการกระจายของนิพจน์หรือคำสั่ง (Inline Function) เข้ามาในการสร้างกรณีทดสอบด้วย

นอกจากนั้น การทดสอบจากแบบจำลอง (Model-based Testing) เป็นเทคนิคหนึ่งที่ได้รับความสนใจจากทั้งภาควิชาการและภาคอุตสาหกรรมในปัจจุบัน ยกตัวอย่างเช่น Offutt และ Abdurazik (1999) Bouquet และคนอื่นๆ (2007) สร้างแบบทดสอบซอฟต์แวร์จากแบบจำลองในภาษา UML (Unified Modeling Language) Larsen และคนอื่นๆ (2010) สร้างแบบทดสอบซอฟต์แวร์จากแบบจำลองในภาษา VDM (Vienna Development Method) การออกแบบระดับความครอบคลุมที่พิจารณาจากแบบจำลอง (model coverage) เช่น Andrews และคนอื่นๆ (2003) และ Briand และคนอื่นๆ (2010) แทนที่จะเป็นการพิจารณาระดับความครอบคลุมจากโครงสร้างของชุดคำสั่งจึงเป็นประเด็นที่สำคัญที่ควรได้รับการสำรวจตรวจสอบเช่นกัน

บรรณานุกรม

Ammann, P., and Offutt, J. 2008 **Introduction to Software Testing**. New York, NY: Cambridge University Press.

Andrews, A., France, R., Ghosh, S. and Craig, G. (2003), **Test adequacy criteria for UML design models.** Software Testing, Verification and Reliability, 13: 95–127.

Bouquet, F., Grandpierre, C., Legeard, B., Peureux, F., Vacelet, N., and Utting, M. 2007. **A subset of precise UML for model-based testing**. In *Proceedings of the 3rd international workshop on Advances in model-based testing* (A-MOST '07). ACM, New York, NY, USA, 95-104.

Briand, L., Labiche, Y. , and Lin, Q. 2010. **Improving the coverage criteria of UML state machines using data flow analysis.** Software Testing, Verification and Reliability, 20, 3: 177-207.

Dijkstra, E W. 1970. **Notes On Structured Programming**. Technological University of Eindhoven.

Hayhurst, Kelly J., Veerhusen, Dan S., Chilenski, John J., and Rierson Leanna K. 2001. NASA/TM-2001-210876 **A Practical Tutorial on Modified Condition/Decision Coverage**. Hanover, MD: NASA Center for AeroSpace Information (CASI).

Jorgensen, P. C. 2002. **Software Testing A Craftsman's Approach**. 2nd edition. CRC Press.

Kuhn, D. R. 1999. **Fault classes and error detection capability of specification-based testing**. ACM Trans. Softw. Eng. Methodol. 8, 4 (Oct. 1999), 411-424.

Larsen, P., Lausdahl, K., and Battle, N. 2010. **Combinatorial Testing for VDM.** In *Proceedings of the 2010 8th IEEE International Conference on Software Engineering and Formal Methods* (SEFM '10). IEEE Computer Society, Washington, DC, USA, 278-285.

McCabe, T. J. 1982. **Structural Testing: A Software Testing Methodology using the Cyclomatic Complexity Metric**. Washington, D.C. :NIST.

Offutt, J., and Abdurazik, A.. 1999. **Generating tests from UML specifications.** In *Proceedings of the 2nd international conference on the unified modeling language: beyond the standard* (UML'99), Robert France and Bernhard Rumpe (Eds.). Springer-Verlag, Berlin, Heidelberg, 416-429.

Page, A., Johnston, K., and Rollison B. 2009. **How we test software at Microsoft**. Redmond, WA: Microsoft Press.

Pezze, M., and Young, M. 2008 **Software Testing and Analysis: Process, Principles, and Techniques**. USA: Wiley.

RTCA, Incorporated. 1992. RTCA/DO-178B **"Software Considerations in Airborne Systems and Equipment Certification"**.

Rajan, A., Whalen, M. W., and Heimdahl, M. P. 2008. **The effect of program and model structure on mc/dc test adequacy coverage**. In Proceedings of the 30th international Conference on Software Engineering (Leipzig, Germany, May 10 - 18, 2008). ICSE '08. ACM, New York, NY, 161-170.

Utting, M. and Legeard, B. 2007 **Practical Model-based Testing**. San Francisco, CA: Morgan Kaufmann.

Zeller, A., and Hildebrandt, R. 2002 **Simplifying and Isolating Failure-Inducing Input.** IEEE Trans. Softw. Eng. 28, 2 (February 2002), 183-200