# แบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไขสำหรับสร้างกรณีทดสอบบนพื้นฐานของแผนภาพกิจกรรมของ UML

ปรัชญานีย์ ไทยเกิด สุภาภรณ์ กานต์สมเกียรติ และอภิรดา ธาดาเดช

SEA Lab ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยสงขลานครินทร์ สงขลา

Email: {s5110220041, supaporn.k, apirada.t}@psu.ac.th

**บทคัดย่อ**

ประสิทธิภาพของการทดสอบซอฟต์แวร์ขึ้นอยู่กับคุณภาพความครอบคลุมของข้อมูลนำเข้าที่ใช้ในการทดสอบ ในปัจจุบันซอฟต์แวร์ส่วนใหญ่ถูกพัฒนาขึ้นโดยใช้เทคโนโลยีเชิงวัตถุ และมีภาษามากมายที่ถูกพัฒนาขึ้นเพื่อสนับสนุนการออกแบบซอฟต์แวร์เชิงวัตถุ ภาษาที่ได้รับความนิยมเป็นอย่างมากคือ ภาษาการออกแบบเชิงโมเดล (Unified Modeling Language: UML) งานวิจัยนี้พิจารณาแผนภาพกิจกรรม (Activity Diagram) ซึ่งเป็นแผนภาพประเภทหนึ่งของ UML แผนภาพนี้ถูกใช้เพื่อโมเดลพฤติกรรมของซอฟต์แวร์ ดังนั้นการนำโมเดลจากขั้นตอนนี้มาใช้ในการสร้างกรณีทดสอบจะทำให้สามารถสร้างกรณีทดสอบได้ในขั้นตอนการออกแบบซอฟต์แวร์ บทความนี้เสนอแบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไขสำหรับสร้างกรณีทดสอบจากแผนภาพกิจกรรม แบบจำลองนี้ช่วยให้กรณีทดสอบที่สร้างขึ้นมีประสิทธิภาพและช่วยลดเวลาในการวิเคราะห์เอกสารความต้องการ ผลลัพธ์ที่ได้จากการทดลองแสดงให้เห็นว่าหลักการที่ได้นำเสนอสามารถช่วยให้กรณีทดสอบที่สร้างมีจำนวนน้อย มีความเหมาะสมและสามารถกระทำได้ตั้งแต่ช่วงต้นของกระบวนการพัฒนาซอฟต์แวร์

***คำสำคัญ*** – วิธีการต้นไม้การจำแนก; แผนภาพกิจกรรม; กรณีทดสอบ

## 1. บทนำ

การทดสอบซอฟต์แวร์เป็นขั้นตอนที่สำคัญในกระบวนการผลิตซอฟต์แวร์ ในการทดสอบซอฟต์แวร์ขั้นตอนการสร้างกรณีทดสอบเป็นขั้นตอนที่สำคัญในการกำหนดประสิทธิภาพของการทดสอบ กรณีทดสอบที่มีประสิทธิภาพนั้นจะต้องมีความครอบคลุมความต้องการของระบบและมีจำนวนกรณีทดสอบที่ไม่มากจนเกินไป [1] ในปัจจุบันการพัฒนาซอฟต์แวร์เชิงวัตถุ (Object-Oriented Software Development) กำลังได้รับความนิยมเป็นอย่างมาก เนื่องจากเป็นแนวคิดที่ใกล้เคียงกับความเป็นจริงและมีความสะดวกรวดเร็วในการพัฒนา ความนิยมที่เพิ่มมากขึ้นจึงมีการสร้างเครื่องมือมาช่วยในการวิเคราะห์และออกแบบระบบซอฟต์แวร์เชิงวัตถุ เช่น ภาษาการออกแบบเชิงโมเดล (UML) [2] ซึ่งเป็นภาษาที่ช่วยในการสร้างแบบจำลองเชิงวัตถุที่ประกอบด้วยแผนภาพหลายชนิดที่ใช้เพื่อจำลองการออกแบบซอฟต์แวร์ทั้งในเชิงโครงสร้างและ เชิงพฤติกรรม หนึ่งในแผนภาพของภาษาการออกแบบเชิงโมเดลคือแผนภาพกิจกรรมซึ่งใช้อธิบายขั้นตอนกิจกรรมการทำงานของซอฟต์แวร์ตั้งแต่เริ่มต้นจนสิ้นสุด ส่วนสำคัญส่วนหนึ่งของแผนภาพกิจกรรมที่มีผลในการกำหนดกิจกรรมถัดไป และเป็นตัวกำหนดเส้นทางการทำกิจกรรมต่างๆ คือ ส่วนการตัดสินใจ (decision point) เส้นทางการตัดสินใจที่จะทำกิจกรรมถัดไปหลังออกจากส่วนการตัดสินใจจะขึ้นอยู่กับข้อมูลนำเข้า (input) ที่ได้รับจากกิจกรรมที่เกิดก่อนหน้าการตัดสินใจ โดยข้อมูลนำเข้าจะถูกกำหนดเป็นเงื่อนไขสำหรับทางเลือกแต่ละเส้นทางที่แยกออกจากส่วนการตัดสินใจ ดังนั้นจากเงื่อนไขการตัดสินใจของเส้นทางต่างๆสามารถใช้เพื่อระบุขอบเขตของข้อมูลนำเข้า (input domain) ที่สำคัญต่อการทำงานของซอฟต์แวร์ได้

ปัจจุบันมีงานวิจัยที่นำแผนภาพกิจกรรมมาใช้ในกระบวนการทดสอบซอฟต์แวร์อย่างแพร่หลาย เช่น Chen และ Poon [3] เสนองานวิจัยเรื่อง “Identification of Categories and Choices in Activity Diagrams” ซึ่งเป็นงานวิจัยที่แก้ปัญหาของงานก่อนหน้าในเรื่องการกำหนด category และ choice ใน choice relation table งานวิจัยนี้จึงใช้แผนภาพกิจกรรมมาช่วยในการกำหนด category, choice และความสัมพันธ์ของ choice แต่ละคู่ใน choice relation table ทำให้ลดความซับซ้อนในการระบุความสัมพันธ์ของ choice ในแต่ละ category ลงได้ ซึ่งจะนำไปสู่ขั้นตอนการสร้างกรณีทดสอบจาก choice relation table ได้สะดวกขึ้น Wang Linzhang และคณะ [4] นำเสนองานวิจัยเรื่อง “Generating Test Cases from UML Activity Diagram based on Gray-Box Method” เพื่อสร้างกรณีทดสอบจากแผนภาพกิจกรรมโดยใช้หลักการ Gray-Box ซึ่งเริ่มต้นจากนำ specification และ operation ของซอฟต์แวร์ มาสร้างแผนภาพกิจกรรม จากนั้นสร้าง test scenarios จากแผนภาพกิจกรรมที่ได้ ขั้นตอนสุดท้ายคือการสร้าง test cases

จาก test scenarios เมื่อพิจารณางานวิจัยที่กล่าวมาข้างต้น พบว่าแผนภาพกิจกรรมถูกนำมาใช้ในการระบวนการทดสอบซอฟต์แวร์ในขั้นตอนก่อนการสร้างกรณีทดสอบเท่านั้น แต่ไม่ได้นำแผนภาพกิจกรรมมาใช้สร้างกรณีทดสอบโดยตรง ดังนั้นบทความนี้จึงเสนอแบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไข (Condition- Classification Tree Model: CCTM) สำหรับสร้างกรณีทดสอบโดยใช้แผนภาพกิจกรรม แบบจำลองนี้ถูกสร้างจาก input domain และเงื่อนไขต่างๆ ซึ่งรวบรวมได้จากแผนภาพกิจกรรม โดยนำ input domain มาสร้างต้นไม้การจำแนกแบบมีเงื่อนไขและใช้ต้นไม้ที่ได้นี้สร้างตารางกรณีทดสอบและกรณีทดสอบที่สอดคล้องกับเงื่อนไขต่างๆ

บทความนี้ประกอบด้วยหัวข้อดังต่อไปนี้ หัวข้อที่ 2 เสนอ The Classification Tree Method (CTM) เป็นการอธิบายหลักการ CTM หัวข้อที่ 3 นำเสนอ UML Activity Diagram ซึ่งจะอธิบายเกี่ยวกับรายละเอียดและสัญลักษณ์ที่ใช้ในแผนภาพกิจกรรม หัวข้อที่ 4 อธิบายวิธีการสร้างกรณีทดสอบโดยใช้แบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไข หัวข้อที่ 5 อธิบายการประเมินประสิทธิภาพของกรณีทดสอบและหัวข้อที่ 6 สรุปและอภิปราย

## 2. The Classification Tree Method (CTM)

วิธีการ Classification Tree Method (CTM) เสนอโดย Grochtmann และ Grimm [5][6] เป็นวิธีหนึ่งที่ใช้เทคนิคการทดสอบแบบ black box เทคนิคของวิธี CTM คือการแบ่ง input domain ออกเป็นหมวดหมู่แต่ละหมู่ของ input domain จะถูกแยกย่อยออกเป็น class การแบ่งหมวดหมู่และ class จะเขียนอยู่ในรูปของต้นไม้ ซึ่งเรียกว่าต้นไม้การจำแนก การสร้างกรณีทดสอบโดยใช้วิธีการ CTM จะได้กรณีทดสอบโดยการรวมโหนดของต้นไม้แต่ละกิ่ง การรวมโหนดจะถูกกำหนดลงในตารางกรณีทดสอบ (test case table) ซึ่งทำให้การสร้างกรณีทดสอบทำได้สะดวกและชัดเจนยิ่งขึ้น โดยมีรายละเอียดของขั้นตอนดังต่อไปนี้

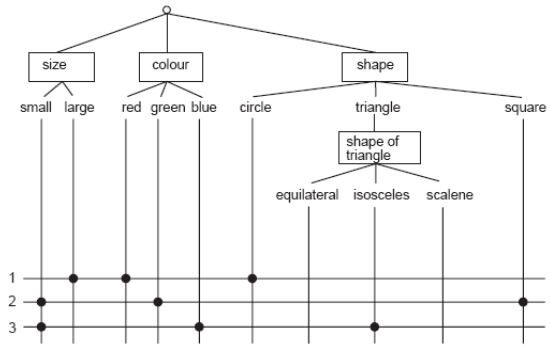
1) วิเคราะห์ specification ของซอฟต์แวร์และแบ่ง input domain ออกเป็นหมวดหมู่พร้อมทั้งระบุ class ที่เป็นไปได้ในแต่ละหมวดหมู่

2) สร้างต้นไม้การจำแนก โดยกำหนดให้แต่ละหมวดหมู่เป็นโหนดลูกแยกออกมาจากโหนดเริ่มต้นและกำหนดให้ class ของหมวดหมู่เป็นโหนดใบของโหนดของหมวดหมู่นั้น

3) สร้างตารางกรณีทดสอบจากต้นไม้การจำแนกที่ได้จากขั้นตอนที่ 2 โดยลากเส้นตารางใต้ต้นไม้การจำแนก

4) เลือกกรณีทดสอบโดยการรวมโหนดใบจากต้นไม้การจำแนกทุกหมวดหมู่ หมวดหมู่ละ 1 โหนด โดยการรวมกันของโหนดจะกระทำในทุกกรณีที่เป็นไปได้ เช่น ภายในต้นไม้การจำแนกประกอบด้วยหมวดหมู่ 2 หมวดหมู่ โดยหมวดหมู่ที่ 1 ประกอบด้วย 3 class และหมวดหมู่ที่ 2 ประกอบด้วย 3 class ดังนั้นจะได้กรณีทดสอบทั้งหมด 6 กรณีทดสอบ

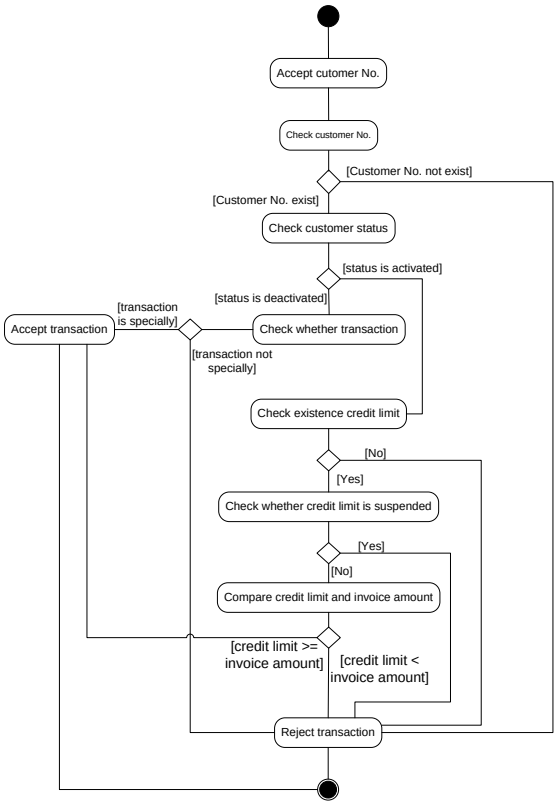
ตัวอย่างการใช้วิธี CTM สร้างกรณีทดสอบจาก requirement specification ของระบบ computer vision [7] ซึ่งกำหนด input domain ออกเป็น 3 กลุ่มได้แก่ size, color และ shape จากนั้นระบุ input domain ย่อยที่เป็นไปได้ในแต่ละกลุ่มแล้วสร้างต้นไม้การจำแนกและตารางกรณีทดสอบ ดังแสดงรายละเอียดดังรูปที่ 1




รูปที่ 1 ตัวอย่างการใช้วิธี CTM สร้างกรณีทดสอบจาก requirement specification ของระบบ computer vision [7]

## 3. UML Activity Diagram

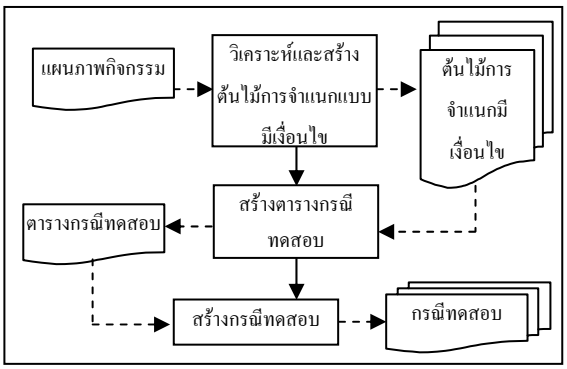
แผนภาพกิจกรรม (activity diagram) [2] เป็นแผนภาพที่แสดงพฤติกรรมการทำงานของระบบซอฟต์แวร์ โดยอธิบายลำดับและขั้นตอนการทำงานที่เกิดขึ้นในซอฟต์แวร์ แผนภาพกิจกรรมมีลักษณะคล้ายกับ ผังงาน (flowchart) โดยจะเริ่มต้นด้วยสัญลักษณ์วงกลมทึบ ● เรียกว่า Initial State และเชื่อมการทำงานต่างๆ ในแผนภาพด้วยลูกศรมีทิศทาง → และแสดงกิจกรรมการทำงานอยู่ภายในสัญลักษณ์ ▭ เรียกว่า State ส่วนสำคัญส่วนหนึ่งที่มีผลในการกำหนดกิจกรรมถัดไปและเป็นตัวกำหนดเส้นทางการทำกิจกรรมต่างๆ คือส่วนการตัดสินใจ (decision point) ซึ่งแสดงโดยใช้สัญลักษณ์ ◇ ในแผนภาพกิจกรรมกรณีที่มีกิจกรรมที่ดำเนินไปพร้อมกันจะแสดงเส้นทางของกิจกรรมด้วยสัญลักษณ์ ┴ เรียกว่าTransition (Join) และสัญลักษณ์ ┬ เรียกว่า Transition (Fork) จุดสิ้นสุดของแผนภาพจะแทนด้วยสัญลักษณ์ ◉ เรียกว่า Final State และมีสัญลักษณ์กรอบสี่เหลี่ยมหรือ swim lane ใช้แบ่งการทำงานในแต่ละส่วนออกจากกัน ตัวอย่างแผนภาพกิจกรรมการขายสินค้า [5] ในรูปที่ 2 เป็นแผนภาพกิจกรรมอธิบายขั้นตอนการทำงานของระบบการขายสินค้าแบบขายปลีก เริ่มต้นการทำงานโดยการให้ลูกค้าเข้าสู่ระบบ จากนั้นระบบจะตรวจสอบข้อมูลบัตรเครดิตและรายการสินค้าที่ลูกค้าสั่งซื้อหากราคาสินค้าที่สั่งซื้อน้อยกว่าวงเงินในบัตรเครดิตของลูกค้าระบบจะทำการขายสินค้าให้แก่ลูกค้า




รูปที่ 2 แผนภาพกิจกรรมการขายสินค้า [5]

## 4. การสร้างกรณีทดสอบจากแผนภาพกิจกรรมโดยใช้หลักการ CCTM

งานวิจัยนี้เสนอแบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไข ซึ่งประยุกต์จากวิธีการ Classification Tree Method (CTM) เพื่อสร้างกรณีทดสอบจากแผนภาพกิจกรรมโดยกระบวนการสร้างกรณีทดสอบที่ได้นำเสนอมีขั้นตอนดังรูปที่ 3




รูปที่ 3 กรอบแนวคิดในการสร้างกรณีทดสอบจากแผนภาพกิจกรรม

ขั้นตอนการสร้างกรณีทดสอบจากแผนภาพกิจกรรมประกอบด้วย 3 ขั้นตอน ดังรายละเอียดต่อไปนี้

**<u>ขั้นตอนที่ 1</u>** วิเคราะห์และสร้างต้นไม้การจำแนกแบบมีเงื่อนไข

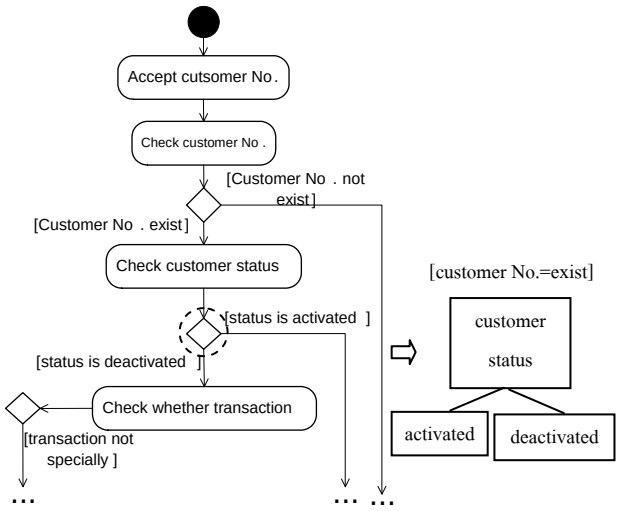
1.1) วิเคราะห์แผนภาพกิจกรรมเพื่อรวบรวมลำดับขั้นตอนการทำกิจกรรมต่างๆ โดยพิจารณาในส่วนการตัดสินใจ (decision point) ที่มีเงื่อนไขกำหนดทางเลือกกิจกรรมถัดไป โดยส่วนการตัดสินใจแต่ละส่วนแสดงลักษณะของ input domain ที่เป็นไปได้ 1 กลุ่ม (category)

1.2) สร้างต้นไม้การจำแนก โดยต้นไม้แต่ละต้นจะถูกสร้างโดยพิจารณาแต่ละตำแหน่งของการตัดสินใจ การสร้างต้นไม้การจำแนกแต่ละต้นมีขั้นตอนดังนี้

1) สร้างโหนดเริ่มต้นสำหรับส่วนการตัดสินใจที่กำลังพิจารณา ตัวอย่างดังแสดงในรูปที่ 4 ซึ่งส่วนการตัดสินใจที่กำลังพิจารณาคือ check customer status ดังนั้นจึงสร้างโหนดเริ่มต้น customer status ขึ้น

2) สร้างโหนดลูก โดยพิจารณาจากเส้นทางที่ออกจากส่วนการตัดสินใจที่กำลังพิจารณา ตัวอย่างดังรูปที่ 4 ส่วนการตัดสินใจ check customer status ประกอบด้วยเส้นทางการตัดสินใจออกมา 2 เส้น ดังนั้นสร้างโหนดลูก 2 โหนดให้กับโหนด customer status

3) ระบุเงื่อนไขให้กับต้นไม้การจำแนกโดยพิจารณาจากเส้นทางก่อนเข้าถึงส่วนการตัดสินใจที่กำลังพิจารณาว่าได้ผ่านเงื่อนไขส่วนการตัดสินใจอื่นๆ ส่วนใดมาบ้างกรณีที่พบให้นำเงื่อนไขการตัดสินใจเหล่านั้นระบุไว้บนต้นไม้การจำแนกภายในเครื่องหมายวงเล็บก้ามปู [ ] ดังตัวอย่างในรูปที่ 4 พบว่าก่อนส่วนการตัดสินใจ check customer status ได้ผ่านส่วนการตัดสินใจ check customer No. มาก่อนโดยผ่านทางเงื่อนไข exist ดังนั้นจึงระบุเงื่อนไข “[customer No.=exist]” ดังแสดงในรูปที่ 4




รูปที่ 4 ตัวอย่างการสร้างต้นไม้การจำแนกโดยพิจารณาส่วนการตัดสินใจ *check customer status*

**ขั้นตอนที่ 2** สร้างตารางกรณีทดสอบ

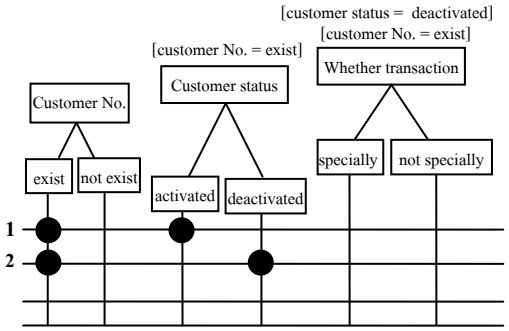
2.1) นำต้นไม้การจำแนกที่ระบุเงื่อนไขทั้งหมดมาวางเรียงกัน โดยเรียงลำดับตามจำนวนเงื่อนไขที่ระบุจากจำนวนน้อยไปหาจำนวนมาก กรณีที่จำนวนเงื่อนไขของต้นไม้การจำแนกต้นใดมีค่าเท่ากันให้วางต้นใดก่อนหรือหลังก็ได้ ตัวอย่างจากแผนภาพกิจกรรมการขายสินค้าในรูปที่ 2 สามารถสร้างต้นไม้การจำแนกที่ระบุเงื่อนไขได้ทั้งหมด 6 ต้น และรูปที่ 5 แสดงการเรียงลำดับต้นไม้การจำแนกที่ระบุเงื่อนไขตามจำนวนเงื่อนไขของต้นไม้แต่ละต้นจะพบว่าต้นไม้การจำแนก *customer No.* ไม่มีเงื่อนไขปรากฏอยู่จึงนำมาวางไว้ในตำแหน่งแรกและต้นไม้การจำแนก *compare credit limit and invoice amount* มีเงื่อนไขปรากฏมากที่สุดจึงวางไว้ในตำแหน่งสุดท้าย

2.2) สร้างตารางกรณีทดสอบโดยลากเส้นคอลัมน์จากโหนดลูกของต้นไม้การจำแนกที่ระบุเงื่อนไขแต่ละต้นและลากเส้นแถวของตารางเพื่อใช้กำหนดกรณีทดสอบ รูปที่ 5 ส่วนล่างแสดงตารางกรณีทดสอบของการขายสินค้า

**ขั้นตอนที่ 3** สร้างกรณีทดสอบ

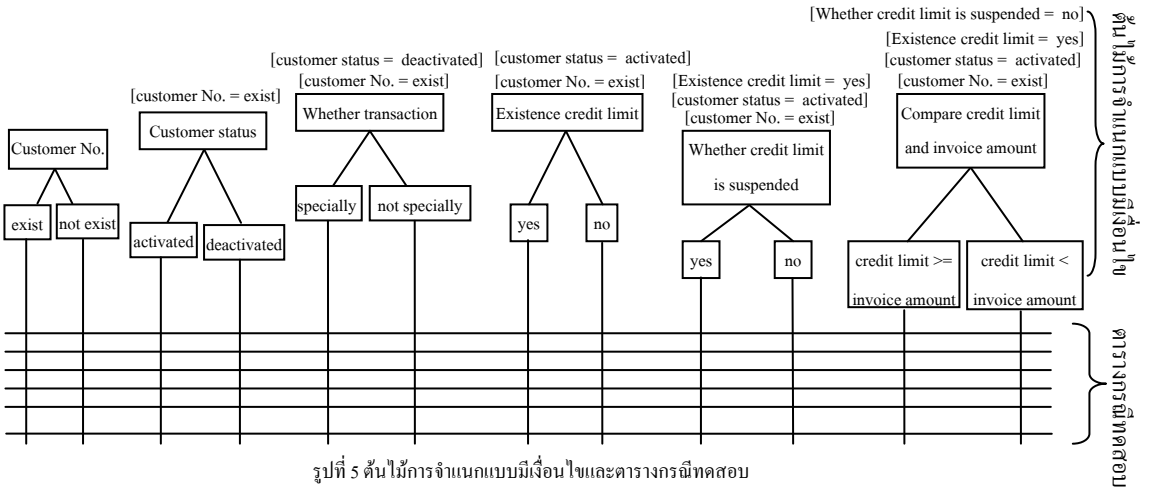
3.1) กำหนดกรณีทดสอบแต่ละกรณีโดยทำสัญลักษณ์ลงบนเส้นแถวของตาราง โดยพิจารณาจากต้นไม้การจำแนกที่มีเงื่อนไขระบุที่ละต้นจากทางด้านซ้ายไปทางด้านขวา ซึ่งเงื่อนไขจะถูกใช้เพื่อระบุว่าแต่ละโหนดในต้นไม้นั้นสามารถเลือกจับกับโหนดลูกในต้นไม้การจำแนกต้นใดได้บ้าง กรณีที่สามารถจับคู่ได้จะทำสัญลักษณ์ลงบนเส้นแถวในตารางดังตัวอย่างในรูปที่ 6 เมื่อพิจารณาเงื่อนไขบนต้นไม้การจำแนก *customer status* คือ [customer No.=exist] จะพบว่าสามารถจับคู่กับต้นไม้การจำแนก *customer No.* ซึ่งมีโหนดลูกคือ exist ดังนั้นจึงทำเครื่องหมายในตารางกรณีทดสอบโดยจับคู่โหนดลูกแต่ละโหนดของต้นไม้การจำแนก *customer status* กับโหนดลูก exist ของต้นไม้การจำแนก *customer No.* ในกรณีที่เงื่อนไขของต้นไม้การจำแนกมีความซ้ำซ้อนกับเงื่อนไขของต้นไม้ที่เคยพิจารณาแล้วให้ทำสัญลักษณ์ต่อในต้นนั้นได้เลยเพื่อลดความซ้ำซ้อน ตัวอย่างเช่นต้นไม้การจำแนก *whether transaction* มีเงื่อนไข [customer No.=exist] ปรากฏเหมือนกับต้นไม้การจำแนก *customer status* จึงทำสัญลักษณ์ต่อในเส้นกรณีทดสอบที่ 2 ดังแสดงในรูปที่ 7

3.2) เมื่อพิจารณาต้นไม้การจำแนกครบทุกต้นแล้วหากมีโหนดลูกในต้นไม้ต้นใดที่ไม่ถูกเลือกเลย ให้เลือกโหนดลูกนั้นเพียงโหนดเดียวเป็น 1 กรณี ตัวอย่างกรณีทดสอบที่ 7 ในรูปที่ 8

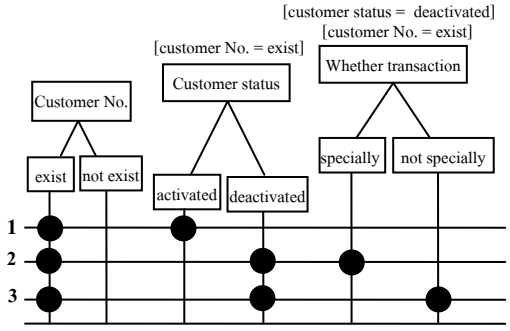



รูปที่ 6 กรณีทดสอบที่ได้จากการพิจารณาเงื่อนไขบนต้นไม้การจำแนก *customer status*

การสร้างกรณีทดสอบจากแผนภาพกิจกรรมการขายสินค้าข้างต้นแสดงรายละเอียดได้ดังรูปที่ 8 หากเปรียบเทียบจำนวนกรณีทดสอบที่ได้จากวิธีที่นำเสนอซึ่งได้ทั้งหมด 7 กรณี กับจำนวนกรณีทดสอบที่ได้จากการใช้วิธี CTM [5] ซึ่งได้กรณีทดสอบ 30 กรณี จะพบว่าจำนวนกรณีทดสอบที่ได้จากวิธีที่นำเสนอในบทความนี้มีจำนวนน้อยกว่าถึงร้อยละ 76.67 ของกรณีทดสอบที่ได้จากวิธี CTM




รูปที่ 5 ต้นไม้การจำแนกแบบมีเงื่อนไขและตารางกรณีทดสอบ




รูปที่ 7 กรณีทดสอบที่ได้จากการพิจารณาเงื่อนไข
บนต้นไม้การจำแนก *whether transaction*

## 5. การประเมินประสิทธิภาพของกรณีทดสอบ

งานวิจัยนี้เลือกใช้วิธี mutation testing เพื่อประเมินประสิทธิภาพของกรณีทดสอบที่สร้างขึ้นจากขั้นตอนที่ได้นำเสนอ วิธี mutation testing เป็นวิธีการในการทดสอบที่เริ่มจากการปรับเปลี่ยนบางส่วนของโปรแกรมต้นฉบับให้แตกต่างไปจากเดิมและเรียกโปรแกรมที่ได้จากการปรับเปลี่ยนว่าโปรแกรม mutant ในการปรับเปลี่ยนส่วนของโปรแกรมนั้นจะแก้ไขโปรแกรมเพียงเล็กน้อยครั้งละ 1 ตำแหน่ง การปรับเปลี่ยนจะกระทำโดยอาศัยตัวดำเนินการที่เรียกว่า mutation operator ซึ่งในงานวิจัยนี้จะยึดการปรับเปลี่ยนโปรแกรมตาม mutation operator ซึ่งได้เสนอไว้ในบทความเรื่อง An experimental determination of sufficient mutation operators [8] โดยโปรแกรมที่ใช้เป็นกรณีศึกษาในงานวิจัยนี้คือโปรแกรมการคำนวณค่าจอดรถและโปรแกรมการคำนวณค่างวด ซึ่งขั้นตอนในการประเมินประสิทธิภาพของกรณีทดสอบมีดังต่อไปนี้

1) ขั้นตอนการสร้างกรณีทดสอบ ซึ่งสร้างจากแผนภาพกิจกรรมที่สัมพันธ์กับโปรแกรมตัวอย่างทั้งสองโปรแกรม ซึ่งเมื่อสร้างกรณีทดสอบตามวิธีการที่นำเสนอทำให้ได้กรณีทดสอบจากโปรแกรมการคำนวณค่าจอดรถ 4 กรณีทดสอบและได้กรณีทดสอบจากโปรแกรมการคำนวณค่างวด 10 กรณีทดสอบดังตารางที่ 1

2) ขั้นตอนการสร้างโปรแกรม mutant โดยการเปลี่ยนแปลงโปรแกรมต้นฉบับซึ่งในการเปลี่ยนแปลงโปรแกรมในงานวิจัยนี้ใช้วิธีการแทนที่ (replacement) ที่เสนอในบทความ [8] ซึ่งประกอบด้วย 3 operators คือ Arithmetic Operator Replacement (AOR), Relational Operator Replacement (ROR) และ Conditional Operator Replacement (COR) เมื่อพิจารณาโปรแกรมต้นฉบับพบว่ามี operator ที่สอดคล้องกับโปรแกรมการคำนวณค่าจอดรถ 2 operator คือ AOR และ ROR ส่วน operator ที่สอดคล้องกับโปรแกรมการคำนวณค่างวดมี 3 operator ได้แก่ AOR, ROR และ COR จำนวนโปรแกรม mutant ที่ถูกสร้างสำหรับโปรแกรมการคำนวณค่าจอดรถและโปรแกรมการคำนวณค่างวดมีจำนวนเท่ากับ 4 และ 10 mutant ตามลำดับ

3) นำกรณีทดสอบที่ได้มาดำเนินการกับโปรแกรมต้นฉบับและโปรแกรม mutant ต่างๆ เพื่อเปรียบเทียบผลลัพธ์ หากผลลัพธ์ที่ได้จากโปรแกรม mutant ใดแตกต่างจากผลลัพธ์ที่ได้จากโปรแกรมต้นฉบับจะเรียกโปรแกรม mutant นั้นว่า killed mutant ซึ่งหมายความว่ากรณีทดสอบที่ใช้ในการดำเนินการกับโปรแกรม mutant สามารถตรวจจับข้อผิดพลาดในโปรแกรม mutant นั้นได้ และอัตราส่วนของจำนวน killed mutant ต่อจำนวนโปรแกรม mutant ทั้งหมดจะแสดงถึงประสิทธิภาพของกรณีทดสอบที่สร้างขึ้นซึ่งจะแสดงเป็นค่า mutation score หาก mutation score มีค่าเข้าใกล้ 1 หมายถึงกรณีทดสอบนั้นมีประสิทธิภาพมากและหาก mutation score มีค่าเข้าใกล้ศูนย์หมายถึงกรณีทดสอบนั้นมีประสิทธิภาพค่อนข้างต่ำ ตารางที่ 1 แสดงผลการประเมินประสิทธิภาพของกรณีทดสอบที่สร้างจากแผนภาพกิจกรรมโดยวิธี mutation testing

ตาราง 1. ผลการประเมินประสิทธิภาพของกรณีทดสอบโดยวิธี mutation testing

| Program | Total test cases | Total mutant | Killed mutant | Mutation score |
|---|---|---|---|---|
| คำนวณค่าจอดรถ | 4 | 77 | 64 | 0.83 |
| คำนวณค่างวด | 10 | 55 | 48 | 0.87 |

จากผลการประเมินประสิทธิภาพของกรณีทดสอบดังกล่าว พบว่าโปรแกรม mutant ที่ให้ผลลัพธ์แตกต่างกับผลลัพธ์จากโปรแกรมต้นฉบับหรือ killed mutant ของโปรแกรมคำนวณค่าจอดรถมีทั้งหมด 64 mutant และจากโปรแกรมคำนวณค่างวดมีทั้งหมด 48 mutant ซึ่งสามารถหา mutation score ได้เท่ากับ 0.83 และ 0.87 ตามลำดับ จากผลการทดลองที่ได้ แสดงให้เห็นว่ากรณีทดสอบที่สร้างจากขั้นตอนที่นำเสนอเป็นกรณีทดสอบที่มีประสิทธิภาพสูง

## 6. สรุปผลและอภิปราย

การทดสอบซอฟต์แวร์เป็นขั้นตอนที่มีค่าใช้จ่ายสูงโดยเฉพาะ การสร้างกรณีทดสอบ หากการทดสอบซอฟต์แวร์กระทำได้เร็วและมีจำนวนกรณีทดสอบที่ไม่มากจนเกินไปจะสามารถช่วยลดค่าใช้จ่ายในขั้นตอนการทดสอบซอฟต์แวร์ลงได้เป็นจำนวนมาก งานวิจัยนี้จึงนำเสนอการสร้างกรณีทดสอบจากแบบจำลองที่ได้จากขั้นตอนการออกแบบซอฟต์แวร์คือแผนภาพกิจกรรมโดยการวิเคราะห์ input domain จากแผนภาพกิจกรรมแล้วสร้างต้นไม้การจำแนกแบบมีเงื่อนไขและสร้างตารางกรณีทดสอบเพื่อให้ได้กรณีทดสอบในท้ายที่สุด ซึ่งประโยชน์ที่ได้รับจากวิธีที่นำเสนอคือ 1) สามารถเริ่มต้นการทดสอบซอฟต์แวร์ได้ตั้งแต่ขั้นตอนการออกแบบ

2) ช่วยลดเวลาในการวิเคราะห์เอกสารความต้องการ เนื่องจากสามารถทำได้โดยอัตโนมัติจากแผนภาพกิจกรรม 3) ช่วยลดความซับซ้อนในการสร้างกรณีทดสอบเนื่องจากมีการระบุเงื่อนไขในการเลือก input domain ที่จะเป็นกรณีทดสอบจึงทำให้กรณีทดสอบที่ได้มีเฉพาะกรณีที่เป็นไปได้เท่านั้น โดยผลจากการใช้วิธีที่นำเสนอกับตัวอย่างแผนภาพกิจกรรมการขายสินค้า [5] พบว่าสามารถสร้างกรณีทดสอบจากแผนภาพกิจกรรมได้โดยตรงและจำนวนกรณีทดสอบที่ได้มีจำนวนน้อยกว่ากรณีทดสอบที่สร้างด้วยวิธี CTM [5] และเมื่อนำกรณีทดสอบที่สร้างขึ้นจากวิธีการที่นำเสนอไปทำการประเมินประสิทธิภาพโดยวิธี mutation testing พบว่า mutation score ที่ได้แสดงให้เห็นว่ากรณีทดสอบที่สร้างจากวิธีการที่นำเสนอเป็นกรณีทดสอบที่มีประสิทธิภาพสูง ในอนาคตผู้วิจัยจะทำการพัฒนาเครื่องมือต้นแบบในการสร้างกรณีทดสอบตามวิธีที่นำเสนอเพื่อช่วยให้การสร้างกรณีทดสอบมีความสะดวกและรวดเร็วขึ้น

## เอกสารอ้างอิง


[1] P.Ammann and J.Offutt, “Introduction to Software Testing”, Cambridge University Press, USA, 2008.

[2] Object Management Group, “Unified modeling language”, Specification v1.5 formal/2003-03-01, Object Management Group, Mar. 2003 .

[3] T. Y. Chen , Pak-Lok Poon , Sau-Fun Tang and T. H. Tse, “Identification of Categories and Choices in Activity Diagrams”, In Proceedings of the 5th International Conference on Quality Software (QSIC 2005), IEEE Computer Society, September. 2005.

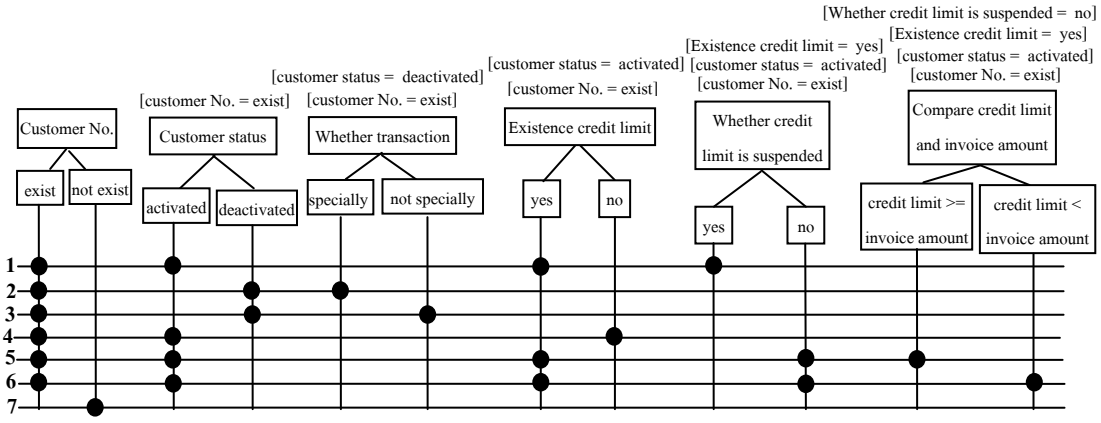
[4] W. Sinzhang, Y. Jiesong, Y. Xiaofeng, H. Jun, L. Xuandong, and Z. Guoliang, “Generating Test Cases from UML Activity Diagram based on Gray-Box Method”, In Proceedings of the 11th Asia-Pacific Software Engineering Conference(APSEC), IEEE Computer Society, November. 2004.

[5] T.Y.Chen and P.L.Poon, “Teaching black box testing”, in Proceedings of the 1998 International Conference on Software engineering: Education and Practice, IEEE Computer Society Press, pp. 324-329, January. 1998.

[6] Grochtmann, M., Grimm, K. and Wegener, J, “Tool – Supported Test Case Design for Black-Box Testing by Means of the Classification – Tree Editor”, EuroSTAR ’93 1 st European International Conference on Software Testing Analysis and Review, London UK, pp. 25 – 28, October. 1993.

[7] M. Grochtmann, J. Wegener, K. Grimm, “Test case design using classification trees and the classification-tree editor CTE”, in: Proceedings of the 8th International Software Quality Week, 1995, pp. 1 - 11.

[8] A. J. Offutt, Ammei Lee, G. Rothermel, R. Untch, and C. Zapf. An experimental determination of sufficient mutation operators. ACM Transaction on Software Engineering Methodology, pp. 99 - 118, April. 1996.


รูปที่ 8 การสร้างกรณีทดสอบจากแผนภาพกิจกรรมการขายสินค้าโดยใช้หลักการต้นไม้การจำแนกแบบมีเงื่อนไข