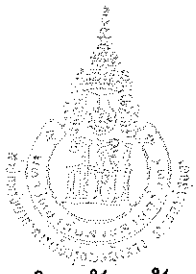



แบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไขสำหรับสร้างกรณีทดสอบบนพื้นฐานของแผนภาพกิจกรรมของ UML

A Condition - Classification Tree Model for Generating Test Cases based on UML Activity Diagrams

ปรัชญานีย์ ไทยเกิด

Pratyanee Thaikerd

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญา
วิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์
มหาวิทยาลัยสงขลานครินทร์

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science Prince of Songkla University

2553

ชื่อวิทยานิพนธ์ แบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไขสำหรับสร้างกรณีทดสอบบนพื้นฐานของแผนภาพกิจกรรมของ UML

ผู้เขียน นางสาวปรัชญานีย์ ไทยเกิด

สาขาวิชา วิทยาการคอมพิวเตอร์

อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก

.....
(ดร.สุภาภรณ์ กานต์สมเกียรติ)

คณะกรรมการสอบ

.....ประธานกรรมการ
(ดร. กฤดาภัทร สีหารี)

.....กรรมการ
(ดร.สุภาภรณ์ กานต์สมเกียรติ)

.....กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.อำนาจ เปาะทอง)

บัณฑิตวิทยาลัย มหาวิทยาลัยสงขลานครินทร์ อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษา ตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต สาขาวิชาวิทยาการคอมพิวเตอร์

.....
(ศาสตราจารย์ ดร.อมรรัตน์ พงศ์ดารา)
คณบดีบัณฑิตวิทยาลัย

ชื่อวิทยานิพนธ์	แบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไขสำหรับสร้างกรณีทดสอบบนพื้นฐานของแผนภาพกิจกรรมของ UML
ผู้เขียน	นางสาวปรัชญานีย์ ไทยเกิด
สาขาวิชา	วิทยาการคอมพิวเตอร์
ปีการศึกษา	2553

บทคัดย่อ

ประสิทธิภาพของการทดสอบซอฟต์แวร์ขึ้นอยู่กับคุณภาพความครอบคลุมของข้อมูลนำเข้าที่ใช้ในการทดสอบ ในปัจจุบันซอฟต์แวร์ส่วนใหญ่ถูกพัฒนาขึ้นโดยใช้เทคโนโลยีเชิงวัตถุ ภาษาการออกแบบเชิงโมเดล (Unified Modeling Language: UML) เป็นภาษาหนึ่งที่ถูกใช้อย่างกว้างขวางในการออกแบบซอฟต์แวร์เชิงวัตถุ งานวิจัยนี้พิจารณาแผนภาพกิจกรรม (activity diagram) ซึ่งเป็นแผนภาพประเภทหนึ่งของ UML แผนภาพนี้ถูกใช้เพื่อแสดงพฤติกรรมของซอฟต์แวร์ ดังนั้นการนำแผนภาพแสดงพฤติกรรมนี้มาใช้ในการสร้างกรณีทดสอบจะทำให้สามารถสร้างกรณีทดสอบได้ในขั้นตอนเริ่มต้นของขั้นตอนการออกแบบซอฟต์แวร์ วิทยานิพนธ์นี้เสนอแบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไขสำหรับสร้างกรณีทดสอบจากแผนภาพกิจกรรม แบบจำลองนี้ช่วยให้กรณีทดสอบที่สร้างขึ้นมีประสิทธิภาพและช่วยลดเวลาในการวิเคราะห์เอกสารความต้องการ ผลลัพธ์ที่ได้จากการทดลองแสดงให้เห็นว่าหลักการที่ได้นำเสนอสามารถช่วยให้กรณีทดสอบที่สร้างมีจำนวนน้อย มีความเหมาะสมและสามารถกระทำได้ตั้งแต่ช่วงต้นของกระบวนการพัฒนาซอฟต์แวร์

Thesis Title	A Condition - Classification Tree Model for Generating Test Cases based on UML Activity Diagrams
Author	Miss Pratyanee Thaikerd
Major Program	Computer Science
Academic Year	2010

ABSTRACT

The effectiveness of testing depends on the quality of the coverage of the test inputs. Most software today is developed using object-oriented technology. Many languages have been developed to support object-oriented software design. Most notably the Unified Modeling Language (UML). The Unified Modeling Language (UML) is now widely used to describe object – oriented design. This thesis focuses on one type of UML diagram, the activity diagram, which is used to model the behavior of software. If test cases can be generated from this model, tests can be generated early in the process, during software design, greatly increasing the chances of finishing all testing before the software is delivered. This thesis proposes a condition – classification tree model for generating test cases from activity diagrams. The model helps to generate effective test cases and helps reduce the time needed to analyze the software specifications. Results from an experiment show that the proposed approach can help generate a relatively small number of test cases at reasonable cost, early in the development process.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จได้ด้วยความช่วยเหลือและสนับสนุนจากบุคคลหลายฝ่าย ผู้วิจัยรู้สึกซาบซึ้งและขอกราบขอบพระคุณอย่างสูง คือ

ดร.สุภาภรณ์ กานต์สมเกียรติ อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่กรุณาให้คำปรึกษาแนะนำ และช่วยเหลือในการแก้ปัญหาต่าง ๆ ให้แก่ผู้วิจัยเสมอมา พร้อมทั้งตรวจทานและแก้ไขวิทยานิพนธ์ให้แก่ผู้วิจัย

ดร.กฤดาภัทร สีหารี ประธานกรรมการสอบวิทยานิพนธ์ ที่กรุณาให้ข้อเสนอแนะและช่วยตรวจทานแก้ไขวิทยานิพนธ์ให้มีความสมบูรณ์

ผู้ช่วยศาสตราจารย์ ดร.อำนาจ เปาะทอง กรรมการในการสอบวิทยานิพนธ์ที่กรุณาให้ข้อเสนอแนะในการทำวิจัย รวมทั้งตรวจทานแก้ไขวิทยานิพนธ์

อาจารย์ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยสงขลานครินทร์ทุกท่านที่ให้ความรู้ทางด้านวิชาการ ซึ่งสามารถนำมาใช้ในการทำวิทยานิพนธ์ได้อย่างดียิ่ง

เจ้าหน้าที่ภาควิชาวิทยาการคอมพิวเตอร์ และเจ้าหน้าที่บัณฑิตวิทยาลัยทุกท่านที่ให้ความช่วยเหลือ และอำนวยความสะดวกเกี่ยวกับเอกสารต่าง ๆ

โครงการส่งเสริมการผลิตครูที่มีความสามารถพิเศษทางวิทยาศาสตร์และคณิตศาสตร์ (โครงการ สควค.) ที่มอบทุนสนับสนุนการศึกษาและทุนสนับสนุนการวิจัย

เพื่อน ๆ พี่ ๆ และน้อง ๆ ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ที่ให้คำปรึกษา และช่วยเหลือในการทำวิทยานิพนธ์

คุณพ่อ คุณแม่ และครอบครัว ที่ให้การสนับสนุนคอยเป็นห่วงสุขภาพและให้กำลังใจแก่ผู้วิจัยมาโดยตลอด

ผู้วิจัยขอขอบคุณทุกท่านเป็นอย่างสูงมา ณ โอกาสนี้

ปรัชญานีย์ ไทยเกิด

สารบัญ

สารบัญ (ต่อ)

รายการตาราง

รายการภาพประกอบ

รายการภาพประกอบ (ต่อ)

รายการภาพประกอบ (ต่อ)

บทที่ 1

บทนำ

1.1 ความสำคัญและที่มาของงานวิจัย

ปัจจุบันการพัฒนาซอฟต์แวร์เชิงวัตถุ (object - oriented software development) กำลังได้รับความนิยมเป็นอย่างมาก เนื่องจากแนวคิดเชิงวัตถุเป็นแนวคิดที่ใกล้เคียงกับความเป็นจริงทำให้สามารถพัฒนาระบบที่มีความซับซ้อนและมีขนาดใหญ่ได้ นอกจากความสะดวกรวดเร็วในการพัฒนาแล้วยังสามารถนำกลับมาแก้ไขเพิ่มเติมในภายหลังได้ง่าย เนื่องจากความนิยมที่เพิ่มมากขึ้นจึงมีการสร้างเครื่องมือมาช่วยในการวิเคราะห์และออกแบบระบบซอฟต์แวร์เชิงวัตถุเพื่อให้ดำเนินงานได้อย่างสะดวกและรวดเร็ว เช่น Unified Modeling Language (UML) ซึ่งเป็นภาษาที่ช่วยในการสร้างแบบจำลองเชิงวัตถุที่ประกอบด้วยแผนภาพหลายชนิดที่สามารถจำลองการออกแบบซอฟต์แวร์ทั้งในเชิงโครงสร้าง (structure) และเชิงพฤติกรรม (behavior) หนึ่งในแผนภาพของ UML คือแผนภาพกิจกรรม (activity diagram) ซึ่งเป็นแผนภาพที่ใช้อธิบายขั้นตอนของกิจกรรมการทำงานของซอฟต์แวร์ตั้งแต่เริ่มต้นจนสิ้นสุด

การทดสอบซอฟต์แวร์ เป็นขั้นตอนที่สำคัญในกระบวนการผลิตซอฟต์แวร์ ประสิทธิภาพของการทดสอบส่วนใหญ่จะขึ้นอยู่กับความเหมาะสมและประสิทธิภาพของกรณีทดสอบที่สร้างขึ้น ยิ่งซอฟต์แวร์มีขนาดใหญ่และมีความซับซ้อนมากขึ้น การทดสอบยิ่งต้องใช้เวลาและเสียค่าใช้จ่ายสูงขึ้นด้วย การสร้างกรณีทดสอบเป็นขั้นตอนที่สำคัญในการทดสอบซอฟต์แวร์ ซึ่งจะเป็นปัจจัยที่ส่งผลต่อประสิทธิภาพที่จะเกิดขึ้นในการทดสอบ กรณีทดสอบที่ดีนั้นจะต้องมีความครอบคลุมความต้องการของระบบและมีจำนวนกรณีทดสอบที่ไม่มากจนเกินไป เพื่อเป็นการประหยัดค่าใช้จ่ายและเวลาในการทดสอบ ในปัจจุบันมีเทคนิคที่ได้รับความนิยมในการทดสอบซอฟต์แวร์อยู่ 2 เทคนิค คือ เทคนิค white–box และเทคนิค black-box โดยเทคนิค white–box จะสร้างกรณีทดสอบจากชุดคำสั่ง (code) ของโปรแกรม สำหรับเทคนิค black-box จะสร้างกรณีทดสอบจาก requirements specification และจากการออกแบบของซอฟต์แวร์

ในปัจจุบันมีงานวิจัยที่นำเสนอการสร้างกรณีทดสอบจาก specification ที่แสดงในรูปแบบของแผนภาพกิจกรรมของ UML โดยอาศัยหลักการครอบคลุมเส้นทางการทำงานและขั้นตอนต่างๆ ที่ปรากฏในแผนภาพกิจกรรม การสร้างกรณีทดสอบดังกล่าวทำให้ได้เส้นทางการ

ทำงานและขั้นตอนต่างๆ จำนวนมาก มีผลให้การพิจารณาและเลือกกรณีทดสอบต้องใช้เวลานาน ดังนั้นงานวิจัยนี้จึงเสนอแบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไข (Condition-Classification Tree Model: CCTM) สำหรับสร้างกรณีทดสอบโดยใช้แผนภาพกิจกรรมของ UML วิธีการที่ได้นำเสนอนี้ช่วยให้ 1) การทดสอบเริ่มต้นได้ตั้งแต่ขั้นตอนการออกแบบโดยใช้ specification 2) ช่วยลดเวลาในการวิเคราะห์เอกสารความต้องการ เนื่องจากสามารถทำได้โดยอัตโนมัติจากแผนภาพกิจกรรม 3) ช่วยลดความซับซ้อนในการสร้างกรณีทดสอบเนื่องจากมีการระบุเงื่อนไข ซึ่งจะทำให้กรณีทดสอบที่มีความเป็นไปได้ต่ำถูกกำจัดออก ทำให้ได้จำนวนกรณีทดสอบลดน้อยลง

1.2 งานวิจัยที่เกี่ยวข้อง

Generating Test Cases from UML Activity Diagram based on Gray-Box Method (Sinzhang, *et al.*, 2004)

งานวิจัยนี้นำเสนอการสร้างกรณีทดสอบจากแผนภาพกิจกรรม (activity diagram) ที่อาศัยเทคนิค gray-box ซึ่งเป็นวิธีที่ผสมผสานทั้ง black-box และ white-box ร่วมกันดังนี้

1) สร้าง activity diagram จาก operation ของซอฟต์แวร์
2) สร้าง test scenarios จาก activity diagram ที่ได้
3) สร้าง test cases จาก test scenarios กรณีทดสอบแต่ละกรณีคือเส้นทางที่จะทดสอบและ input, output ที่เป็นไปได้ในเส้นทางนั้น

นอกจากนั้นกลุ่มผู้วิจัยได้นำหลักการที่เสนอไปพัฒนาต้นแบบเครื่องมือเพื่อสนับสนุนการทำงานตามหลักการดังกล่าวอีกด้วย

Tool – Supported Test Case Design for Black-Box Testing by Means of the Classification – Tree Editor (Grochtmann, *et al.*, 1993)

งานวิจัยนี้นำเสนอเครื่องมือในการสร้างกรณีทดสอบแบบอัตโนมัติที่เรียกว่า CTE (Classification - Tree Editor) ซึ่งพัฒนามาจากหลักการ classification tree method ซึ่งเป็นหลักการที่สร้างกรณีทดสอบโดยอาศัยเทคนิค black–box ผลการประเมินเครื่องมือที่ได้นำเสนอนี้พบว่าเป็นเครื่องมือที่อำนวยความสะดวกในการสร้างกรณีทดสอบและส่งผลให้กรณีทดสอบมีประสิทธิภาพสูงขึ้นด้วย

Teaching Black Box Testing (Chen and Poon, 1998)

งานวิจัยนี้นำเสนอการสร้างกรณีทดสอบแบบอาศัยเทคนิค black-box โดยใช้วิธี classification tree method เพื่อตรวจสอบว่าวิธีการนี้ใช้งานง่ายไม่ซับซ้อนจึงได้นำมาสอนให้แก่นักศึกษาระดับปริญญาตรี ภาควิชาวิทยาการคอมพิวเตอร์ในมหาวิทยาลัย Melbourne ผลลัพธ์ที่ได้เป็นที่น่าพอใจมากซึ่งนักศึกษาร้อยละ 80 เห็นว่าวิธีการนี้สามารถนำไปปฏิบัติใช้ได้จริงโดยมีรายละเอียดขั้นตอนดังแสดงในหัวข้อ 2.3 classification tree method

Identification of Categories and Choices in Activity Diagrams (Chen, *et al.*, 2005)

งานวิจัยนี้นำเสนอวิธีในการระบุ category และ choice ใน choice relation table โดยการพิจารณาส่วนการตัดสินใจและเงื่อนไขต่างๆ ในแผนภาพกิจกรรมเพื่อใช้ในการเลือก choice แต่ละคู่ใน choice relation table เพื่อใช้เป็นกรณีทดสอบ ซึ่งวิธีนี้ช่วยให้การเลือก choice ในแต่ละ category เพื่อใช้เป็นกรณีทดสอบมีความถูกต้องและมีความเป็นไปได้สูง

1.3 วัตถุประสงค์ของงานวิจัย

1.3.1 เพื่อนำเสนอแบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไข (Condition-Classification Tree Model: CCTM) สำหรับสร้างกรณีทดสอบบนพื้นฐานของแผนภาพกิจกรรมของ UML

1.3.2 เพื่อพัฒนาเครื่องมือสำหรับสร้างกรณีทดสอบโดยใช้แบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไข

1.4 ขอบเขตการดำเนินงานของการวิจัย

ขอบเขตของการดำเนินงานการวิจัยมีรายละเอียดดังนี้

1.4.1 นำเสนอแบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไข (Condition-Classification Tree Model: CCTM) สำหรับสร้างกรณีทดสอบบนพื้นฐานของแผนภาพกิจกรรมของ UML

1.4.2 การสร้างแบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไข (Condition-Classification Tree Model: CCTM) จะพิจารณา input domain และเงื่อนไขต่างๆ จากตำแหน่งการตัดสินใจ (decision points) และเงื่อนไข (conditions) ของแผนภาพกิจกรรมของ UML

1.4.3 พัฒนาเครื่องมือสำหรับสร้างกรณีทดสอบโดยใช้แบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไข

1.5 ขั้นตอนและระยะเวลาการดำเนินงาน

1.5.1 ขั้นตอนการดำเนินงาน

1) ศึกษางานวิจัยและเอกสารที่เกี่ยวข้อง

2) เสนอแบบจำลอง

3) พัฒนาเครื่องมือสนับสนุนการทำงานของแบบจำลอง

4) ทดสอบวิธีการที่นำเสนอ

5) สรุปผลและเขียนบทความวิจัยเพื่อเผยแพร่

6) จัดทำรายงานการวิจัย

1.5.2 ระยะเวลาการดำเนินงาน

มิถุนายน 2552 – กันยายน 2553

1.5.3 แผนการดำเนินการวิจัย

ระยะเวลาการดำเนินการวิจัยแสดงดังตารางที่ 1.1

ตารางที่ 1.1 ระยะเวลาการดำเนินการวิจัย

กิจกรรม/ขั้นตอนการดำเนินงาน	เดือน															
	2552							2553								
	6	7	8	9	10	11	12	1	2	3	4	5	6	7	8	9
1) ศึกษางานวิจัยและเอกสารที่เกี่ยวข้อง																
2) เสนอแบบจำลอง																
3) พัฒนาเครื่องมือสนับสนุนการทำงานของแบบจำลอง																
4) ทดสอบวิธีการที่นำเสนอ																
5) สรุปผลและเขียนบทความวิจัยเพื่อเผยแพร่																
6) จัดทำรายงานการวิจัย																

1.6 สถานที่และเครื่องมือที่ใช้

1.6.1 สถานที่

ห้องวิจัยกลุ่มวิศวกรรมซอฟต์แวร์และงานประยุกต์ ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยสงขลานครินทร์

1.6.2 เครื่องมือที่ใช้

1) ด้านฮาร์ดแวร์

เครื่องคอมพิวเตอร์ส่วนบุคคลที่มีคุณลักษณะ ได้แก่ หน่วยความจำขนาด 2 GB และฮาร์ดดิสก์ความจุ 320 GB สำหรับพัฒนาและทดสอบระบบ

2) ด้านซอฟต์แวร์

2.1) ระบบปฏิบัติการ Microsoft Windows XP

2.2) เครื่องมือสนับสนุนการทำงาน Microsoft Visual Basic 2008 Express Edition

1.7 ประโยชน์ที่คาดว่าจะได้รับ

1.7.1 ได้แบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไข (Condition - Classification Tree Model: CCTM) สำหรับสร้างกรณีทดสอบบนพื้นฐานของแผนภาพกิจกรรมของ UML

1.7.2 ได้เครื่องมือสำหรับสร้างกรณีทดสอบโดยใช้แบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไข

บทที่ 2

ทฤษฎีที่เกี่ยวข้อง

เนื้อหาในบทนี้กล่าวถึงทฤษฎีและหลักการต่างๆ โดยส่วนแรกเกี่ยวข้องกับการทดสอบซอฟต์แวร์ ระดับการทดสอบซอฟต์แวร์ ขั้นตอนการทดสอบซอฟต์แวร์ เทคนิคการทดสอบซอฟต์แวร์และการประเมินประสิทธิภาพของกรณีทดสอบ ในส่วนที่สองเป็นเนื้อหาเกี่ยวกับแผนภาพกิจกรรมของภาษาการออกแบบเชิงวัตถุ และส่วนที่สามเป็นการเสนอเนื้อหาเกี่ยวกับหลักการสร้างกรณีทดสอบแบบ Classification Tree Method (CTM) ซึ่งมีรายละเอียดดังต่อไปนี้

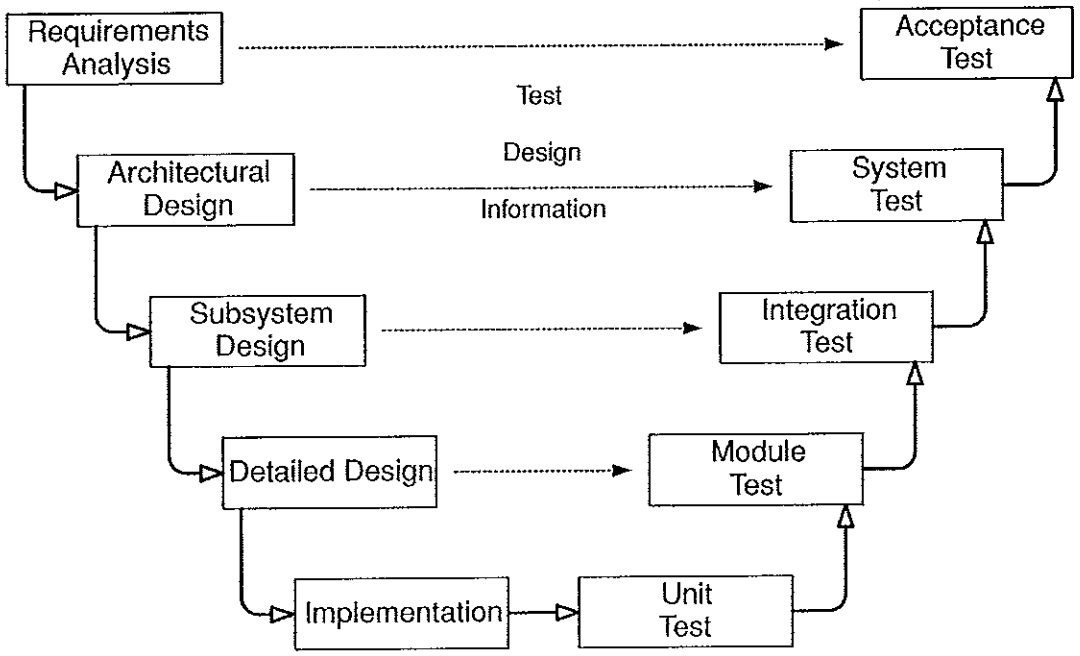
2.1 การทดสอบซอฟต์แวร์ (Software Testing)

การทดสอบซอฟต์แวร์เป็นกิจกรรมที่จัดทำขึ้นเพื่อประเมินและปรับปรุงคุณภาพของซอฟต์แวร์ (Ammann and Offutt, 2008) โดยการตรวจหาข้อผิดพลาดและปัญหาที่เกิดขึ้นแล้วทำการแก้ไขข้อผิดพลาดหรือปัญหาดังกล่าวให้ถูกต้อง วัตถุประสงค์ของการทดสอบซอฟต์แวร์คือการพิสูจน์ว่าซอฟต์แวร์ทำงานได้ครบทุกฟังก์ชันตามข้อกำหนดความต้องการ

2.1.1 ระดับการทดสอบซอฟต์แวร์

การทดสอบซอฟต์แวร์แต่ละระดับจะทำควบคู่ไปกับขั้นตอนการพัฒนาซอฟต์แวร์ซึ่งสามารถแสดงให้เห็นถึงระดับการทดสอบที่เกี่ยวข้องกับกิจกรรมการพัฒนาซอฟต์แวร์ได้โดยการใช้แผนภาพวี หรือ V Model ดังภาพประกอบที่ 2.1 ซึ่งสามารถแบ่งระดับการทดสอบออกตามขั้นตอนการพัฒนาซอฟต์แวร์ได้ 5 ระดับ (Ammann and Offutt, 2008) ดังต่อไปนี้

1) การทดสอบการยอมรับ (acceptance testing) เป็นการทดสอบเพื่อประเมินความสมบูรณ์และประสิทธิภาพของระบบว่าสามารถทำงานได้ถูกต้องตามความต้องการและเป็นที่พึงพอใจของผู้ใช้หรือไม่ ซึ่งการทดสอบจะสอดคล้องกับขั้นตอนการวิเคราะห์ความต้องการของผู้ใช้ (requirements analysis)



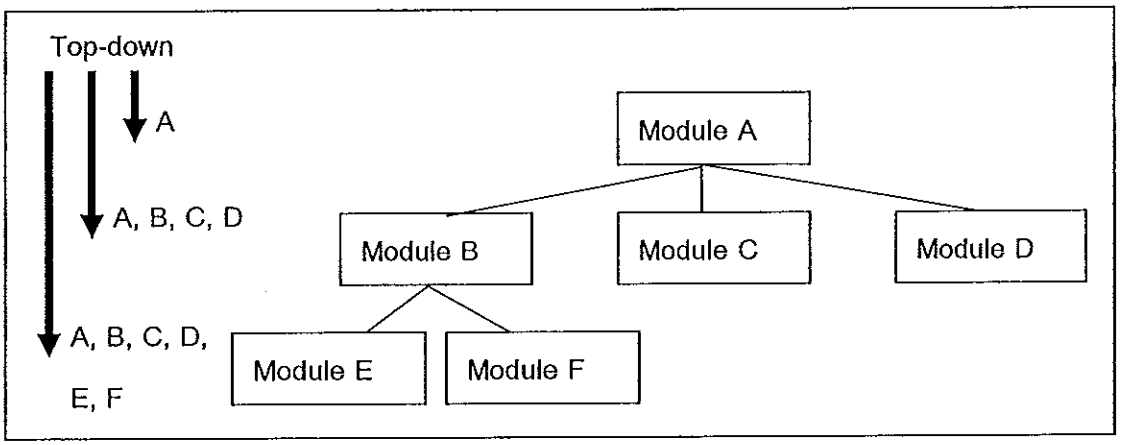
ภาพประกอบที่ 2.1 แผนภาพระดับการทดสอบซอฟต์แวร์และขั้นตอนการพัฒนาซอฟต์แวร์ V Model (Ammann and Offutt, 2008)

2) การทดสอบระบบ (system testing) เป็นการทดสอบการทำงานของระบบโดยรวมซึ่งเกิดจากการรวมกันของระบบย่อยหลายๆ ส่วนว่าสามารถทำงานได้ตรงตาม specification ที่กำหนดไว้หรือไม่ ซึ่งจะสอดคล้องกับขั้นตอนการออกแบบสถาปัตยกรรมของซอฟต์แวร์ (architectural design)

3) การทดสอบระดับรวมหน่วย (integration testing) เป็นการทดสอบการทำงานของกลุ่มโปรแกรมหรือส่วนประกอบย่อยที่ถูกประสานเข้าด้วยกัน (Myers, 2004) เพื่อทำงานหน้าที่ใดหน้าที่หนึ่งร่วมกัน ถึงแม้ว่าแต่ละโปรแกรมย่อยจะผ่านการทดสอบมาแล้วก็ตาม แต่เมื่อต้องมาทำงานร่วมกันอาจเกิดข้อผิดพลาดบางประการขึ้นได้ จึงต้องมีการทดสอบการรวมหน่วยเพื่อค้นหาข้อผิดพลาดดังกล่าว จึงกล่าวได้ว่าการทดสอบระดับรวมหน่วยเป็นการทดสอบส่วนประสานการทำงานระหว่างหน่วยย่อยนั่นเอง ดังนั้นสิ่งที่จะถูกทดสอบในระดับนี้มี 2 อย่าง คือ ส่วนประสานและผลการทำงานของหน่วยรวม การทดสอบระดับรวมหน่วยสามารถทำได้ 2 ลักษณะ คือ แบบรวมหน่วยทั้งหมดและทดสอบในครั้งเดียว (big bang) และแบบเพิ่มทีละหน่วยหรือโมดูล (incremental) แต่การทดสอบแบบ big bang เป็นการทดสอบที่มีความเสี่ยงต่อการเกิดข้อผิดพลาดสูงมาก เนื่องจากการทดสอบจัดทำขึ้นเพียงครั้งเดียวหลังจากการประสาน

หน่วยย่อยหรือโมดูลย่อยทุกโมดูลเข้าด้วยกัน ดังนั้นจึงนิยมทดสอบแบบเพิ่มทีละโมดูลมากกว่า เนื่องจากการทดสอบจัดทำขึ้นทุกครั้งที่มีการประสานโมดูลเพิ่ม สำหรับวิธีการทดสอบแบบเพิ่มทีละหน่วยมีให้เลือกใช้ 2 วิธี ได้แก่

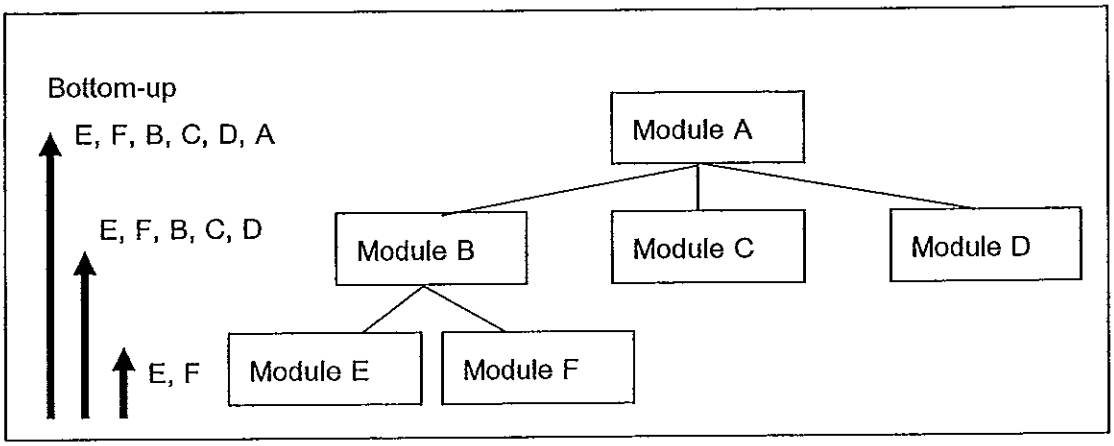
- การทดสอบแบบเพิ่มโมดูลจากบนลงล่าง (top - down approach) เป็นการทดสอบโดยเพิ่มทีละโมดูลจากบนลงล่างตามลำดับโครงสร้างควบคุม นั่นคือโมดูลที่อยู่ระดับบนจะเรียกใช้โมดูลที่อยู่ระดับล่าง (เป็นโมดูลหลักและโมดูลย่อย) ดังแสดงตัวอย่างในภาพประกอบที่ 2.2



ภาพประกอบที่ 2.2 ตัวอย่างการทดสอบแบบเพิ่มโมดูลจากบนลงล่าง (top - down approach)

จากภาพประกอบที่ 2.2 โมดูล A คือโมดูลหลักทำหน้าที่เรียกใช้โมดูลย่อยอื่นๆ เมื่อเริ่มทดสอบจะเพิ่มโมดูล B เข้ามาก่อนแล้วทดสอบรวมกับโมดูล A จากนั้นจะเลือกว่าจะเพิ่มโมดูลในแนวนอนหรือแนวตั้ง หากเลือกเพิ่มโมดูลในแนวนอนจะต้องเพิ่มโมดูล C เพื่อมาทดสอบรวมกับโมดูล A และ B แล้วเพิ่มโมดูล D เพื่อทดสอบรวมอีกครั้ง และเพิ่มโมดูลย่อย E และ F ตามลำดับ แต่หากเลือกเพิ่มโมดูลในแนวตั้งหลังจากเพิ่มโมดูล B แล้วจะต้องเพิ่มโมดูล E, F, C และ D ตามลำดับ

- การทดสอบแบบเพิ่มโมดูลจากล่างขึ้นบน (bottom - up approach) จะทดสอบโดยเริ่มจากโมดูลล่างสุดก่อนโดยการรวมโมดูลที่อยู่ระดับล่างสุดเข้าด้วยกันแล้วทดสอบ จากนั้นเพิ่มโมดูลเข้ามาทีละโมดูลแล้วทำการทดสอบ ทำเช่นนี้ไปเรื่อยๆ จนกว่าจะครบทุกโมดูล ดังแสดงตัวอย่างในภาพประกอบที่ 2.3



ภาพประกอบที่ 2.3 ตัวอย่างการทดสอบแบบเพิ่มโมดูลจากล่างขึ้นบน (bottom-up approach)

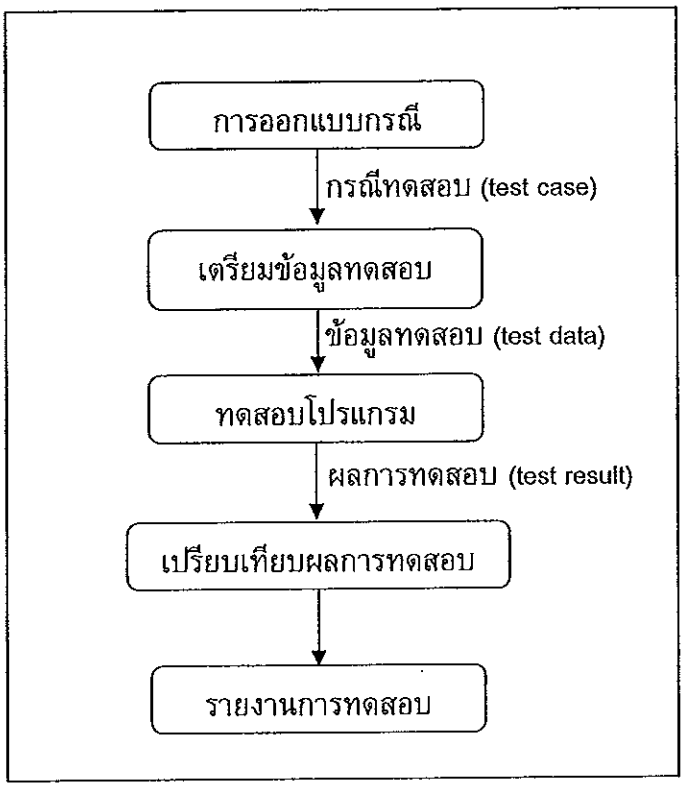
จากภาพประกอบที่ 2.3 เริ่มต้นโดยการทดสอบโมดูล E รวมกับโมดูล F ก่อนแล้วเพิ่มโมดูล B เพื่อมาทดสอบรวมกับโมดูล E และ F แล้วเพิ่มโมดูล C เพื่อทดสอบรวมอีกครั้ง และเพิ่มโมดูลย่อย D และ A เข้ามาทดสอบทีละโมดูลตามลำดับ

4) การทดสอบระดับโมดูล (module testing) เป็นการทดสอบการทำงานแต่ละโมดูลว่าสามารถทำงานได้ถูกต้องหรือไม่ การทดสอบระดับโมดูลนี้จะสอดคล้องกับขั้นตอนการออกแบบรายละเอียด (detailed design)

5) การทดสอบระดับหน่วย (unit testing) เป็นการทดสอบระดับย่อยที่สุดเพื่อประเมินการทำงานของแต่ละหน่วยย่อยว่าสามารถทำงานได้ถูกต้องหรือไม่ ซึ่งจะเกี่ยวข้องกับขั้นตอนการดำเนินงาน (implementation)

2.1.2 ขั้นตอนการทดสอบซอฟต์แวร์

โดยทั่วไปการทดสอบซอฟต์แวร์จะเริ่มต้นจากการออกแบบ "กรณีทดสอบ" ซึ่งหมายถึงการกำหนดข้อมูลนำเข้าและผลลัพธ์ที่คาดว่าจะได้รับจากการทำงานของโปรแกรมในสถานการณ์สมมติต่าง ๆ จากนั้นนำกรณีทดสอบที่ออกแบบไว้มากำหนดชุดข้อมูลทดสอบ (test data) ให้สอดคล้องกัน เพื่อนำไปใช้ในการประมวลผลของโปรแกรม จากนั้นทำการทดสอบโดยการปฏิบัติการโปรแกรม แล้วนำผลลัพธ์ไปเปรียบเทียบกับผลลัพธ์ที่คาดหวังและสรุปเป็นรายงานการทดสอบ ซึ่งแสดงขั้นตอนการทดสอบ ได้ดังภาพประกอบที่ 2.4



ภาพประกอบที่ 2.4 ขั้นตอนการทดสอบซอฟต์แวร์

2.1.3 เทคนิคการทดสอบซอฟต์แวร์

ในการทดสอบซอฟต์แวร์แต่ละระดับนั้นจะมีเทคนิคที่ใช้แตกต่างกันตามความเหมาะสม (Ammann and Offutt, 2008) ปัจจุบันเทคนิคที่นิยมใช้ในการทดสอบมี 2 เทคนิคดังนี้

1) Black-box testing หรือการทดสอบแบบกล่องดำ บางครั้งเรียกว่า "การทดสอบเชิงพฤติกรรม" เนื่องจากเป็นการทดสอบผลการทำงานของซอฟต์แวร์ในแต่ละหน้าที่ตามข้อกำหนดความต้องการ (requirements specification) เพื่อดูว่าซอฟต์แวร์ทำงานได้ถูกต้องตามความต้องการหรือไม่ โดยไม่คำนึงถึงคำสั่งภายใน ซึ่งการทดสอบแบบนี้สามารถค้นพบข้อผิดพลาดในส่วนของหน้าที่การทำงานที่ผิดพลาด หน้าที่การทำงานที่ขาดหายไป ความผิดพลาดของส่วนประสานกับระบบอื่น ความผิดพลาดของการตัดสินใจทำงานต่อหรือหยุดการทำงาน ความผิดพลาดของการประมวลผลข้อมูล เป็นต้น

2) White-box testing หรือการทดสอบแบบกล่องขาว เป็นการทดสอบที่ใช้เส้นทางควบคุมการทำงานและโครงสร้างควบคุมที่ได้จากโปรแกรมมาช่วยออกแบบกรณีทดสอบเพื่อใช้ทดสอบสิ่งต่อไปนี้

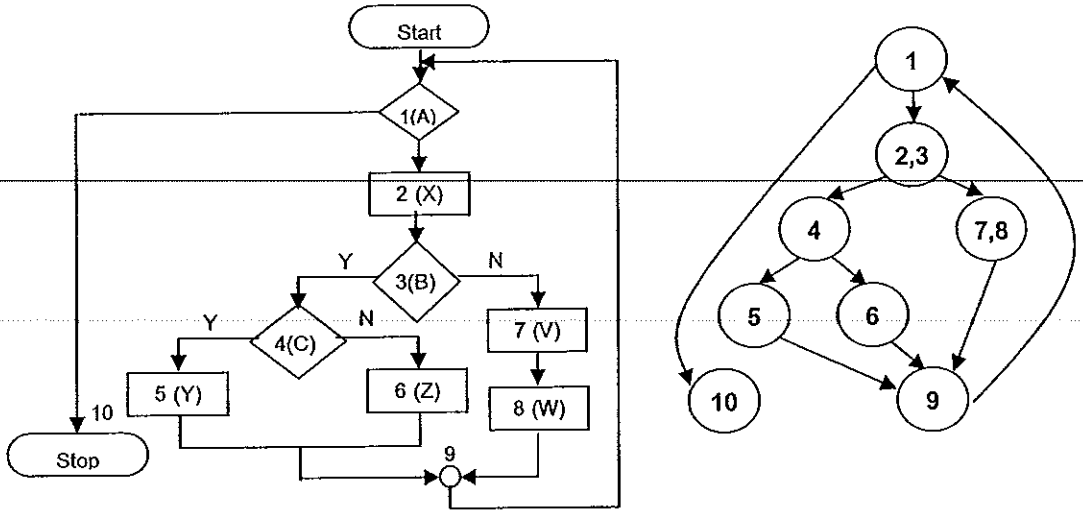
- ทดสอบว่าทุกเส้นทางในกระบวนการจะต้องสามารถทำงานได้อย่างถูกต้อง
- ทดสอบการตัดสินใจทางตรรกะทุกการตัดสินใจทั้งค่าที่เป็นจริงและค่าที่เป็นเท็จ
- ทดสอบการทำงานภายในลูปทุกลูปตามจำนวนครั้งของการวนลูป
- ทดสอบโครงสร้างข้อมูลภายในให้ถูกต้องก่อนที่จะส่งไปประมวลผลต่อยังโปรแกรมหรือหน่วยอื่น

การกำหนดชุดข้อมูลเพื่อทดสอบโปรแกรมแบบ white-box มักเริ่มจากการสร้างผังงาน (flowchart) หรือ กราฟการไหลของงาน (process flow graph) จากประโยคคำสั่งในโปรแกรม ซึ่งผังงานนั้นสามารถเป็นผังงานแสดงการทำงานของโปรแกรมหรือผังงานแสดงโครงสร้างควบคุมโปรแกรมดังภาพประกอบที่ 2.6 (ก) แสดงตัวอย่างผังงานที่สอดคล้องกับตัวอย่างโปรแกรมในภาพประกอบที่ 2.5 สำหรับกราฟการไหลของงานเป็นแผนภาพแสดงให้เห็นโครงสร้างควบคุมของโปรแกรม สัญลักษณ์ในกราฟการไหลของงานจะประกอบไปด้วยวงกลมแทนโหนด (node) และเส้นที่เชื่อมระหว่างโหนด เรียกว่า เส้นเชื่อม (edge) โดยทั่วไปจะแทนประโยคคำสั่งแต่ละประโยคในโปรแกรมด้วยโหนดแต่ละโหนดแล้วลากเส้นเชื่อมระหว่างโหนดตามทิศทางการทำงานของโปรแกรม ตัวอย่างกราฟการไหลของงานแสดงในภาพประกอบที่ 2.6 (ข)

```
Function f1()
{
    (1)     while A {
            (2) process X;
            (3) If B
                    (4) if C
                            (5) process Y;
                    (6) else process Z;
                else {
                    (7) process V;
                    (8) process W; }
            (9) } // while
} // function (10)
```

ภาพประกอบที่ 2.5 ตัวอย่างโปรแกรมที่ต้องการทดสอบ

จากผังงานหรือกราฟการไหลของงานการกำหนดชุดข้อมูลจะเริ่มต้นจากการกำหนดเส้นทางการทดสอบโปรแกรมก่อนเพื่อกำหนดเส้นทางการทดสอบจากกราฟ จึงจะสามารถกำหนดชุดข้อมูลทดสอบของแต่ละเส้นทางได้อย่างครบถ้วนเพื่อให้ทุกๆ เส้นทางต้องผ่านการทดสอบอย่างน้อย 1 ครั้ง



(ก) (ข)

ภาพประกอบที่ 2.6 ตัวอย่างผังงานและกราฟการไหลของงานของโปรแกรมในภาพประกอบที่ 2.5

จากภาพประกอบที่ 2.6 เมื่อกำหนดเส้นทางการทดสอบจากเส้นทางการทำงานของโปรแกรมทุกเส้นทางแล้วพบว่ามีทั้งหมด 4 เส้นทางที่จะต้องทดสอบดังนี้

1) 1, 10

2) 1, 2, 3, 4, 6, 9, 1, 10

3) 1, 2, 3, 4, 5, 9, 1, 10

4) 1, 2, 3, 7, 8, 9, 1, 10

กำหนดชุดข้อมูลทดสอบที่สอดคล้องกับทุกเส้นทาง ทำการทดสอบโดยใช้ชุดข้อมูลทดสอบแล้วนำผลที่ได้จากการทดสอบมาเปรียบเทียบกับผลลัพธ์ที่พึงประสงค์ หากผลลัพธ์ที่ได้จากการทดสอบไม่เป็นไปตามผลลัพธ์ที่พึงประสงค์แสดงให้เห็นว่าโปรแกรมจะต้องได้รับการแก้ไขให้ถูกต้อง การทดสอบแบบ white-box มีข้อดีคือ สามารถตรวจหาข้อผิดพลาดของโปรแกรมได้อย่างละเอียด เนื่องจากตรวจสอบจากภายในโปรแกรมจึงมั่นใจได้ว่าโปรแกรมจะมีความผิดพลาดน้อย

2.1.4 Mutation Testing

สิ่งสำคัญอีกประการหนึ่งในการทดสอบซอฟต์แวร์คือประสิทธิภาพของกรณีทดสอบที่สร้างขึ้น (Ammann and Offutt, 2008) ซึ่งประสิทธิภาพของกรณีทดสอบจะถูกประเมินโดยวิธีต่างๆ mutation testing เป็นวิธีหนึ่งที่ใช้ประเมินประสิทธิภาพของกรณีทดสอบ โดยหลักการที่สำคัญของ mutation testing คือการปรับเปลี่ยนโปรแกรมต้นฉบับที่ต้องการทดสอบให้ต่างไปจากเดิม การปรับเปลี่ยนจะกระทำกับหลายๆ จุด โดยจะปรับเปลี่ยนทีละจุดเพียงเล็กน้อยซึ่งจะเรียกโปรแกรมที่ผ่านการปรับเปลี่ยนแต่ละครั้งว่า mutant จากนั้นนำกรณีทดสอบที่สร้างขึ้นมาทดสอบกับโปรแกรมต้นฉบับและโปรแกรม mutant เพื่อดูผลลัพธ์ที่ได้ ซึ่งถ้าผลลัพธ์ที่ได้จากโปรแกรมต้นฉบับแตกต่างจากผลลัพธ์ที่ได้จากโปรแกรม mutant จะเรียก mutant นั้นว่า dead mutant และเรียก mutant ที่ให้ผลลัพธ์เหมือนกับโปรแกรมต้นฉบับว่า live mutant สำหรับ live mutant ที่ไม่ว่าใช้กรณีทดสอบใดๆ ก็ให้ผลลัพธ์ที่ไม่แตกต่างกับโปรแกรมต้นฉบับจะถูกเรียกว่า equivalent mutant จำนวนร้อยละของ dead mutant แสดงถึงประสิทธิภาพของกรณีทดสอบที่เรียกว่า mutation score โดยหาได้จากสูตรดังต่อไปนี้

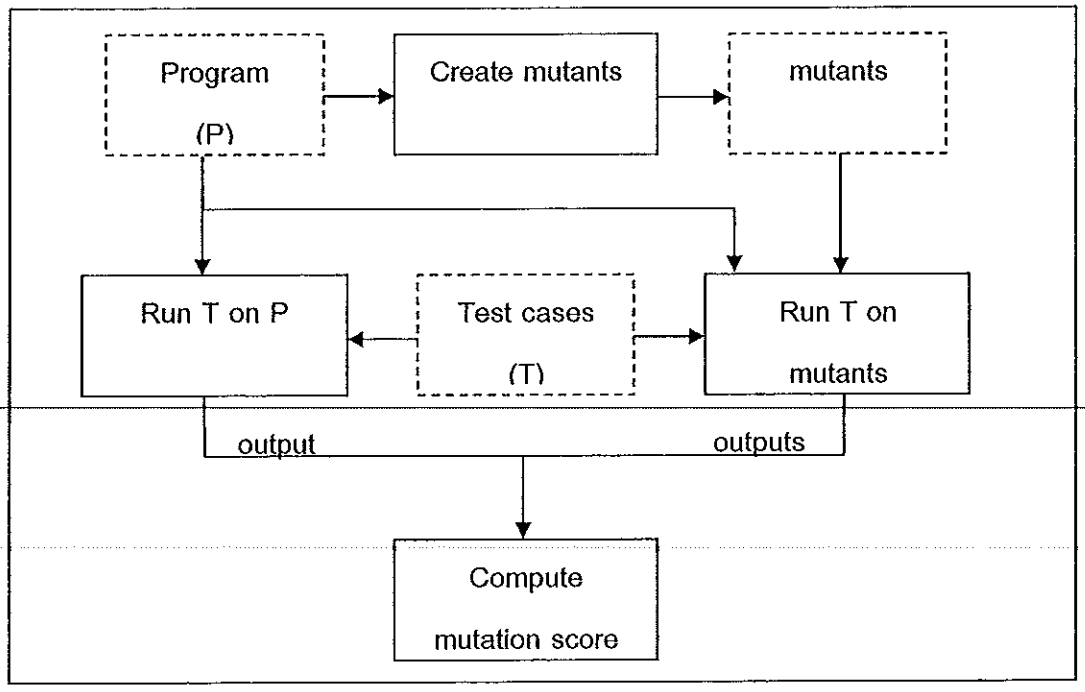
$$\text{Mutation score} = \frac{\#\text{dead mutants}}{\#\text{total mutants} - \#\text{equivalent mutants}}$$

เมื่อ #dead mutants หมายถึงจำนวนของ dead mutant

#total mutants หมายถึงจำนวนของ mutant ทั้งหมด

#equivalent mutants หมายถึงจำนวนของ equivalent mutant

หาก mutant ทั้งหมดให้ผลลัพธ์ที่แตกต่างจากโปรแกรมต้นฉบับและไม่มี equivalent mutant จะทำให้ mutation score มีค่าเท่ากับ 1 ซึ่งเป็นค่าสูงสุดในการแสดงถึงประสิทธิภาพของกรณีทดสอบที่นำมาพิจารณา ในภาพประกอบที่ 2.7 ขั้นตอนการทำ mutation testing เริ่มต้นจากการสร้างโปรแกรม mutant จากโปรแกรมต้นฉบับ จากนั้นนำกรณีทดสอบที่สร้างได้มาปฏิบัติการผ่านโปรแกรมต้นฉบับและโปรแกรม mutant ทีละโปรแกรม หากผลลัพธ์ที่ได้จากโปรแกรม mutant ตัวใดแตกต่างจากผลลัพธ์ที่ได้จากโปรแกรมต้นฉบับให้จัดโปรแกรม mutant ตัวนั้น เป็น dead mutant เมื่อทำครบทุกโปรแกรม mutant แล้ว หากพบว่ามีโปรแกรม mutant ตัวใดให้ผลลัพธ์ที่ไม่แตกต่างกับผลลัพธ์จากโปรแกรมต้นฉบับให้จัดโปรแกรม mutant นั้นเป็น equivalent mutant โดยที่สัญลักษณ์สี่เหลี่ยมผืนผ้าแทนกระบวนการทำงานต่างๆ และสัญลักษณ์สี่เหลี่ยมผืนผ้าที่มีเส้นประ แทน input หรือ output ในกระบวนการ mutation testing



ภาพประกอบที่ 2.7 กระบวนการ mutation testing

Mutation operators (Yu – Seung Ma, *et al.*, 2005) เป็นสิ่งสำคัญที่ช่วยในการสร้างโปรแกรม mutant ซึ่ง mutation operator จะถูกใช้เพื่อเป็นแนวทางในการปรับเปลี่ยนโปรแกรมเพื่อสร้าง mutant เช่น operator การแทนที่ตัวดำเนินการสัมพันธ์ (relational operator replacement) จากตัวอย่างบางส่วนของโปรแกรมต่อไปนี้

1. if (x > 20)
2. call A ();
3. if (x > 40)
4. call B ();

สามารถสร้างโปรแกรม mutant โดยใช้ operator การแทนที่ตัวดำเนินการสัมพันธ์ในบรรทัดที่ 1 ได้ดังนี้

1. if (x < 20)
2. call A ();
3. if (x > 40)
4. call B ();

การแทนที่จะถูกกระทำเพียงจุดเดียวสำหรับการสร้างโปรแกรม mutant แต่ละโปรแกรม

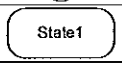
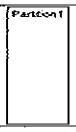
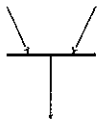
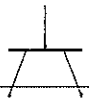
2.2 UML Activity Diagram

Unified Modeling Language (UML) เป็นภาษาเชิงโมเดล (OMG, 2003) ที่ใช้ในการสร้างแบบจำลองระบบ ซึ่งถูกกำหนดโดย Object Management Group (OMG) โดยใช้หลักการพัฒนาโปรแกรมเชิงวัตถุ (Object Oriented Programming: OOP) รูปแบบของภาษาอยู่ในรูปแบบของสัญลักษณ์ ซึ่งใช้สำหรับสื่อความหมาย และมีกฎระเบียบที่มีความหมายต่อการเขียนโปรแกรม โดยทั่วไปแผนภาพที่มีอยู่ใน UML แบ่งได้เป็น 2 ประเภท คือ

- แผนภาพที่แสดงโครงสร้างของโปรแกรม ได้แก่ class diagram, object diagram, component diagram และ deployment diagram
- แผนภาพที่แสดงพฤติกรรมของระบบ ได้แก่ use case diagram, sequence diagram, activity diagram, collaboration diagram และ statechart diagram

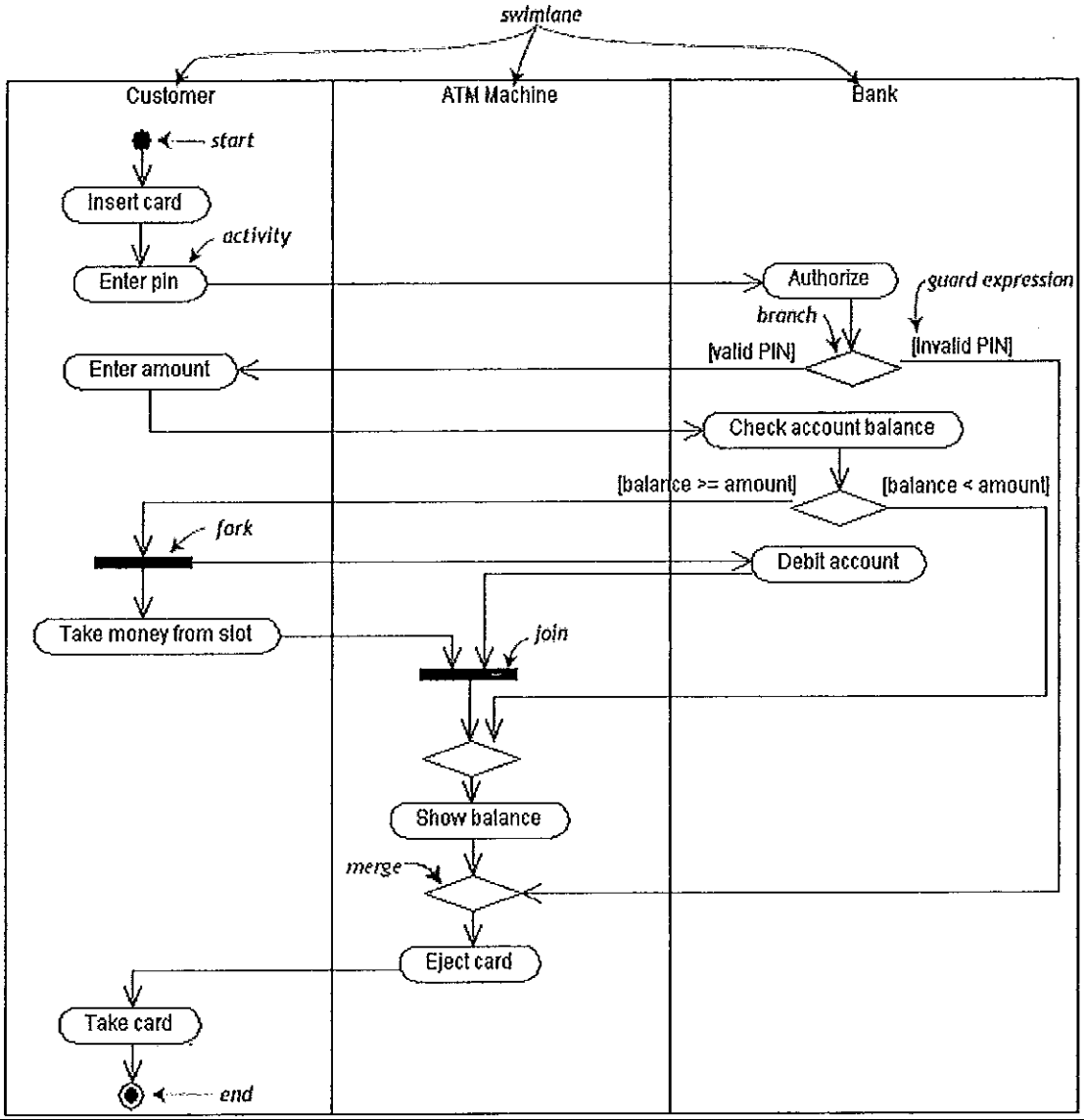
แผนภาพกิจกรรม (activity diagram) เป็นแผนภาพที่แสดงพฤติกรรมการทำงานของระบบซอฟต์แวร์ โดยอธิบายลำดับและขั้นตอนการทำงานที่เกิดขึ้นในซอฟต์แวร์ มีลักษณะคล้ายกับผังงาน (flowchart) สัญลักษณ์ที่ใช้ในแผนภาพกิจกรรมแสดงดังตารางที่ 2.1

ตารางที่ 2.1 สัญลักษณ์ที่ใช้ในแผนภาพกิจกรรม

สัญลักษณ์	ชื่อสัญลักษณ์	คำอธิบาย
	Initial State	แสดงสถานะจุดเริ่มต้น
	Final State	แสดงสถานะจุดสิ้นสุด
	State	แสดงสถานะการทำงานของกิจกรรมนั้น
	Decision, Merge	แสดงตัวเลือกในการตัดสินใจ หรือแสดงการรวมกัน (merge) ของการทำงานในโปรแกรม ซึ่งถ้าแสดงการตัดสินใจจะปรากฏเงื่อนไขที่ระบุอยู่ภายในสัญลักษณ์วงเล็บก้ามปู [ ] ส่วนการแสดงการรวมการทำงานจะไม่มีการระบุเงื่อนไข
	Control Flow	แสดงการเปลี่ยนสถานะการทำงานจากกิจกรรมหนึ่งไปยังอีกกิจกรรมหนึ่ง
	Swim lane	ใช้เพื่อแบ่งแยกการทำงานว่าใครมีหน้าที่ในส่วนใดบ้าง
	Transition (Join)	แสดงการทำงานของกิจกรรมที่รวมการทำงานจากหลายกิจกรรมที่ทำงานขนานกัน มารวมกันเป็นกิจกรรมเดียว
	Transition (Fork)	แสดงการทำงานของกิจกรรมที่แยกการทำงานจากกิจกรรมเดียวออกเป็นหลายกิจกรรมที่ทำงานขนานกันไป

ตัวอย่างแผนภาพกิจกรรมการถอนเงินจากตู้ ATM (Randy Miller, 2003: Online) ซึ่งอธิบายขั้นตอนการทำงานในการถอนเงินจากตู้ ATM ซึ่งการทำงานจะแบ่งออกเป็น 3 ส่วน ได้แก่ ส่วนของลูกค้า ส่วนของตู้ ATM และส่วนของธนาคาร โดยเริ่มต้นให้ลูกค้าใส่บัตร ATM แล้วป้อนรหัส จากนั้นในส่วนของธนาคารจะทำการตรวจสอบความถูกต้องของรหัสหากรหัสถูกต้องจะให้ลูกค้าป้อนจำนวนที่ต้องการถอนซึ่งธนาคารจะตรวจสอบยอดในบัญชีของลูกค้าก่อนหากจำนวนเงินในบัญชีมีพอสำหรับการถอนก็จะอนุมัติการถอนเงินพร้อมทั้งหักยอดบัญชีคงเหลือและในขณะเดียวกัน ในส่วนของตู้ ATM จะทำการออกเงินให้แก่ลูกค้าแล้วแสดงยอดเงิน

คงเหลือที่หน้าจอ จากนั้นลูกค้าจะรับเงินจากตู้และสุดท้ายตู้ ATM จะทำการคืนบัตรให้แก่ลูกค้า ซึ่งมีรายละเอียดดังภาพประกอบที่ 2.8



ภาพประกอบที่ 2.8 แผนภาพกิจกรรมการถอนเงินจากตู้ ATM

(Randy Miller, 2003: Online)

2.3 Classification Tree Method (CTM)

Classification Tree Method (CTM) เป็นวิธีหนึ่งที่ใช้ในการทดสอบซอฟต์แวร์แบบ black-box (Grochtmann and Grimm, 1993) หลักการของ Classification Tree Method คือการแบ่ง input domain ออกเป็นหมวดหมู่แล้วแยกย่อยแต่ละหมู่ของ input domain ออกเป็น class ภายใต้เงื่อนไขเดียวกัน การจัดแบ่งโดเมนและ class แสดงอยู่ในรูปของต้นไม้ ซึ่งเรียกว่าต้นไม้การจำแนก (classification tree) การสร้างกรณีทดสอบโดยใช้ classification tree method จะนำต้นไม้การจำแนกไปสร้างตารางกรณีทดสอบ (combination table หรือ test case table) จากนั้นจะสร้างกรณีทดสอบโดยการรวมโหนดของต้นไม้แต่ละกิ่ง ซึ่งสามารถดูได้ง่ายและชัดเจน ขั้นตอนของ classification tree method มีลำดับการทำงานดังต่อไปนี้

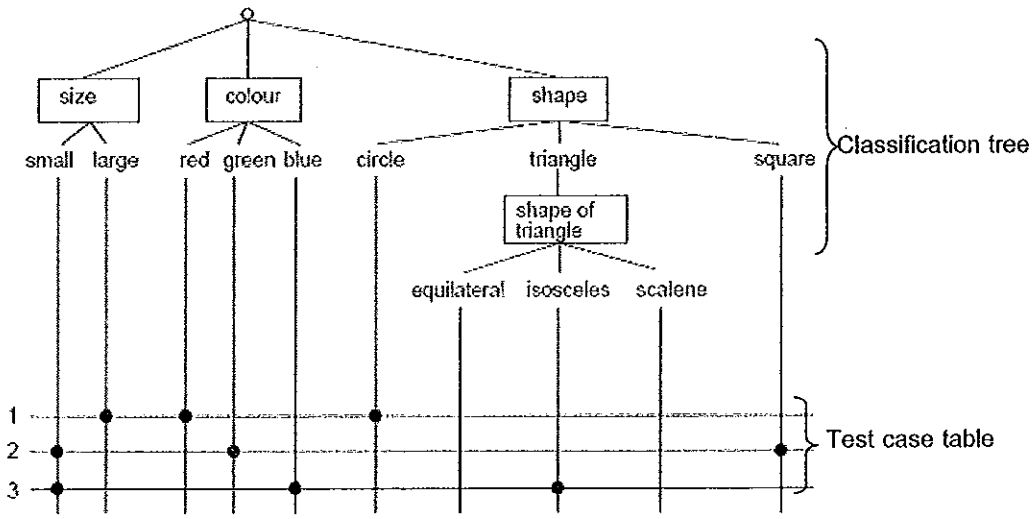
ขั้นที่ 1 วิ เคราะห์ requirements specification ของซอฟต์แวร์และแบ่ง input domain ออกเป็นหมวดหมู่พร้อมทั้งระบุ class ที่เป็นไปได้ในแต่ละหมวดหมู่

ขั้นที่ 2 สร้างต้นไม้การจำแนก โดยกำหนดให้แต่ละหมวดหมู่เป็นโหนดลูกแยกออกมาจากโหนดเริ่มต้นและกำหนดให้ class ของหมวดหมู่เป็นโหนดใบของโหนดของหมวดหมู่นั้น

ขั้นที่ 3 สร้างตารางกรณีทดสอบจากต้นไม้การจำแนกที่ได้จากขั้นตอนที่ 2 โดยลากเส้นตารางใต้ต้นไม้การจำแนก

ขั้นที่ 4 เลือกกรณีทดสอบโดยการรวมโหนดใบจากต้นไม้การจำแนกทุกหมวดหมู่ หมวดหมู่ละ 1 โหนด โดยการรวมกันของโหนดจะกระทำในทุกกรณีที่เป็นไปได้ เช่น ภายในต้นไม้การจำแนกประกอบด้วยหมวดหมู่ 2 หมวดหมู่ โดยหมวดหมู่ที่ 1 ประกอบด้วย 3 class และหมวดหมู่ที่ 2 ประกอบด้วย 3 class ดังนั้นจะได้กรณีทดสอบทั้งหมด 9 กรณีทดสอบ

ตัวอย่างการใช้วิธี CTM สร้างกรณีทดสอบจาก requirements specification ของระบบ computer vision ซึ่งกำหนด input domain เป็น 3 กลุ่มได้แก่ size, color และ shape จากนั้นระบุ input domain ย่อยที่เป็นไปได้ในแต่ละกลุ่มแล้วสร้างต้นไม้การจำแนกและตารางกรณีทดสอบ ดังแสดงรายละเอียดดังภาพประกอบที่ 2.9



ภาพประกอบที่ 2.9 การใช้วิธี CTM สร้างกรณีทดสอบจาก requirements specification ของระบบ computer vision

บทที่ 3

การสร้างกรณีทดสอบจากแผนภาพกิจกรรมโดยใช้หลักการ CCTM

เนื้อหาในบทนี้กล่าวถึงการสร้างกรณีทดสอบจากแผนภาพกิจกรรมโดยใช้หลักการ Condition - Classification Tree Model (CCTM) ซึ่งส่วนแรกจะกล่าวถึงกรอบแนวคิดของหลักการที่นำเสนอและขั้นตอนต่างๆ ในการสร้างกรณีทดสอบ จากนั้นจะนำเสนอตัวอย่างการสร้างกรณีทดสอบตามหลักการที่นำเสนอ

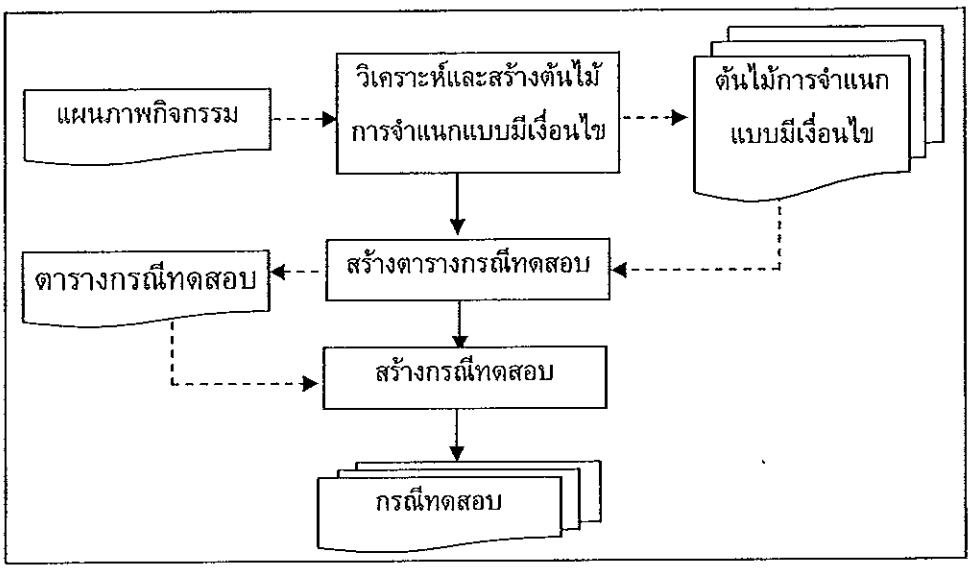
3.1 กระบวนการสร้างกรณีทดสอบจากแผนภาพกิจกรรมโดยใช้หลักการ CCTM

งานวิจัยนี้เสนอแบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไข ซึ่งประยุกต์มาจากวิธีการ Classification Tree Method ที่สร้างกรณีทดสอบจาก requirements specification ของซอฟต์แวร์ ในงานวิจัยนี้ปรับใช้กับ specification ที่เขียนในรูปของแผนภาพกิจกรรมพร้อมทั้งเพิ่มการกำหนดเงื่อนไขเพื่อให้การสร้างกรณีทดสอบมีความสะดวกและรวดเร็วขึ้น โดยมีรายละเอียดขั้นตอนดังต่อไปนี้

3.1.1 กรอบแนวคิดในการสร้างกรณีทดสอบจากแผนภาพกิจกรรมโดยใช้หลักการ CCTM

กรอบแนวคิดที่นำเสนอคือการสร้างกรณีทดสอบจาก specification ของซอฟต์แวร์ที่อยู่ในรูปแผนภาพกิจกรรม ซึ่งส่วนสำคัญส่วนหนึ่งของแผนภาพกิจกรรมที่มีผลในการกำหนดกิจกรรมถัดไป และเป็นตัวกำหนดเส้นทางการทำกิจกรรมต่างๆ คือ ส่วนการตัดสินใจ (decision point) เส้นทางการตัดสินใจที่ออกจากส่วนการตัดสินใจจะขึ้นอยู่กับข้อมูลนำเข้า (input) ที่ได้รับจากกิจกรรมก่อนหน้า โดยข้อมูลนำเข้าจะถูกกำหนดเป็นเงื่อนไขสำหรับทางเลือกแต่ละเส้นทางที่แยกออกจากส่วนการตัดสินใจ ดังนั้นจากเงื่อนไขการตัดสินใจของเส้นทางต่างๆสามารถใช้เพื่อระบุขอบเขตของข้อมูลนำเข้า (input domain) ที่สำคัญต่อการทำงานของซอฟต์แวร์ได้ ขั้นตอนแรกของวิธีการที่นำเสนอนี้เริ่มต้นจากการวิเคราะห์ input domain จากตำแหน่งการตัดสินใจของแผนภาพกิจกรรม แต่ละตำแหน่งการตัดสินใจจะพิจารณาเป็นหมวดหมู่หนึ่งของ input domain จากนั้นนำ input domain แต่ละหมวดหมู่มาสร้างเป็น

ต้นไม้การจำแนกพร้อมทั้งระบุเงื่อนไขของ input domain ของแต่ละหมวดหมู่ ขั้นตอนถัดไปจะนำต้นไม้การจำแนกที่ระบุเงื่อนไขทั้งหมดที่สร้างขึ้นไปใช้เพื่อสร้างตารางกรณีทดสอบ สุดท้ายกรณีทดสอบจะถูกกำหนดขึ้นจากตารางกรณีทดสอบโดยพิจารณาจากเงื่อนไขที่ระบุไว้บนต้นไม้การจำแนก รายละเอียดขั้นตอนแสดงดังภาพประกอบที่ 3.1



ภาพประกอบที่ 3.1 กรอบแนวคิดในการสร้างกรณีทดสอบจากแผนภาพกิจกรรม

3.1.2 กระบวนการสร้างกรณีทดสอบจากแผนภาพกิจกรรมโดยใช้หลักการ CCTM

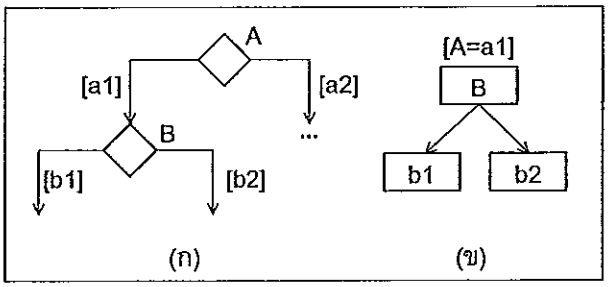
การสร้างกรณีทดสอบด้วยหลักการ CCTM มีขั้นตอนและรายละเอียดดังต่อไปนี้

ขั้นตอนที่ 1 วิเคราะห์และสร้างต้นไม้การจำแนกแบบมีเงื่อนไข

1) วิเคราะห์แผนภาพกิจกรรมโดยพิจารณาส่วนการตัดสินใจ (decision point) และเงื่อนไขต่างๆ ของการตัดสินใจที่ระบุในสัญลักษณ์ [ ] โดยที่แต่ละส่วนการตัดสินใจระบุถึงลักษณะของ input domain ที่เป็นไปได้ 1 กลุ่ม

2) นำแต่ละการตัดสินใจพร้อมเงื่อนไขที่ระบุในสัญลักษณ์ [ ] มาสร้างต้นไม้การจำแนก 1 ต้น ซึ่งมีขั้นตอนการสร้างต้นไม้การจำแนกดังต่อไปนี้

2.1) สร้างโหนดเริ่มต้นสำหรับต้นไม้การจำแนก เรียกว่า category จากตัวอย่างในภาพประกอบที่ 3.2 (ก) ส่วนการตัดสินใจ B จะถูกนำมาสร้างเป็นโหนด B ดังแสดงในภาพประกอบที่ 3.2 (ข)



ภาพประกอบที่ 3.2 ตัวอย่างการสร้างต้นไม้การจำแนกแบบมีเงื่อนไขจากแผนภาพกิจกรรม

2.2) สร้างโหนดลูกสำหรับ category ย่อยในกลุ่มนี้ เรียกว่า option โดย option ต่างๆ พิจารณาจากเส้นทางที่ออกจากส่วนการตัดสินใจที่กำลังพิจารณา ดังตัวอย่างในภาพประกอบที่ 3.2 (ข) โหนดลูกของโหนด B ได้แก่ โหนด b1 และโหนด b2

2.3) ระบุเงื่อนไขให้กับต้นไม้การจำแนก โดยพิจารณาจากเส้นทางก่อนเข้าถึงส่วนการตัดสินใจของต้นไม้การจำแนกนี้ ว่าได้ผ่านเงื่อนไขส่วนการตัดสินใจส่วนใดมาบ้าง ให้นำเงื่อนไขการตัดสินใจเหล่านั้นมาระบุไว้บนต้นไม้การจำแนกนี้ภายในสัญลักษณ์ [ ] และนำไปใส่ไว้บนต้นไม้การจำแนก ดังตัวอย่างในภาพประกอบที่ 3.2 ส่วนการตัดสินใจ B ต้องผ่านเส้นทางที่ออกจากส่วนการตัดสินใจ A ทางเส้น a1 ดังนั้นจึงระบุเงื่อนไข [A=a1] ไว้บนต้นไม้การจำแนก B

ขั้นตอนที่ 2 สร้างตารางกรณีทดสอบ

1) นำต้นไม้การจำแนกที่ระบุเงื่อนไขทั้งหมดมาวางเรียงกันโดยเรียงลำดับตามจำนวนเงื่อนไขที่ระบุไว้บนต้นไม้การจำแนกจากจำนวนน้อยไปหาจำนวนมาก กรณีที่จำนวนเงื่อนไขของต้นไม้การจำแนกต้นใดมีค่าเท่ากันสามารถวางต้นใดก่อนหรือหลังก็ได้

2) สร้างตาราง โดยลากเส้นคอลัมน์จากโหนดลูกของต้นไม้การจำแนกที่ระบุเงื่อนไขแต่ละต้นและลากเส้นแถวของตารางเพื่อใช้กำหนดกรณีทดสอบในขั้นตอนที่ 3

ขั้นตอนที่ 3 สร้างกรณีทดสอบ

1) กำหนดกรณีทดสอบแต่ละกรณีโดยทำสัญลักษณ์ลงบนเส้นแถวของตาราง โดยเริ่มพิจารณาจากต้นไม้การจำแนกที่ระบุเงื่อนไขจากทางด้านซ้าย ซึ่งเงื่อนไขจะถูกใช้เพื่อระบุว่าแต่ละโหนดในต้นนั้นสามารถเลือกจับกับโหนดลูกในต้นไม้การจำแนกต้นใดได้บ้าง แล้วทำสัญลักษณ์ลงบนเส้นแถวในตาราง กรณีที่เงื่อนไขของต้นไม้การจำแนกมีความซ้ำซ้อนกับเงื่อนไขของต้นที่เคยพิจารณาแล้ว ให้ทำสัญลักษณ์ต่อในต้นนั้นได้เลยเพื่อลดความซ้ำซ้อน

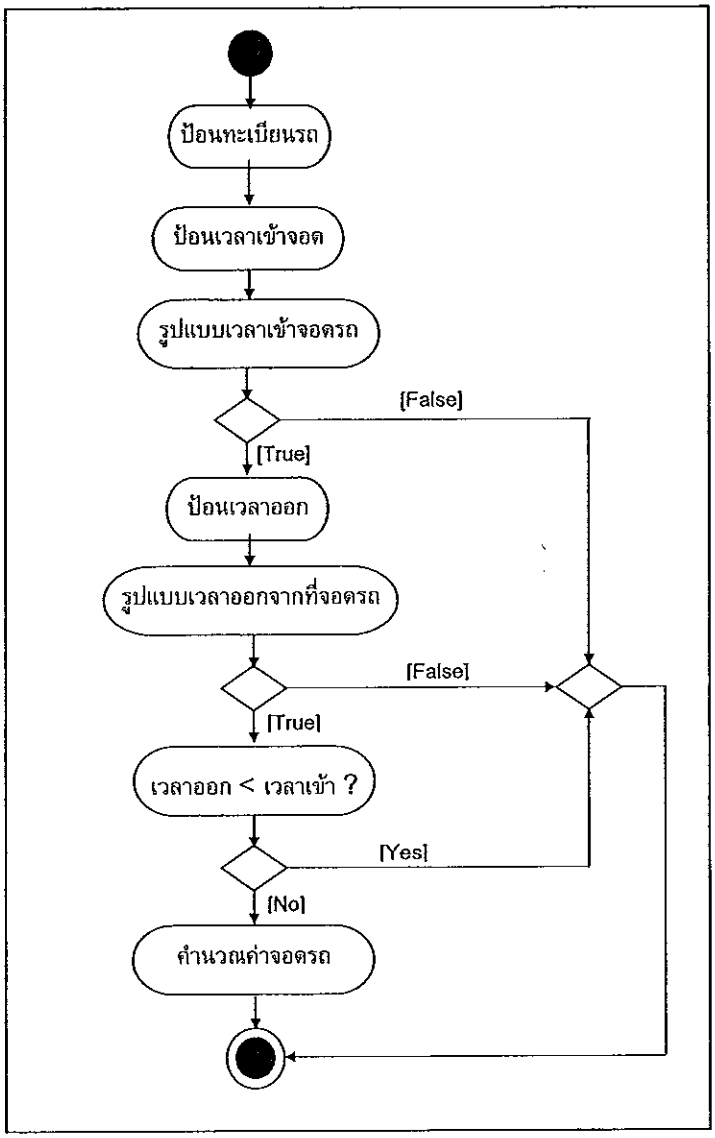
2) เมื่อพิจารณาต้นไม้การจำแนกครบทุกต้นแล้วหากมีโหนดลูกในต้นไม้ต้นใดที่ไม่ถูกเลือกเลย ให้เลือกโหนดลูกนั้นเพียงโหนดเดียวเป็น 1 กรณี

3.2 ตัวอย่างการสร้างกรณีทดสอบจากแผนภาพกิจกรรมโดยใช้หลักการ CCTM

เพื่อแสดงตัวอย่างการสร้างกรณีทดสอบโดยวิธีที่ได้นำเสนอ ในงานวิจัยนี้ได้เลือกใช้แผนภาพกิจกรรมของโปรแกรมคำนวณค่าจอดรถ และแผนภาพกิจกรรมของโปรแกรมคำนวณจำนวนเงินที่ต้องผ่อนชำระต่องวดซึ่งแสดงรายละเอียดดังต่อไปนี้

3.2.1 กรณีตัวอย่างที่ 1 การสร้างกรณีทดสอบจากแผนภาพกิจกรรมของโปรแกรมคำนวณค่าจอดรถ

รายละเอียดแผนภาพกิจกรรมของโปรแกรมคำนวณค่าจอดรถเริ่มต้นจากการให้ผู้ใช้ป้อนทะเบียนรถที่ต้องการคำนวณค่าจอดรถ จากนั้นให้ผู้ใช้ป้อนเวลาที่เข้าจอดรถซึ่งต้องป้อนให้ถูกต้องตามรูปแบบที่กำหนดไว้หากมีการป้อนที่ผิดรูปแบบจะทำการยกเลิกการคำนวณทันที แต่หากรูปแบบการป้อนถูกต้องจะถามเวลาที่ออกจากที่จอดรถซึ่งจะต้องป้อนเวลาออกให้ถูกต้องตามรูปแบบเช่นกัน หากป้อนเวลาออกผิดรูปแบบจะยกเลิกการทำงานแต่หากการป้อนเวลาออกถูกต้องจะมีการตรวจสอบว่าเวลาที่ออกจากที่จอดรถน้อยกว่าเวลาที่เข้าจอดรถหรือไม่ หากน้อยกว่าจะจบการทำงานและหากมากกว่าจะทำการคำนวณค่าจอดรถให้แก่ผู้ใช้ ซึ่งมีรายละเอียดดังภาพประกอบที่ 3.3



ภาพประกอบที่ 3.3 แผนภาพกิจกรรมของโปรแกรมคำนวณค่าจอดรถ

(พนิดา พานิชกุล, 2548)

จากแผนภาพกิจกรรมของโปรแกรมคำนวณค่าจอดรถสามารถสร้างกรณีทดสอบโดยหลักการ CCTM ได้ดังต่อไปนี้

ขั้นตอนที่ 1 วิเคราะห์และสร้างต้นไม้การจำแนกแบบมีเงื่อนไข

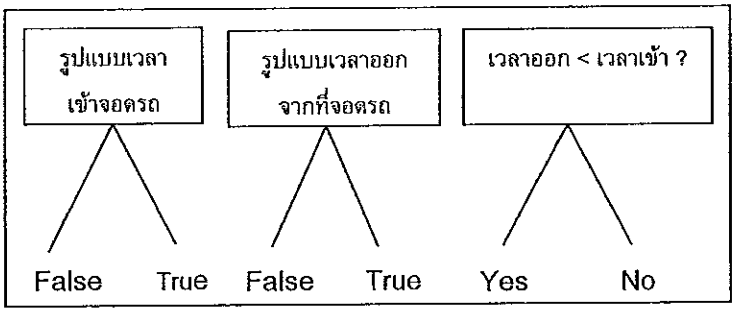
1) วิเคราะห์แผนภาพกิจกรรมพบว่ามีส่วนการตัดสินใจ (decision point) ที่เป็นทางเลือกปรากฏอยู่ทั้งหมด 3 ตำแหน่ง ซึ่งส่วนการตัดสินใจแต่ละส่วนคือลักษณะของ input

domain ที่เป็นไปได้ 1 กลุ่ม (category) ซึ่งในที่นี้จะประกอบด้วย 3 categories และแต่ละ category สามารถแยกออกเป็นกลุ่มย่อยเรียกว่า option ได้ 2 options รายละเอียดแสดงได้ดังตารางที่ 3.1

ตารางที่ 3.1 category และ option จากการวิเคราะห์แผนภาพกิจกรรมของโปรแกรมคำนวณค่าจอดรถ

Category	Option	
1. รูปแบบเวลาเข้าจอดรถ	True	False
2. รูปแบบเวลาออกจากที่จอดรถ	True	False
3. เวลาออก < เวลาเข้า ?	Yes	No

2) จาก category และ option สามารถสร้างต้นไม้การจำแนกได้ทั้งหมด 3 ต้น ซึ่งแสดงได้ดังภาพประกอบที่ 3.4



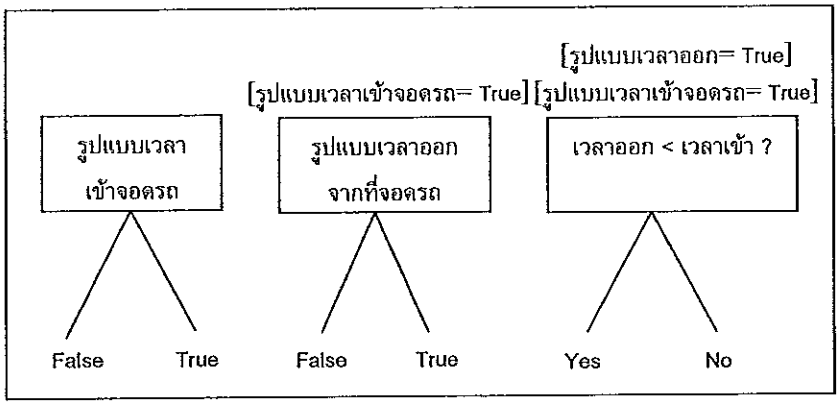
ภาพประกอบที่ 3.4 ต้นไม้การจำแนกจากการวิเคราะห์แผนภาพกิจกรรมของโปรแกรมคำนวณค่าจอดรถ

ระบุเงื่อนไขให้กับต้นไม้การจำแนกแต่ละต้น โดยพิจารณาจากเส้นทางก่อนเข้าถึงส่วนการตัดสินใจของต้นไม้การจำแนกแต่ละต้นที่กำลังพิจารณาว่าได้ผ่านเงื่อนไขส่วนการตัดสินใจส่วนใดมาบ้างให้นำเงื่อนไขการตัดสินใจเหล่านั้นมาระบุไว้บนต้นไม้การจำแนกภายในสัญลักษณ์ [ ]

เริ่มต้นพิจารณาจากต้นไม้การจำแนก *รูปแบบเวลาเข้าจอดรถ* พบว่าส่วนการตัดสินใจ *รูปแบบเวลาเข้าจอดรถ* ไม่ได้ผ่านส่วนการตัดสินใจอื่นมาเลยจึงไม่ต้องระบุเงื่อนไขให้กับต้นไม้การจำแนก *รูปแบบเวลาเข้าจอดรถ*

พิจารณาต้นไม้การจำแนก *รูปแบบเวลาออกจากที่จอดรถ* พบว่าส่วนการตัดสินใจ *รูปแบบเวลาออกจากที่จอดรถ* ต้องผ่านส่วนการตัดสินใจ *รูปแบบเวลาเข้าจอดรถ* โดยมีเงื่อนไขเป็น True ดังนั้นระบุเงื่อนไข [รูปแบบเวลาเข้าจอดรถ=True] ให้กับต้นไม้การจำแนก *รูปแบบเวลาออกจากที่จอดรถ*

พิจารณาต้นไม้การจำแนก *เวลาออก < เวลาเข้า ?* พบว่าส่วนการตัดสินใจ *เวลาออก < เวลาเข้า ?* ต้องผ่านส่วนการตัดสินใจ *รูปแบบเวลาเข้าจอดรถ* โดยมีเงื่อนไขเป็น True และต้องผ่านส่วนการตัดสินใจ *รูปแบบเวลาออกจากที่จอดรถ* โดยมีเงื่อนไขเป็น True ดังนั้นระบุเงื่อนไข [รูปแบบเวลาเข้าจอดรถ=True] และ [รูปแบบเวลาออกจากที่จอดรถ=True] ให้กับต้นไม้การจำแนก *เวลาออก < เวลาเข้า ?* ต้นไม้การจำแนกทั้งหมดที่มีการระบุเงื่อนไขสามารถแสดงได้ดังภาพประกอบที่ 3.5



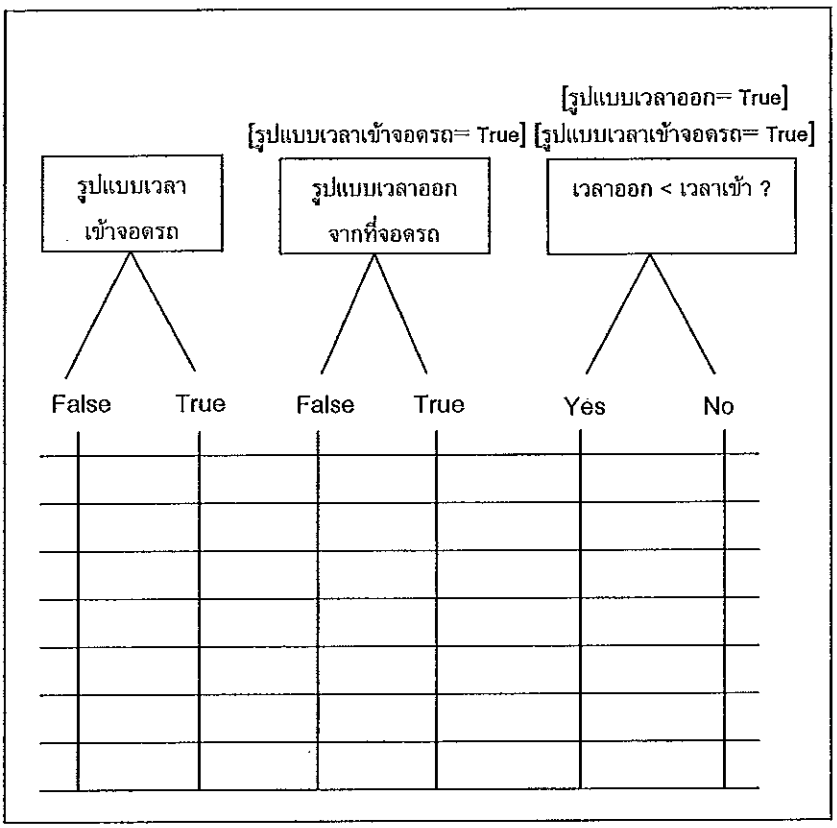
ภาพประกอบที่ 3.5 ต้นไม้การจำแนกที่ระบุเงื่อนไข จากการวิเคราะห์แผนภาพกิจกรรมของโปรแกรมคำนวณค่าจอดรถ

ขั้นตอนที่ 2 สร้างตารางกรณีทดสอบ

1) นำต้นไม้การจำแนกที่ระบุเงื่อนไขทั้งหมดมาวางเรียงกันโดยเรียงตามลำดับจำนวนเงื่อนไขที่ระบุจากน้อยไปหามาก ดังแสดงในภาพประกอบที่ 3.6 ซึ่งต้นไม้การจำแนก *เวลาออก < เวลาเข้า ?* มีเงื่อนไขจำนวน 2 เงื่อนไข และต้นไม้การจำแนก *รูปแบบเวลาออกจากที่จอดรถ* มีเงื่อนไขจำนวน 1 เงื่อนไข ดังนั้นต้นไม้การจำแนก *เวลาออก < เวลาเข้า ?* จึงมีลำดับต่อจากต้นไม้การจำแนก *รูปแบบเวลาออกจากที่จอดรถ*

2) สร้างตาราง โดยลากเส้นคอลัมน์จากโหนดลูกของต้นไม้การจำแนกที่ระบุเงื่อนไขแต่ละต้นและลากเส้นแถวของตารางเพื่อใช้กำหนดกรณีทดสอบซึ่งจากตัวอย่างข้างต้น

สามารถแสดงการเรียงต้นไม้การจำแนกที่ระบุเงื่อนไขและตารางกรณีทดสอบได้ดังภาพประกอบที่ 3.6



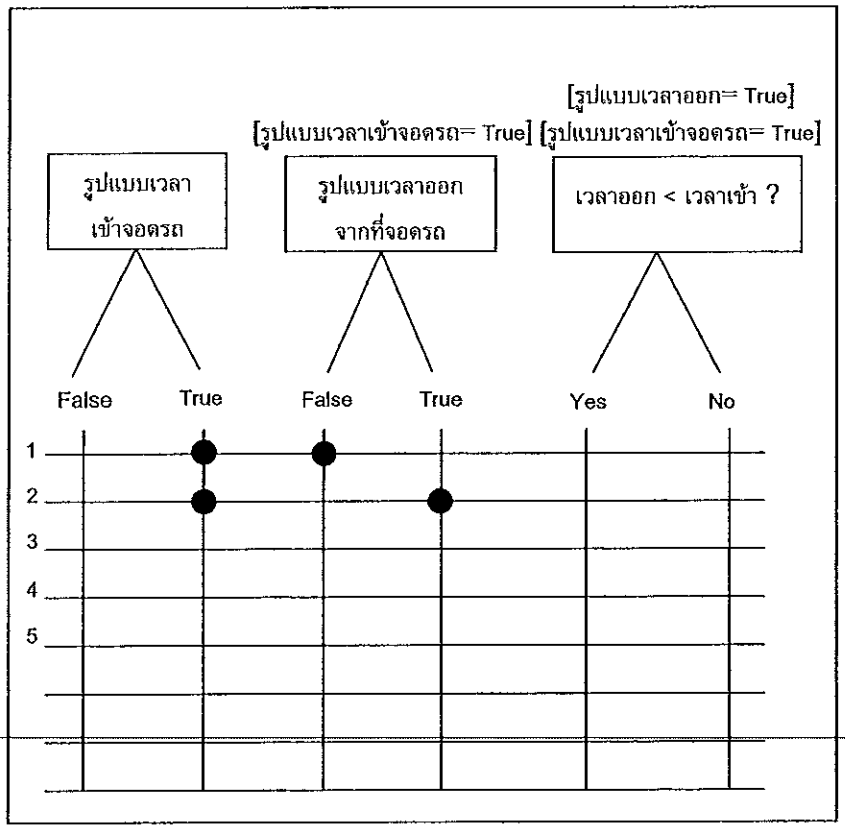
ภาพประกอบที่ 3.6 ต้นไม้การจำแนกที่ระบุเงื่อนไขและตารางกรณีทดสอบจากการวิเคราะห์แผนภาพกิจกรรมของโปรแกรมคำนวณค่าจอดรถ

ขั้นตอนที่ 3 สร้างกรณีทดสอบ

การกำหนดกรณีทดสอบในที่นี้จะเริ่มพิจารณาจากต้นไม้การจำแนกต้นที่ 1 พบว่าไม่มีเงื่อนไขที่ระบุไว้จึงพิจารณาต้นไม้การจำแนกต้นที่ 2 ต่อ พบว่ามีเงื่อนไข [รูปแบบเวลาเข้าจอดรถ = True] ดังนั้นการระบุกรณีทดสอบจึงทำสัญลักษณ์บนตารางกรณีทดสอบได้ดังต่อไปนี้

1) เส้นที่ 1 ทำสัญลักษณ์บนเส้นที่ลากจากโหนด True ของต้นไม้การจำแนก *รูปแบบเวลาเข้าจอดรถ* และทำสัญลักษณ์บนเส้นที่ลากจากโหนด False ของต้นไม้การจำแนก *รูปแบบเวลาออกจากที่จอดรถ*

2) เส้นที่ 2 ทำสัญลักษณ์บนเส้นที่ลากจากโหนด True ของต้นไม้การจำแนก ***รูปแบบเวลาเข้าจอดรถ*** และทำสัญลักษณ์บนเส้นที่ลากจากโหนด True ของต้นไม้การจำแนก ***รูปแบบเวลาออกจากที่จอดรถ*** ภาพประกอบที่ 3.7 แสดงการระบุกรณีทดสอบจากการพิจารณาเงื่อนไขของต้นไม้การจำแนก ***รูปแบบเวลาออกจากที่จอดรถ*** จะพบว่าต้นไม้การจำแนกต้นที่ 1 เป็นต้นไม้การจำแนกที่ไม่มีการระบุเงื่อนไขไม่สามารถเลือกโหนดลูกจากต้นอื่นได้แต่สามารถถูกเลือกจากต้นไม้การจำแนกต้นอื่นได้ตามเงื่อนไข ดังขั้นตอนข้างต้นซึ่งโหนด True ของต้นไม้การจำแนก ***รูปแบบเวลาเข้าจอดรถ*** ถูกเลือกจากต้นไม้การจำแนก ***รูปแบบเวลาออกจากที่จอดรถ*** ตามเงื่อนไข [รูปแบบเวลาเข้าจอดรถ = True]

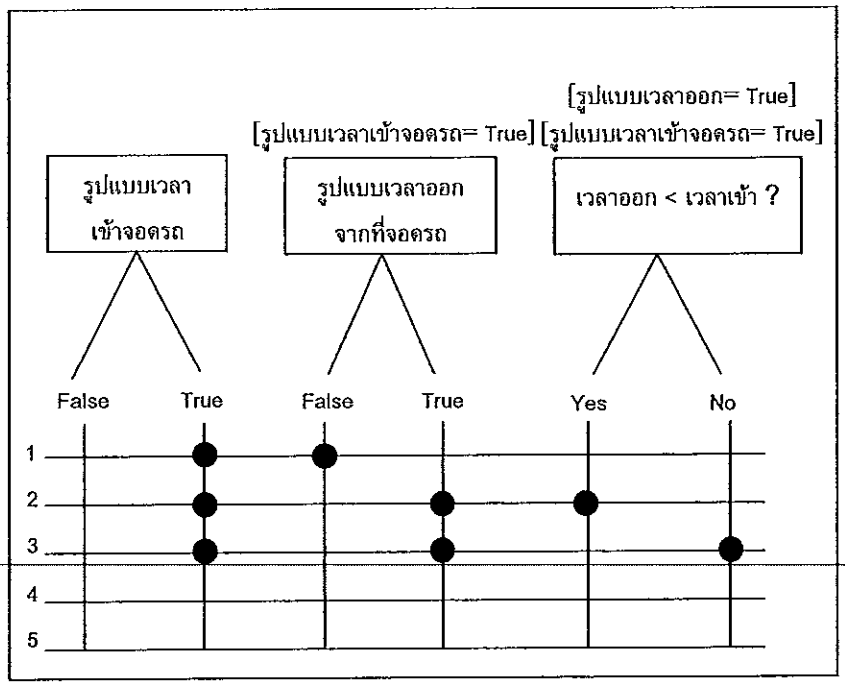


ภาพประกอบที่ 3.7 การระบุกรณีทดสอบจากการพิจารณาเงื่อนไขของต้นไม้การจำแนก ***รูปแบบเวลาออกจากที่จอดรถ***

พิจารณาต้นไม้การจำแนกต้นที่ 3 พบว่ามีเงื่อนไข [รูปแบบเวลาเข้าจอดรถ=True] และ [รูปแบบเวลาออกจากที่จอดรถ=True] ดังนั้นการระบุกรณีทดสอบจึงทำสัญลักษณ์บนตารางกรณีทดสอบได้ดังต่อไปนี้

1) จากเงื่อนไขที่ปรากฏพบว่ามีเงื่อนไขที่ซ้ำกับเงื่อนไขที่เคยปรากฏในต้นไม้ก่อนหน้าแล้ว คือ [รูปแบบเวลาเข้าจอดรถ=True] ดังนั้นการระบุกรณีทดสอบจะพิจารณาต่อจากเส้นที่เคยระบุกรณีทดสอบมาแล้ว ในที่นี้พิจารณาเส้นที่ 2 ซึ่งได้มาจากเงื่อนไข [รูปแบบเวลาเข้าจอดรถ=True] และ [รูปแบบเวลาออกจากที่จอดรถ=True] ดังนั้นจึงทำสัญลักษณ์ต่อในเส้นที่ 2 โดยเพิ่มสัญลักษณ์บนเส้นที่ลากจากโหนด Yes ของต้นไม้การจำแนก ***เวลาออก < เวลาเข้า ?***

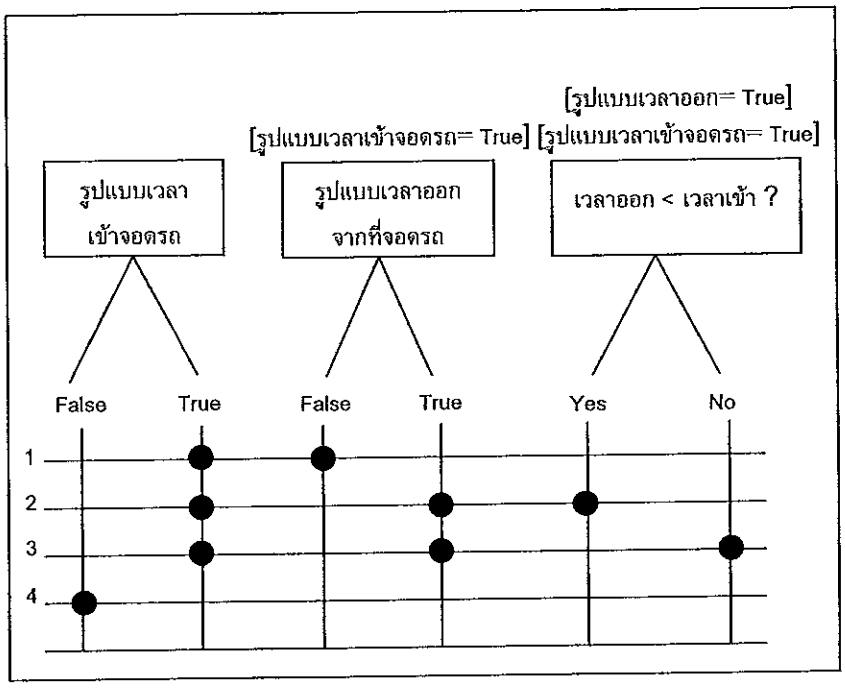
2) เส้นที่ 3 ทำสัญลักษณ์ให้เหมือนกับเส้นที่ 2 แต่เปลี่ยนเป็นเลือกโหนด No ของต้นไม้การจำแนก ***เวลาออก < เวลาเข้า ?*** การระบุกรณีทดสอบจากการพิจารณาเงื่อนไขของต้นไม้การจำแนก ***เวลาออก < เวลาเข้า ?*** สามารถแสดงได้ดังภาพประกอบที่ 3.8



ภาพประกอบที่ 3.8 การระบุกรณีทดสอบจากการพิจารณาเงื่อนไขของต้นไม้การจำแนก ***เวลาออก < เวลาเข้า ?***

พิจารณาต้นไม้การจำแนก ***รูปแบบเวลาเข้าจอดรถ*** พบว่ามีโหนด False ไม่มีการทำสัญลักษณ์ปรากฏอยู่เลยจึงทำการระบุเป็นกรณีทดสอบ 1 กรณี โดยเส้นที่ 4 ทำ

สัญลักษณ์บนเส้นที่ลากจากโหนด False ของต้นไม้การจำแนก ***รูปแบบเวลาเข้าจอดรถ*** เพียงโหนดเท่านั้น ซึ่งการระบุกรณีทดสอบจากการพิจารณาเงื่อนไขของต้นไม้การจำแนกทั้งหมดสามารถแสดงได้ดังภาพประกอบที่ 3.9



ภาพประกอบที่ 3.9 กรณีทดสอบจากการพิจารณาแผนภาพกิจกรรมของโปรแกรมคำนวณค่าจอดรถ

ดังนั้นกรณีทดสอบที่ได้จากแผนภาพกิจกรรมของโปรแกรมคำนวณค่าจอดรถมีทั้งหมด 4 กรณีดังนี้

กรณีที่ 1 รูปแบบเวลาเข้าจอดรถ = True, รูปแบบเวลาออกจากที่จอดรถ = False

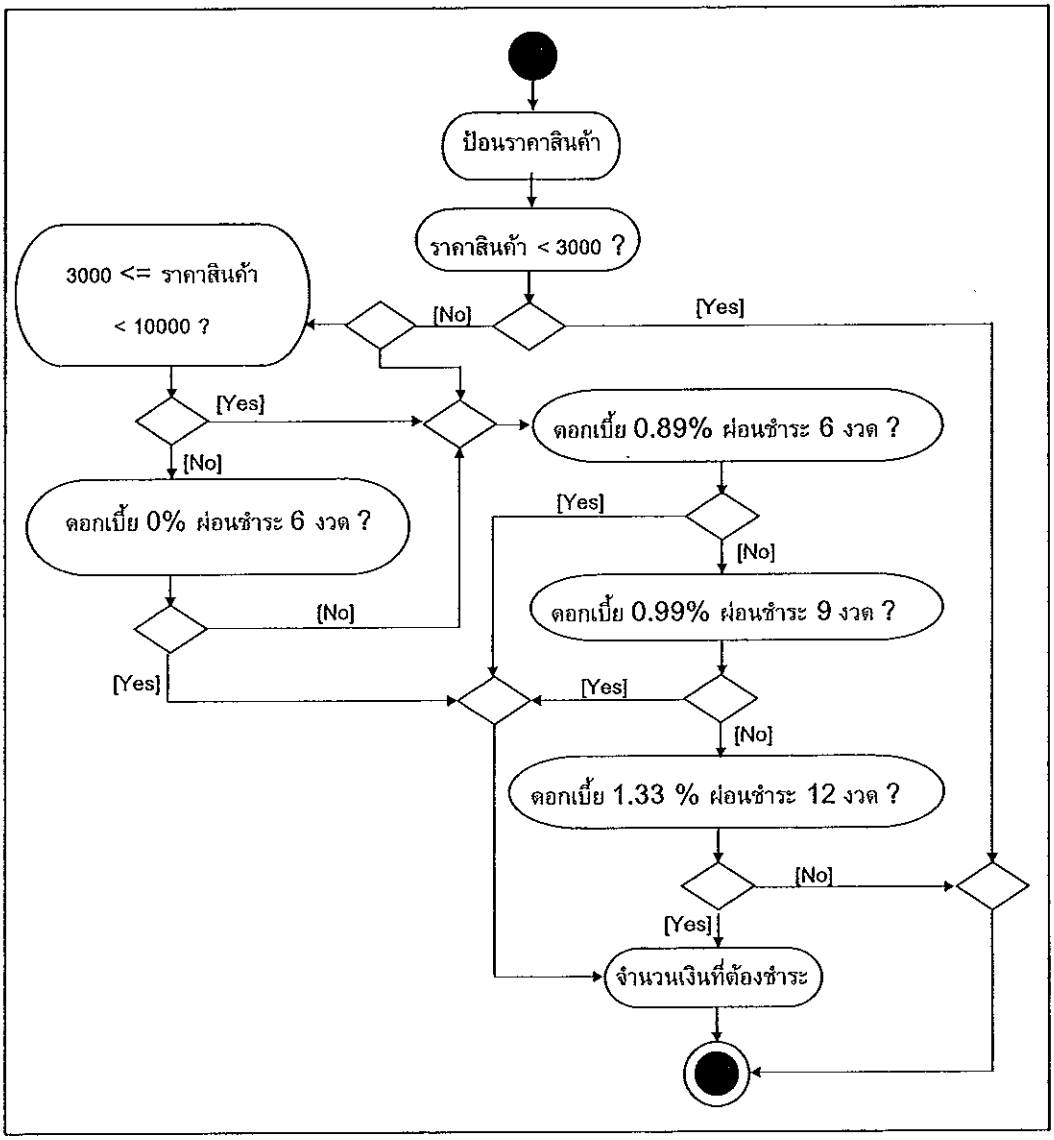
กรณีที่ 2 รูปแบบเวลาเข้าจอดรถ = True, รูปแบบเวลาออกจากที่จอดรถ = True, เวลาออก < เวลาเข้า ? = Yes

กรณีที่ 3 รูปแบบเวลาเข้าจอดรถ = True, รูปแบบเวลาออกจากที่จอดรถ = True, เวลาออก < เวลาเข้า ? = No

กรณีที่ 4 รูปแบบเวลาเข้าจอดรถ = False

3.2.2 กรณีตัวอย่างที่ 2 การสร้างกรณีทดสอบจากแผนภาพกิจกรรมของโปรแกรมคำนวณจำนวนเงินที่ต้องผ่อนชำระต่องวด

รายละเอียดแผนภาพกิจกรรมของโปรแกรมคำนวณจำนวนเงินที่ต้องผ่อนชำระต่องวดเริ่มต้นจากการป้อนราคาสินค้าที่ต้องการคำนวณค่างวด จากนั้นจะทำการตรวจสอบว่าราคาสินค้าที่ป้อนน้อยกว่า 3,000 หรือไม่ถ้าน้อยกว่าจะจบการทำงานแต่ถ้าไม่น้อยกว่าจะทำการตรวจสอบว่าราคาสินค้าน้อยกว่า 10,000 หรือไม่ถ้าใช่ผู้ใช้มีสิทธิ์เลือกรูปแบบการผ่อนชำระดอกเบี้ยได้ 3 แบบคือดอกเบี้ย 0.89 % ผ่อนชำระ 6 เดือน หรือ การผ่อนชำระที่ดอกเบี้ย 0.99 % ผ่อนชำระ 9 เดือน หรือเลือกการผ่อนชำระที่ดอกเบี้ย 1.33 % ผ่อนชำระ 12 เดือน แต่หากราคาสินค้าที่ป้อนมีค่าตั้งแต่ 10,000 ขึ้นไปผู้ใช้สามารถเลือกการผ่อนชำระที่ดอกเบี้ย 0 % ผ่อนชำระ 6 เดือนได้ แต่ถ้าผู้ใช้ไม่ต้องการเลือกผ่อน 0% ผู้ใช้สามารถเลือกรูปแบบอื่นๆ ได้ซึ่งมีรายละเอียดดังภาพประกอบที่ 3.10



ภาพประกอบที่ 3.10 แผนภาพกิจกรรมของโปรแกรมคำนวณจำนวนเงินที่ต้องผ่อนชำระต่องวด (พนิดา พานิชกุล, 2548)

จากแผนภาพกิจกรรมของโปรแกรมคำนวณจำนวนเงินที่ต้องผ่อนชำระต่องวด สามารถสร้างกรณีทดสอบโดยหลักการ CCTM ได้ดังต่อไปนี้

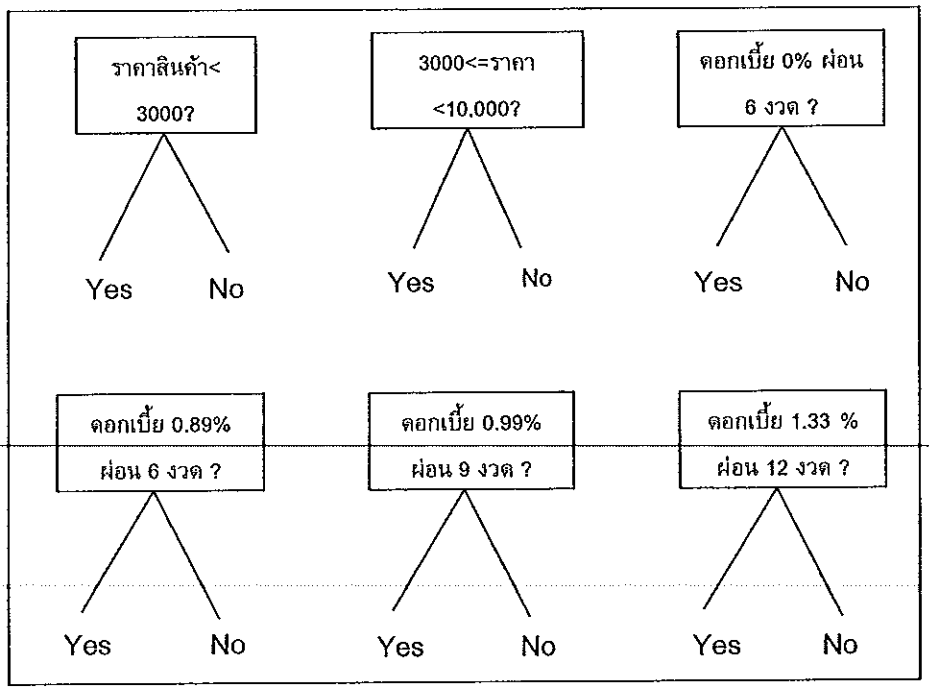
ขั้นตอนที่ 1 วิเคราะห์และสร้างต้นไม้การจำแนกแบบมีเงื่อนไข

1) วิเคราะห์แผนภาพกิจกรรมพบว่ามีส่วนการตัดสินใจ (decision point) ที่เป็นทางเลือกปรากฏอยู่ทั้งหมด 6 ตำแหน่งซึ่งสามารถแบ่งออกเป็น 6 categories และแต่ละ category ประกอบด้วย 2 options รายละเอียดได้ดังตารางที่ 3.2

ตารางที่ 3.2 input domain จากการวิเคราะห์แผนภาพกิจกรรมของโปรแกรมคำนวณจำนวนเงินที่ต้องผ่อนชำระต่องวด

Category	Option	
1. ราคาสินค้า< 3000?	Yes	No
2. 3000<=ราคา<10,000?	Yes	No
3. ดอกเบี้ย 0% ผ่อน 6 งวด?	Yes	No
4. ดอกเบี้ย 0.89% ผ่อน 6 งวด?	Yes	No
5. ดอกเบี้ย 0.99% ผ่อน 9 งวด?	Yes	No
6. ดอกเบี้ย 1.33 % ผ่อน 12 งวด?	Yes	No

2) จาก category และ option สามารถสร้างต้นไม้การจำแนกได้ทั้งหมด 6 ต้น ซึ่งแสดงได้ดังภาพประกอบที่ 3.11



ภาพประกอบที่ 3.11 ต้นไม้การจำแนกแสดง input domain จากการวิเคราะห์แผนภาพกิจกรรมของโปรแกรมคำนวณจำนวนเงินที่ต้องผ่อนชำระต่องวด

ระบุเงื่อนไขให้กับต้นไม้การจำแนกแต่ละต้น โดยพิจารณาต้นไม้ทีละต้น ซึ่งพิจารณาจากเส้นทางก่อนเข้าถึงส่วนการตัดสินใจของต้นไม้การจำแนกที่กำลังพิจารณาว่าได้ผ่านเงื่อนไขส่วนการตัดสินใจส่วนใดมาบ้างให้นำเงื่อนไขการตัดสินใจเหล่านั้นมาระบุไว้บนต้นไม้การจำแนกภายในเครื่องหมายวงเล็บก้ามปู [ ]

จากต้นไม้การจำแนกทั้ง 6 ต้น เริ่มต้นพิจารณาจากต้นไม้การจำแนก *ราคาสินค้า < 3,000?* พบว่าส่วนการตัดสินใจ *ราคาสินค้า < 3,000?* ไม่ได้ผ่านส่วนการตัดสินใจอื่นมาเลยจึงไม่ต้องระบุเงื่อนไขให้กับต้นไม้การจำแนก *ราคาสินค้า < 3,000?*

พิจารณาต้นไม้การจำแนก *3,000 <= ราคาสินค้า < 10,000?* พบว่าส่วนการตัดสินใจ *3,000 <= ราคาสินค้า < 10,000?* ต้องผ่านส่วนการตัดสินใจ *ราคาสินค้า < 3,000?* โดยมีเงื่อนไขเป็น No ดังนั้นระบุเงื่อนไข [ราคาสินค้า < 3,000? = No] ให้กับต้นไม้การจำแนก *3,000 <= ราคาสินค้า < 10,000?*

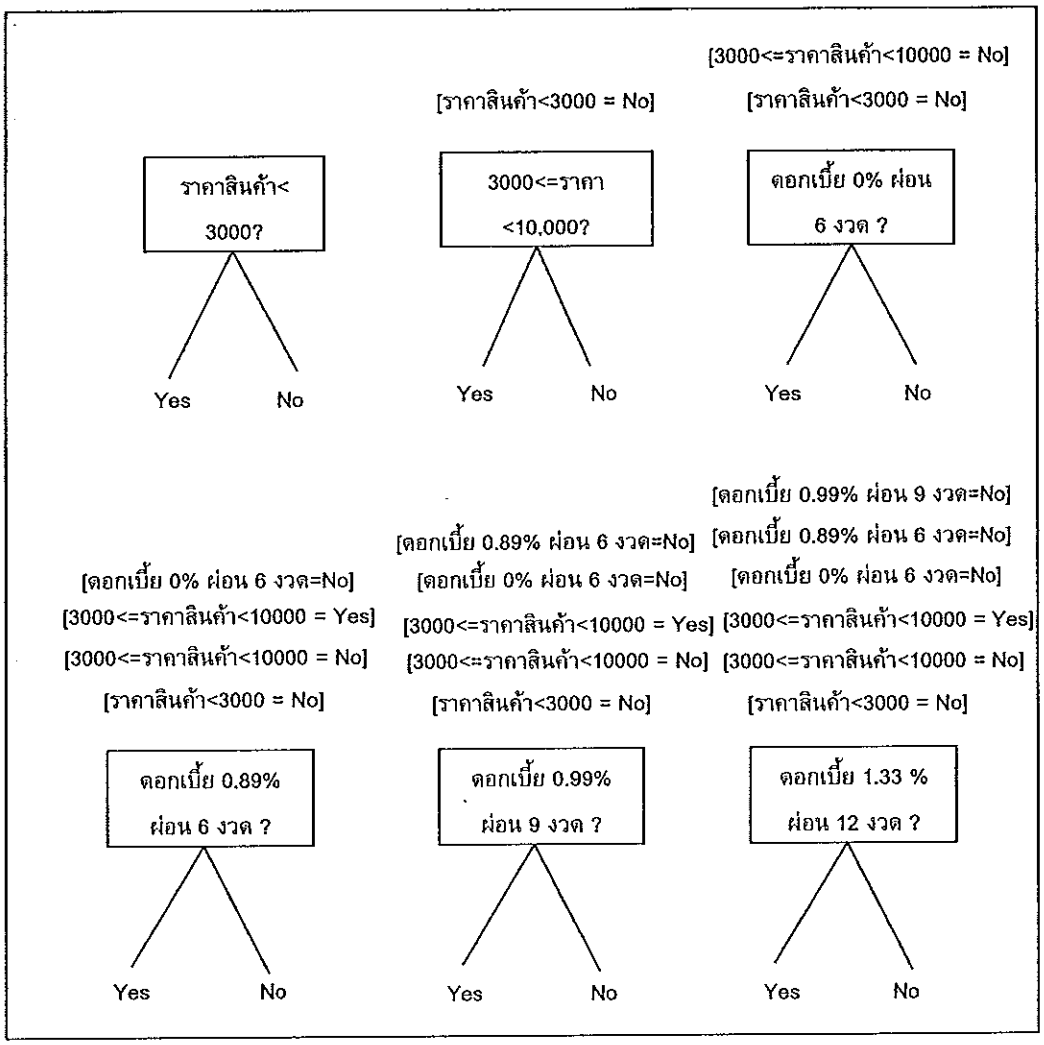
พิจารณาต้นไม้การจำแนก *ดอกเบี้ย 0% ผ่อนชำระ 6 งวด?* พบว่าส่วนการตัดสินใจ *ดอกเบี้ย 0% ผ่อนชำระ 6 งวด?* ต้องผ่านส่วนการตัดสินใจ *ราคาสินค้า < 3,000?* โดยมีเงื่อนไขเป็น No และต้องผ่านส่วนการตัดสินใจ *3,000 <= ราคาสินค้า < 10,000?* โดยมีเงื่อนไขเป็น No ดังนั้นระบุเงื่อนไข [ราคาสินค้า < 3,000? = No] และ [3,000 <= ราคาสินค้า < 10,000? = No] ให้กับต้นไม้การจำแนก *ดอกเบี้ย 0% ผ่อนชำระ 6 งวด?*

พิจารณาต้นไม้การจำแนก *ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด?* พบว่าส่วนการตัดสินใจ *ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด?* ต้องผ่านส่วนการตัดสินใจ *ราคาสินค้า < 3,000?* โดยมีเงื่อนไขเป็น No และต้องผ่านส่วนการตัดสินใจ *3,000 <= ราคาสินค้า < 10,000?* โดยมีเงื่อนไขเป็น No ผ่านส่วนการตัดสินใจ *3,000 <= ราคาสินค้า < 10,000?* โดยมีเงื่อนไขเป็น Yes และผ่านส่วนการตัดสินใจ *ดอกเบี้ย 0% ผ่อนชำระ 6 งวด?* โดยมีเงื่อนไขเป็น No ดังนั้นระบุเงื่อนไข [ราคาสินค้า < 3,000? = No] และ [3,000 <= ราคาสินค้า < 10,000? = No] [3,000 <= ราคาสินค้า < 10,000? = Yes] และเงื่อนไข [ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด? = No] ให้กับต้นไม้การจำแนก *ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด?*

พิจารณาต้นไม้การจำแนก *ดอกเบี้ย 0.99% ผ่อนชำระ 9 งวด?* พบว่าส่วนการตัดสินใจ *ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด?* ต้องผ่านส่วนการตัดสินใจ *ราคาสินค้า < 3,000 ?* โดยมีเงื่อนไขเป็น No และต้องผ่านส่วนการตัดสินใจ *3,000 <= ราคาสินค้า < 10,000?* โดยมีเงื่อนไขเป็น No ผ่านส่วนการตัดสินใจ *3,000 <= ราคาสินค้า < 10,000?* โดยมี

เงื่อนไขเป็น Yes ผ่านส่วนการตัดสินใจ ***ดอกเบี้ย 0% ผ่อนชำระ 6 งวด?*** โดยมีเงื่อนไขเป็น No และผ่านส่วนการตัดสินใจ ***ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด?*** โดยมีเงื่อนไขเป็น No ดังนั้นระบุเงื่อนไข [ราคาสินค้า < 3,000 ? = No] และ [3,000 <= ราคาสินค้า < 10,000 ? = No] [3,000 <= ราคาสินค้า < 10,000 ? = Yes] เงื่อนไข [ดอกเบี้ย 0% ผ่อนชำระ 6 งวด? = No] และเงื่อนไข [ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด? = No] ให้กับต้นไม้การจำแนก ***ดอกเบี้ย 0.99% ผ่อนชำระ 9 งวด?***

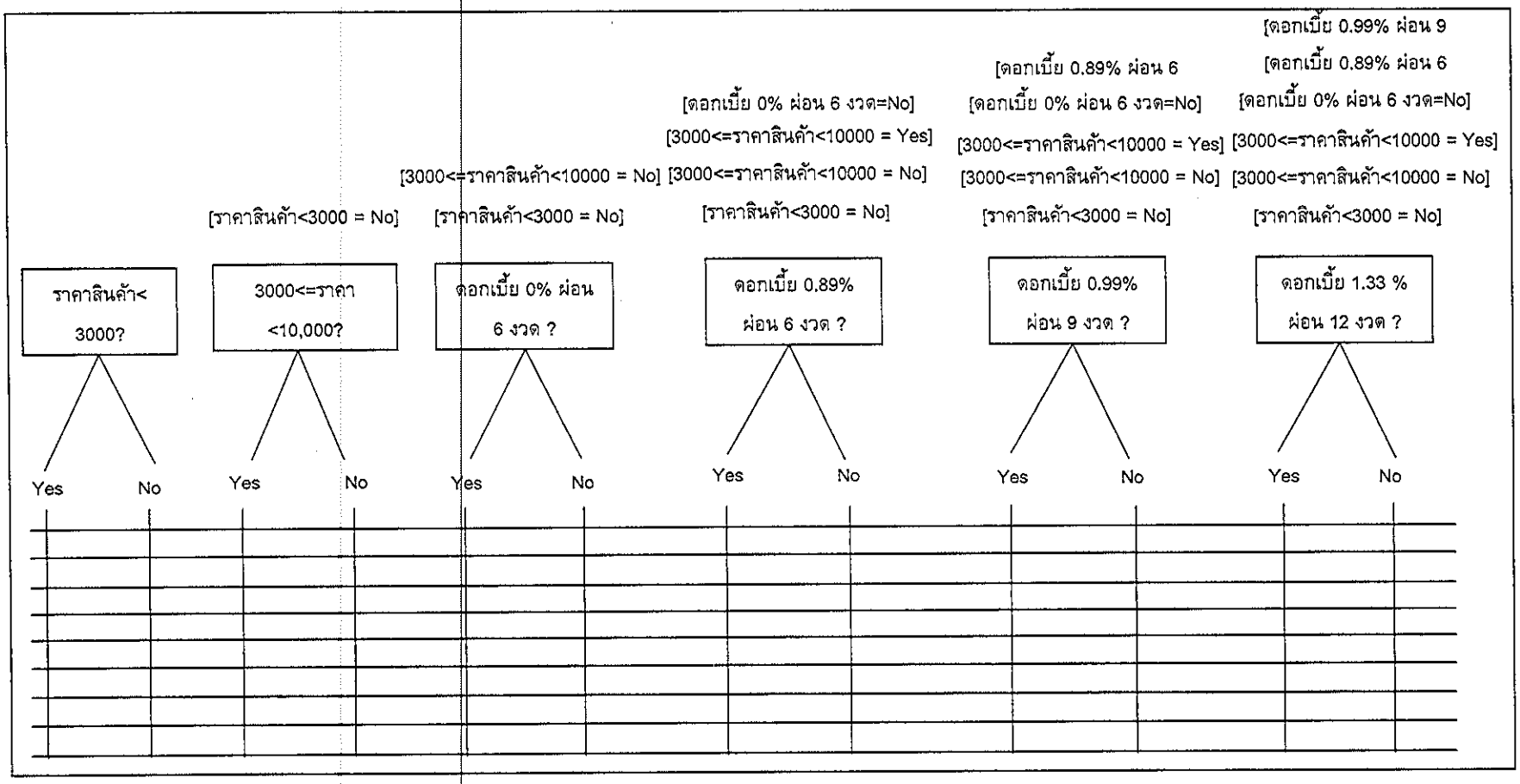
พิจารณาต้นไม้การจำแนก ***ดอกเบี้ย 1.33% ผ่อนชำระ 12 งวด?*** พบว่าส่วนการตัดสินใจ ***ดอกเบี้ย 1.33% ผ่อนชำระ 12 งวด?*** ต้องผ่านส่วนการตัดสินใจ ***ราคาสินค้า < 3,000 ?*** โดยมีเงื่อนไขเป็น No และต้องผ่านส่วนการตัดสินใจ ***3,000 <= ราคาสินค้า < 10,000?*** โดยมีเงื่อนไขเป็น No ผ่านส่วนการตัดสินใจ ***3,000 <= ราคาสินค้า < 10,000 ?*** โดยมีเงื่อนไขเป็น Yes ผ่านส่วนการตัดสินใจ ***ดอกเบี้ย 0% ผ่อนชำระ 6 งวด?*** โดยมีเงื่อนไขเป็น No ผ่านส่วนการตัดสินใจ ***ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด?*** โดยมีเงื่อนไขเป็น No และผ่านส่วนการตัดสินใจ ***ดอกเบี้ย 0.99% ผ่อนชำระ 9 งวด?*** โดยมีเงื่อนไขเป็น No ดังนั้นระบุเงื่อนไข [ราคาสินค้า < 3,000 ? = No] และ [3,000 <= ราคาสินค้า < 10,000 ? = No] [3,000 <= ราคาสินค้า < 10,000 ? = Yes] เงื่อนไข [ดอกเบี้ย 0% ผ่อนชำระ 6 งวด? = No] เงื่อนไข [ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด? = No] และเงื่อนไข [ดอกเบี้ย 0.99% ผ่อนชำระ 9 งวด? = No] ให้กับต้นไม้การจำแนก ***ดอกเบี้ย 1.33% ผ่อนชำระ 12 งวด?*** ซึ่งต้นไม้การจำแนกทั้งหมดที่มีการระบุเงื่อนไขสามารถแสดงได้ดังภาพประกอบที่ 3.12



ภาพประกอบที่ 3.12 ต้นไม้การจำแนกที่ระบุเงื่อนไขจากการวิเคราะห์แผนภาพกิจกรรมของโปรแกรมคำนวณจำนวนเงินที่ต้องผ่อนชำระต่องวด

ขั้นตอนที่ 2 สร้างตารางกรณีทดสอบ

นำต้นไม้การจำแนกที่ระบุเงื่อนไขทั้งหมดมาวางเรียงกันโดยเรียงตามลำดับจำนวนเงื่อนไขที่ระบุบนต้นไม้จากจำนวนน้อยไปหาจำนวนมาก และสร้างตาราง โดยลากเส้นคอลัมน์จากโหนดลูกของต้นไม้การจำแนกที่ระบุเงื่อนไขแต่ละต้นและลากเส้นแถวของตารางเพื่อใช้กำหนดกรณีทดสอบซึ่งจากตัวอย่างข้างต้นสามารถแสดงต้นไม้การจำแนกที่ระบุเงื่อนไขและตารางกรณีทดสอบได้ดังภาพประกอบที่ 3.13



ภาพประกอบที่ 3.13 ต้นไม้การจำแนกที่ระบุเงื่อนไขและตารางกรณีทดสอบแสดง input domain จากการวิเคราะห์แผนภาพกิจกรรมของโปรแกรมคำนวณจำนวนเงินที่ต้องผ่อนชำระต่องวด

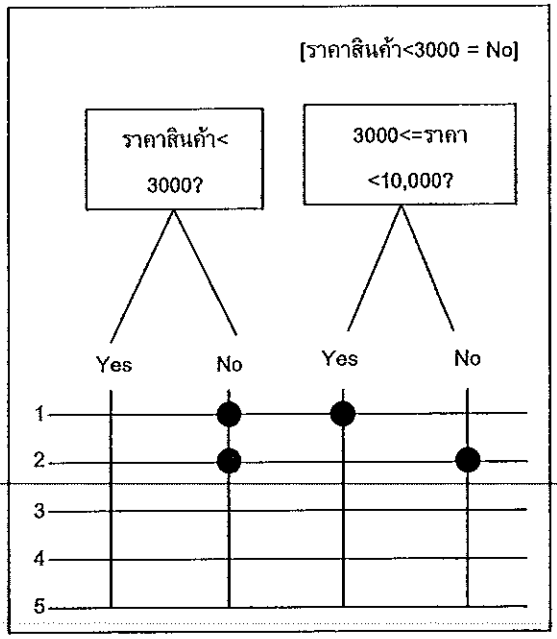
ขั้นตอนที่ 3 สร้างกรณีทดสอบ

การกำหนดกรณีทดสอบในที่นี้จะเริ่มพิจารณาจากต้นไม้การจำแนกต้นที่ 1 พบว่าไม่มีเงื่อนไขที่ระบุไว้

พิจารณาต้นไม้การจำแนกต้นที่ 2 พบว่ามีเงื่อนไข [ราคาสินค้า < 3,000 ? = No] ดังนั้นการระบุกรณีทดสอบจึงทำสัญลักษณ์บนตารางกรณีทดสอบได้ดังต่อไปนี้

1) เส้นที่ 1 ทำสัญลักษณ์บนเส้นที่ลากจากโหนด No ของต้นไม้การจำแนก ***ราคาสินค้า < 3,000 ?*** และทำสัญลักษณ์บนเส้นที่ลากจากโหนด Yes ของต้นไม้การจำแนก ***3,000 <= ราคาสินค้า < 10,000 ?***

2) เส้นที่ 2 ทำสัญลักษณ์บนเส้นที่ลากจากโหนด No ของต้นไม้การจำแนก ***ราคาสินค้า < 3,000 ?*** และทำสัญลักษณ์บนเส้นที่ลากจากโหนด No ของต้นไม้การจำแนก ***3,000 <= ราคาสินค้า < 10,000 ?*** การระบุกรณีทดสอบจากการพิจารณาเงื่อนไขของต้นไม้การจำแนก ***3,000 <= ราคาสินค้า < 10,000 ?*** สามารถแสดงได้ดังภาพประกอบที่ 3.14

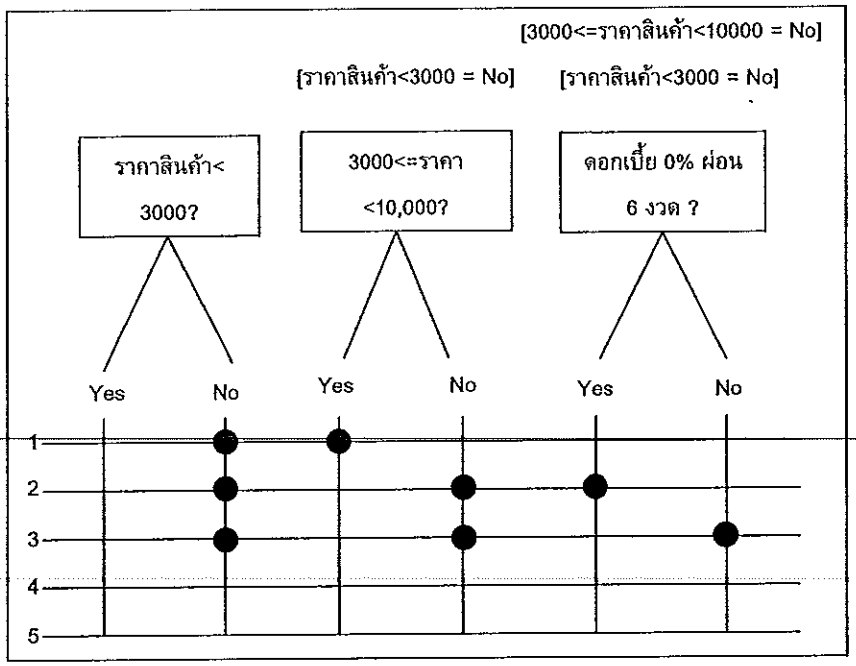


ภาพประกอบที่ 3.14 การระบุกรณีทดสอบจากการพิจารณาเงื่อนไขของต้นไม้การจำแนก ***3,000 <= ราคาสินค้า < 10,000 ?***

พิจารณาต้นไม้การจำแนกต้นที่ 3 พบว่ามีเงื่อนไข [ราคาสินค้า < 3,000 ? = No] และ [3,000 <= ราคาสินค้า < 10,000 ? = No] ดังนั้นการระบุกรณีทดสอบจึงทำสัญลักษณ์บนตารางกรณีทดสอบได้ดังต่อไปนี้

1) จากเงื่อนไขที่ปรากฏพบว่ามีเงื่อนไขที่ซ้ำกับเงื่อนไขที่เคยปรากฏในต้นไม้ก่อนหน้าแล้ว คือ [ราคาสินค้า < 3,000 ? = No] ดังนั้นการระบุกรณีทดสอบจะพิจารณาต่อจากเส้นที่เคยระบุกรณีทดสอบมาแล้ว ในที่นี้พิจารณาเส้นที่ 2 ซึ่งได้มาจากเงื่อนไข [ราคาสินค้า < 3,000 ? = No] และ [3,000 <= ราคาสินค้า < 10,000 ? = No] ดังนั้นจึงทำสัญลักษณ์ต่อในเส้นที่ 2 โดยเพิ่มสัญลักษณ์บนเส้นที่ลากจากโหนด Yes ของต้นไม้การจำแนก ***ดอกเบี้ย 0% ผ่อนชำระ 6 งวด?***

2) เส้นที่ 3 ทำสัญลักษณ์ให้เหมือนกับเส้นที่ 2 แต่เปลี่ยนจากโหนด Yes ของต้นไม้การจำแนก ***ดอกเบี้ย 0% ผ่อนชำระ 6 งวด?*** เป็นโหนด No การระบุกรณีทดสอบจากการพิจารณาเงื่อนไขของต้นไม้การจำแนก ***ดอกเบี้ย 0% ผ่อนชำระ 6 งวด?*** สามารถแสดงได้ดังภาพประกอบที่ 3.15

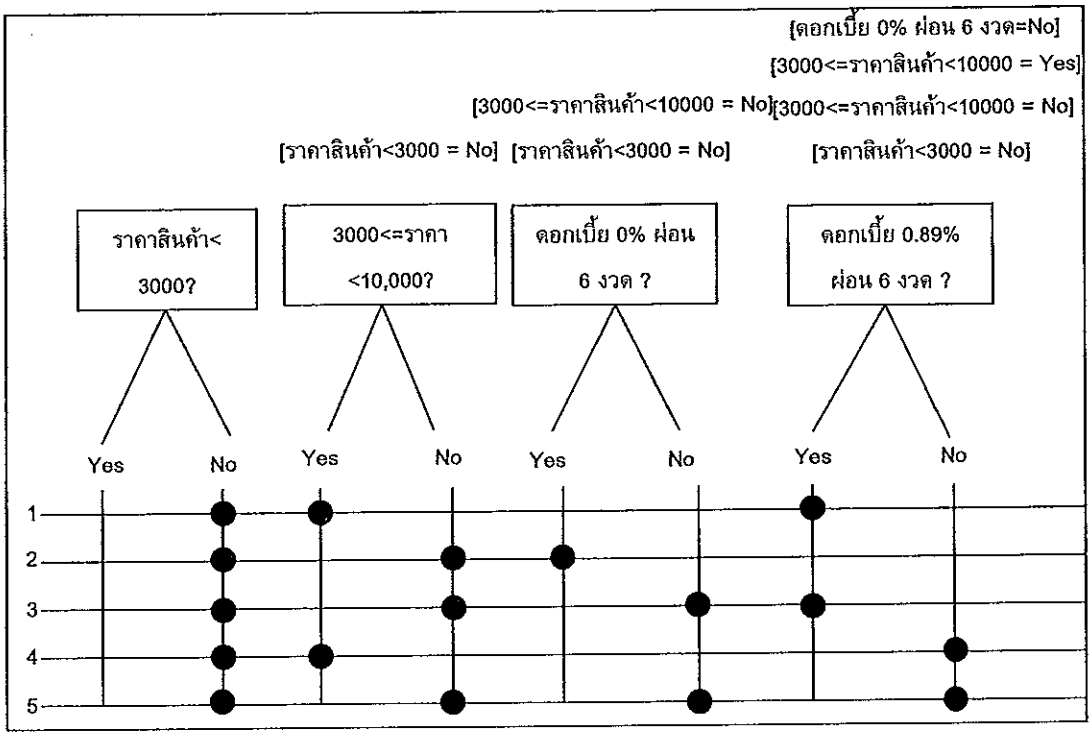


ภาพประกอบที่ 3.15 การระบุกรณีทดสอบจากการพิจารณาเงื่อนไขของต้นไม้การจำแนก ***ดอกเบี้ย 0% ผ่อนชำระ 6 งวด?***

พิจารณาต้นไม้การจำแนกต้นที่ 4 พบว่ามีเงื่อนไข [ราคาสินค้า < 3,000 ? = No] [3,000 <= ราคาสินค้า < 10,000 ? = No] [3,000 <= ราคาสินค้า < 10,000 ? = Yes] และ [ดอกเบี้ย 0% ผ่อนชำระ 6 งวด ? = No] ดังนั้นการระบุกรณีทดสอบจึงทำสัญลักษณ์บนตารางกรณีทดสอบได้ดังต่อไปนี้

1) จากเงื่อนไขที่ปรากฏพบว่ามีเงื่อนไขที่ซ้ำกับเงื่อนไขที่เคยปรากฏในต้นไม้ก่อนหน้าแล้ว คือ [ราคาสินค้า < 3,000 ? = No] และ [3,000 <= ราคาสินค้า < 10,000 ? = No] ดังนั้นการระบุกรณีทดสอบจะพิจารณาต่อจากเส้นที่เคยระบุกรณีทดสอบมาแล้ว ในที่นี้พิจารณาเส้นที่ 1 ซึ่งได้มาจากเงื่อนไข [ราคาสินค้า < 3,000 ? = No] และ [3,000 <= ราคาสินค้า < 10,000 ? = Yes] ดังนั้นจึงทำสัญลักษณ์ต่อในเส้นที่ 1 โดยเพิ่มสัญลักษณ์บนเส้นที่ลากจากโหนด Yes ของต้นไม้การจำแนก ***ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด?*** และในเส้นที่ 4 ทำสัญลักษณ์เหมือนกับเส้นที่ 1 แต่เปลี่ยนจากโหนด Yes ของต้นไม้การจำแนก ***ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด?*** เป็นโหนด No

2) พิจารณาเส้นที่ 3 ซึ่งได้มาจากเงื่อนไข [ราคาสินค้า < 3,000 ? = No] [3,000 <= ราคาสินค้า < 10,000 ? = No] และ [ดอกเบี้ย 0% ผ่อนชำระ 6 งวด? = No] ดังนั้นจึงทำสัญลักษณ์ต่อในเส้นที่ 3 โดยเพิ่มสัญลักษณ์บนเส้นที่ลากจากโหนด Yes ของต้นไม้การจำแนก ***ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด?*** และในเส้นที่ 5 ทำสัญลักษณ์เหมือนกับเส้นที่ 3 แต่เปลี่ยนจากโหนด Yes ของต้นไม้การจำแนก ***ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด?*** เป็นโหนด No การระบุกรณีทดสอบจากการพิจารณาเงื่อนไขของต้นไม้การจำแนก ***ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด?*** สามารถแสดงได้ดังภาพประกอบที่ 3.16

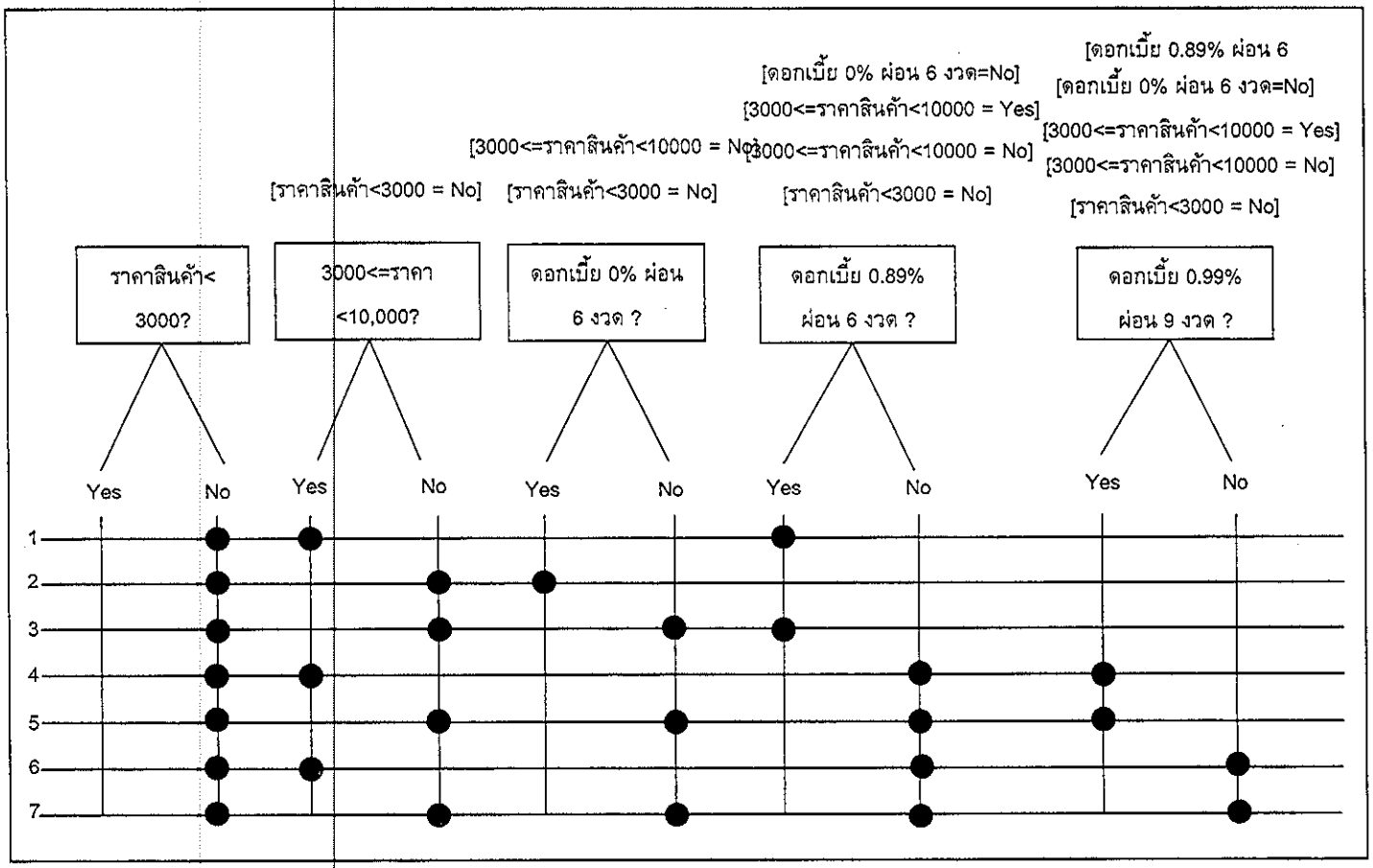


ภาพประกอบที่ 3.16 การระบุกรณีทดสอบจากการพิจารณาเงื่อนไขของต้นไม้การจำแนก ***ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด?***

พิจารณาต้นไม้การจำแนกต้นที่ 5 พบว่ามีเงื่อนไข [ราคาสินค้า < 3,000 ? = No] [3,000 <= ราคาสินค้า < 10,000 ? = No] [3,000 <= ราคาสินค้า < 10,000 ? = Yes] [ดอกเบี้ย 0% ผ่อนชำระ 6 งวด? = No] และ [ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด? = No] ดังนั้น การระบุกรณีทดสอบจึงทำสัญลักษณ์บนตารางกรณีทดสอบได้ดังต่อไปนี้

1) จากเงื่อนไขที่ปรากฏพบว่ามีเงื่อนไขที่ซ้ำกับเงื่อนไขที่เคยปรากฏในต้นไม้ก่อนหน้าแล้ว คือ [ราคาสินค้า < 3,000 ? = No] [3,000 <= ราคาสินค้า < 10,000 ? = No] [3,000 <= ราคาสินค้า < 10,000 ? = Yes] และ [ดอกเบี้ย 0% ผ่อนชำระ 6 งวด? = No] ดังนั้น การระบุกรณีทดสอบจะพิจารณาต่อจากเส้นที่เคยระบุกรณีทดสอบด้วย ในที่นี้พิจารณาเส้นที่ 4 ซึ่งได้มาจากเงื่อนไข [ราคาสินค้า < 3,000 ? = No] [3,000 <= ราคาสินค้า < 10,000 ? = Yes] และ [ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด? = No] ดังนั้นจึงทำสัญลักษณ์ต่อในเส้นที่ 4 โดยเพิ่มสัญลักษณ์บนเส้นที่ลากจากโหนด Yes ของต้นไม้การจำแนก ***ดอกเบี้ย 0.99% ผ่อนชำระ 9 งวด?*** และในเส้นที่ 6 ทำสัญลักษณ์เหมือนกับเส้นที่ 4 แต่เปลี่ยนจากโหนด Yes ของต้นไม้การจำแนก ***ดอกเบี้ย 0.99% ผ่อนชำระ 9 งวด?*** เป็นโหนด No

2) พิจารณาเส้นที่ 5 ซึ่งได้มาจากเงื่อนไข [ราคาสินค้า < 3,000 ? = No] [3,000 <= ราคาสินค้า < 10,000 ? = No] [ดอกเบี้ย 0% ผ่อนชำระ 6 งวด? = No] และ [ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด? = No] ดังนั้นจึงทำสัญลักษณ์ต่อในเส้นที่ 5 โดยเพิ่มสัญลักษณ์บนเส้นที่ลากจากโหนด Yes ของต้นไม้การจำแนก ***ดอกเบี้ย 0.99% ผ่อนชำระ 9 งวด?*** และในเส้นที่ 7 ทำสัญลักษณ์เหมือนกับเส้นที่ 5 แต่เปลี่ยนจากโหนด Yes ของต้นไม้การจำแนก ***ดอกเบี้ย 0.99% ผ่อนชำระ 9 งวด?*** เป็นโหนด No การระบุกรณีทดสอบจากการพิจารณาเงื่อนไขของต้นไม้การจำแนก ***ดอกเบี้ย 0.99% ผ่อนชำระ 9 งวด?*** สามารถแสดงได้ดังภาพประกอบที่ 3.17

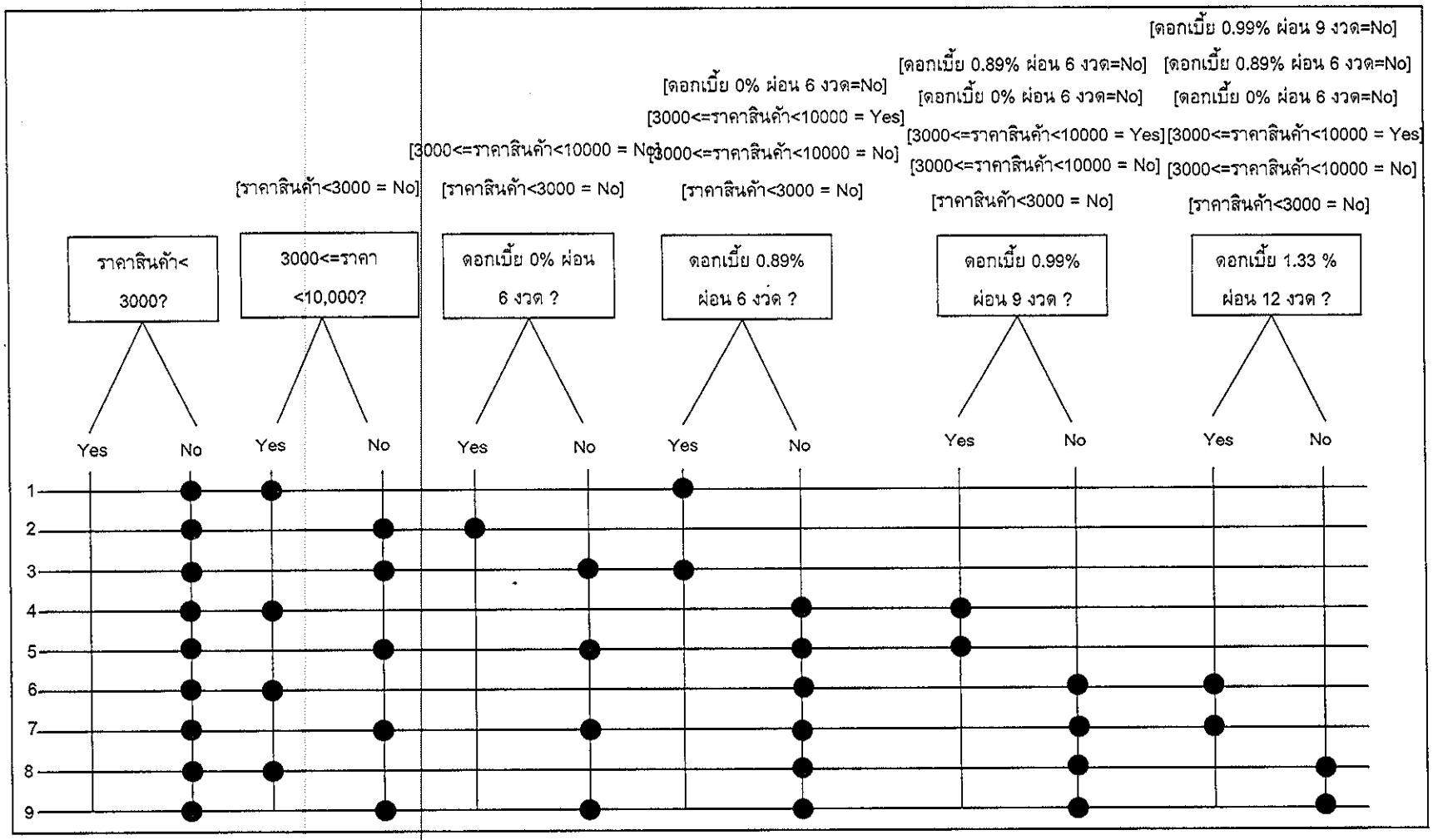


ภาพประกอบที่ 3.17 การระบุกรณีทดสอบจากการพิจารณาเงื่อนไขของต้นไม้การจำแนก ***ดอกเบี้ย 0.99% ผ่อนชำระ 9 งวด?***

พิจารณาต้นไม้การจำแนกต้นที่ 6 พบว่ามีเงื่อนไข [ราคาสินค้า < 3,000 ? = No] [3,000 <= ราคาสินค้า < 10,000 ? = No] [3,000 <= ราคาสินค้า < 10,000 ? = Yes] [ดอกเบี้ย 0% ผ่อนชำระ 6 งวด? = No] [ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด? = No] และ [ดอกเบี้ย 0.99% ผ่อนชำระ 9 งวด? = No] ดังนั้นการระบุกรณีทดสอบจึงทำสัญลักษณ์บนตารางกรณีทดสอบได้ดังต่อไปนี้

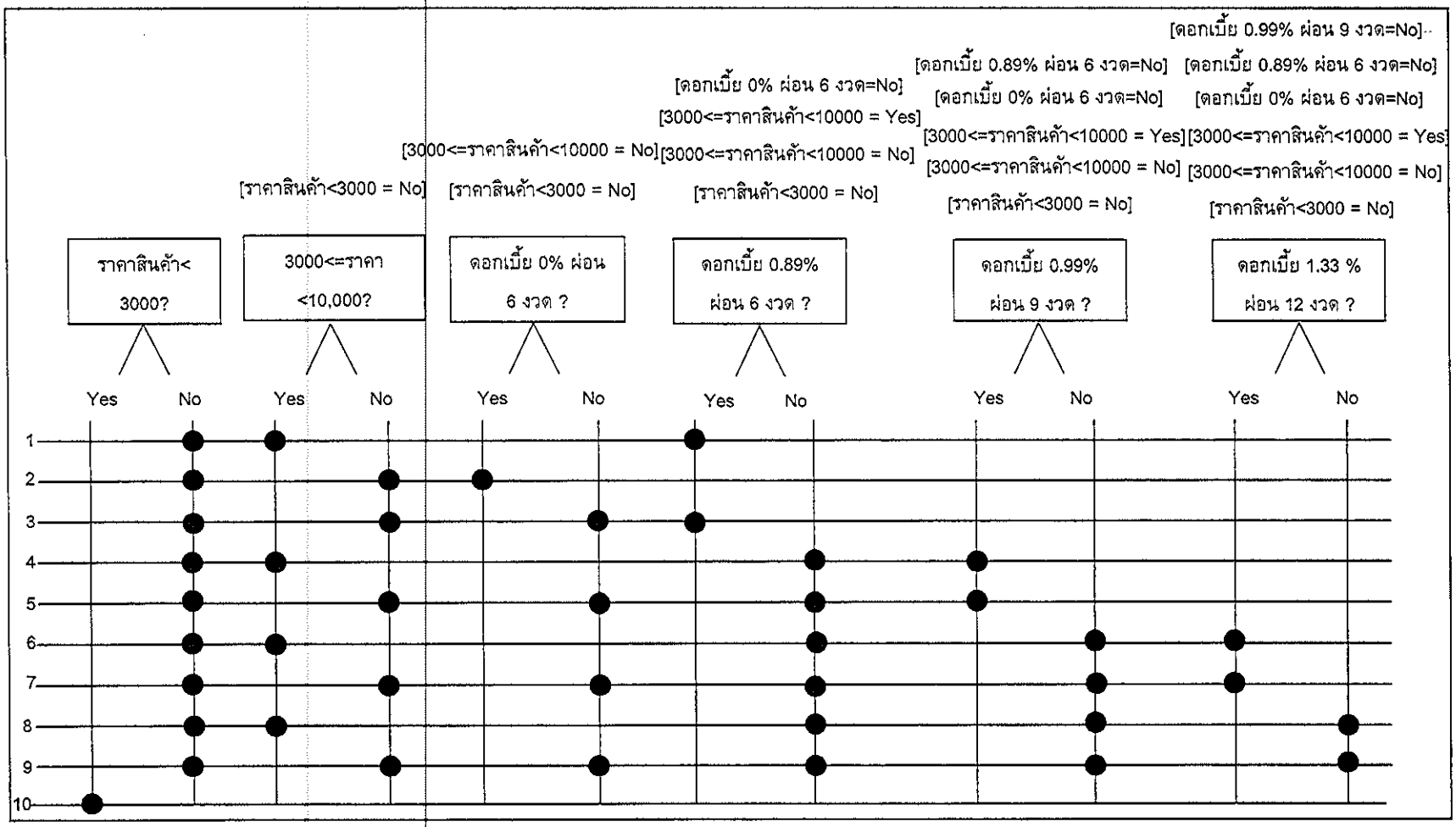
1) จากเงื่อนไขที่ปรากฏพบว่ามีเงื่อนไขที่ซ้ำกับเงื่อนไขที่เคยปรากฏในต้นไม้ก่อนหน้าแล้ว คือ [ราคาสินค้า < 3,000 ? = No] [3,000 <= ราคาสินค้า < 10,000 ? = No] [3,000 <= ราคาสินค้า < 10,000 ? = Yes] [ดอกเบี้ย 0% ผ่อนชำระ 6 งวด? = No] และ [ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด? = No] ดังนั้นการระบุกรณีทดสอบจะพิจารณาต่อจากเส้นที่เคยระบุกรณีทดสอบด้วย ในที่นี้พิจารณาเส้นที่ 6 ซึ่งได้มาจากเงื่อนไข [ราคาสินค้า < 3,000 ? = No] [3,000 <= ราคาสินค้า < 10,000 ? = Yes] [ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด? = No] และ [ดอกเบี้ย 0.99% ผ่อนชำระ 9 งวด? = No] ดังนั้นจึงทำสัญลักษณ์ต่อในเส้นที่ 6 โดยเพิ่มสัญลักษณ์บนเส้นที่ลากจากโหนด Yes ของต้นไม้การจำแนก ***ดอกเบี้ย 1.33% ผ่อนชำระ 12 งวด?*** และในเส้นที่ 8 ทำสัญลักษณ์เหมือนกับเส้นที่ 6 แต่เปลี่ยนจากโหนด Yes ของต้นไม้การจำแนก ***ดอกเบี้ย 1.33% ผ่อนชำระ 12 งวด?*** เป็นโหนด No

2) พิจารณาเส้นที่ 7 ซึ่งได้มาจากเงื่อนไข [ราคาสินค้า < 3,000 ? = No] [3,000 <= ราคาสินค้า < 10,000 ? = No] [ดอกเบี้ย 0% ผ่อนชำระ 6 งวด? = No] [ดอกเบี้ย 0.89% ผ่อนชำระ 6 งวด? = No] และ [ดอกเบี้ย 0.99% ผ่อนชำระ 9 งวด? = No] ดังนั้นจึงทำสัญลักษณ์ต่อในเส้นที่ 7 โดยเพิ่มสัญลักษณ์บนเส้นที่ลากจากโหนด Yes ของต้นไม้การจำแนก ***ดอกเบี้ย 1.33% ผ่อนชำระ 12 งวด?*** และในเส้นที่ 9 ทำสัญลักษณ์เหมือนกับเส้นที่ 7 แต่เปลี่ยนจากโหนด Yes ของต้นไม้การจำแนก ***ดอกเบี้ย 1.33% ผ่อนชำระ 12 งวด?*** เป็นโหนด No การระบุกรณีทดสอบจากการพิจารณาเงื่อนไขของต้นไม้การจำแนก ***ดอกเบี้ย 1.33% ผ่อนชำระ 12 งวด?*** สามารถแสดงได้ดังภาพประกอบที่ 3.18



ภาพประกอบที่ 3.18 การระบุกรณีทดสอบจากการพิจารณาเงื่อนไขของต้นไม้การจำแนก ***ดอกเบี้ย 1.33% ผ่อนชำระ 12 งวด?***

พิจารณาต้นไม้การจำแนก *ราคา < 3,000 ?* พบว่ามีโหนด Yes ไม่มีการทำสัญลักษณ์ปรากฏอยู่เลยจึงทำการระบุเป็นกรณีทดสอบ 1 กรณี โดยเส้นที่ 10 ทำสัญลักษณ์บนเส้นที่ลากจากโหนด Yes ของต้นไม้การจำแนก *ราคา < 3,000* เพียงอันเดียวเท่านั้น ซึ่งการระบุกรณีทดสอบจากการพิจารณาเงื่อนไขของต้นไม้การจำแนกทั้งหมด สามารถแสดงได้ดังภาพประกอบที่ 3.19



ภาพประกอบที่ 3.19 กรณีทดสอบจากการพิจารณาเงื่อนไขของต้นไม้การจำแนกที่ได้จากแผนภาพกิจกรรมของโปรแกรมคำนวณจำนวนเงินที่ต้องผ่อนชำระต่องวด

ดังนั้นกรณีทดสอบที่ได้จากแผนภาพกิจกรรมของโปรแกรมคำนวณค่าจอดรถ มีทั้งหมด 10 กรณีดังนี้

กรณีที่ 1 ราคา < 3,000 ? = No, 3000<=ราคา<10,000? = Yes, ดอกเบี้ย 0.89% ผ่อน 6 งวด ? = Yes

กรณีที่ 2 ราคา < 3,000 ? = No, 3000<=ราคา<10,000? = No, ดอกเบี้ย 0% ผ่อน 6 งวด ? = Yes

กรณีที่ 3 ราคา < 3,000 ? = No, 3000<=ราคา<10,000? = No, ดอกเบี้ย 0% ผ่อน 6 งวด ? = No, ดอกเบี้ย 0.89% ผ่อน 6 งวด ? = Yes

กรณีที่ 4 ราคา < 3,000 ? = No, 3000<=ราคา<10,000? = Yes, ดอกเบี้ย 0.89% ผ่อน 6 งวด ? = No, ดอกเบี้ย 0.99% ผ่อน 9 งวด ? = Yes

กรณีที่ 5 ราคา < 3,000 ? = No, 3000<=ราคา<10,000? = No, ดอกเบี้ย 0% ผ่อน 6 งวด ? = No, ดอกเบี้ย 0.89% ผ่อน 6 งวด ? = No, ดอกเบี้ย 0.99% ผ่อน 9 งวด ? = Yes

กรณีที่ 6 ราคา < 3,000 ? = No, 3000<=ราคา<10,000? = Yes, ดอกเบี้ย 0.89% ผ่อน 6 งวด ? = No, ดอกเบี้ย 0.99% ผ่อน 9 งวด ? = No, ดอกเบี้ย 1.33% ผ่อน 12 งวด ? = Yes

กรณีที่ 7 ราคา < 3,000 ? = No, 3000<=ราคา<10,000? = No, ดอกเบี้ย 0% ผ่อน 6 งวด ? = No, ดอกเบี้ย 0.89% ผ่อน 6 งวด ? = No, ดอกเบี้ย 0.99% ผ่อน 9 งวด ? = No, ดอกเบี้ย 1.33% ผ่อน 12 งวด ? = Yes

กรณีที่ 8 ราคา < 3,000 ? = No, 3000<=ราคา<10,000? = Yes, ดอกเบี้ย 0.89% ผ่อน 6 งวด ? = No, ดอกเบี้ย 0.99% ผ่อน 9 งวด ? = No, ดอกเบี้ย 1.33% ผ่อน 12 งวด ? = No

กรณีที่ 9 ราคา < 3,000 ? = No, 3000<=ราคา<10,000? = No, ดอกเบี้ย 0% ผ่อน 6 งวด ? = No, ดอกเบี้ย 0.89% ผ่อน 6 งวด ? = No, ดอกเบี้ย 0.99% ผ่อน 9 งวด ? = No, ดอกเบี้ย 1.33% ผ่อน 12 งวด ? = No

กรณีที่ 10 ราคา < 3,000 ? = Yes

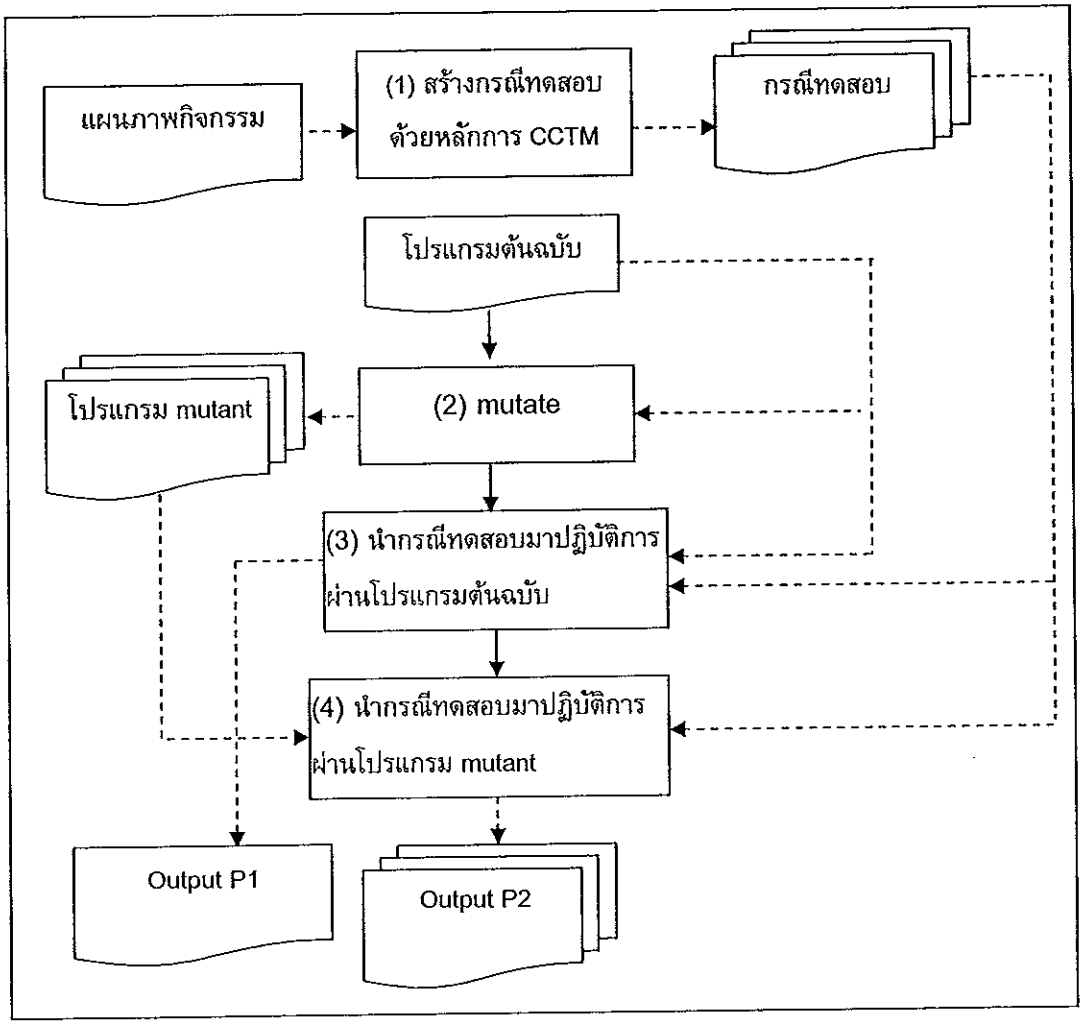
บทที่ 4

การประเมินประสิทธิภาพของกรณีทดสอบ

เนื้อหาในบทนี้จะกล่าวถึงขั้นตอนการประเมินประสิทธิภาพของกรณีทดสอบที่สร้างจากแผนภาพกิจกรรมด้วยหลักการ CCTM ซึ่งใช้วิธีการ mutation testing โดยเลือกใช้ตัวอย่างแผนภาพกิจกรรมของโปรแกรม 4 โปรแกรมมาใช้ในการประเมิน ซึ่งรายละเอียดของการประเมินประสิทธิภาพของกรณีทดสอบมีดังต่อไปนี้

4.1 ขั้นตอนการประเมินประสิทธิภาพของกรณีทดสอบ

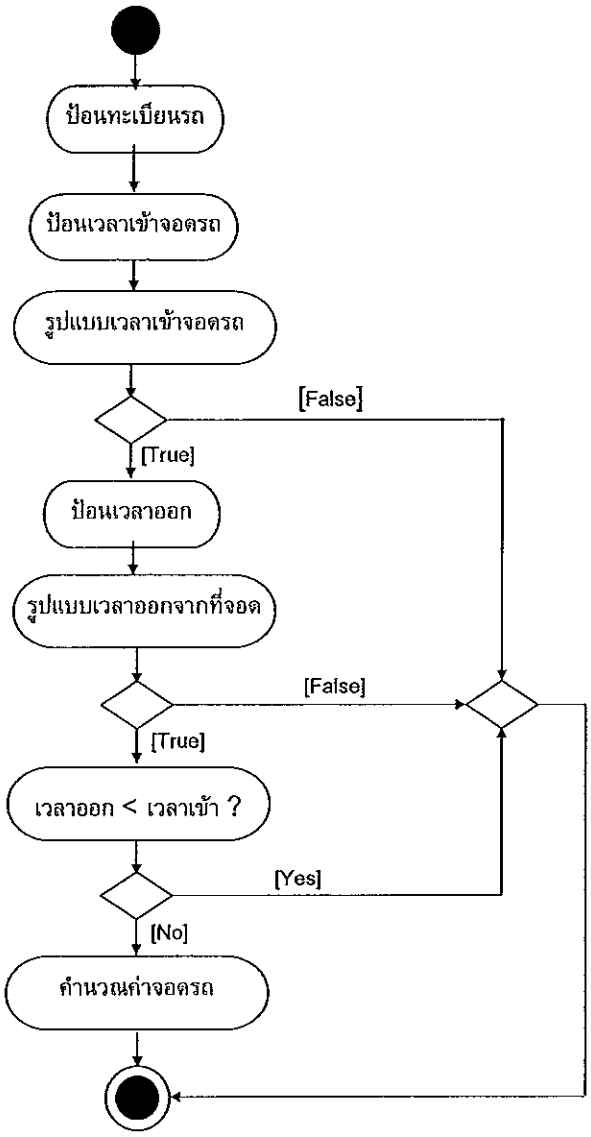
ขั้นตอนการประเมินประสิทธิภาพของกรณีทดสอบที่สร้างขึ้นจากแผนภาพกิจกรรมด้วยหลักการ CCTM ซึ่งในงานวิจัยนี้ใช้วิธี mutation testing มีขั้นตอนหลักทั้งหมด 4 ขั้นตอนโดยเริ่มจาก (1) ขั้นตอนการสร้างกรณีทดสอบ จากนั้นเป็นขั้นตอนที่ (2) กระบวนการ mutate เพื่อให้ได้โปรแกรม mutant ขั้นตอนที่ (3) จะนำกรณีทดสอบที่ได้มาปฏิบัติการผ่านโปรแกรมต้นฉบับและปฏิบัติการผ่านโปรแกรม mutant ขั้นตอนสุดท้ายขั้นตอนที่ (4) จะทำการเปรียบเทียบผลลัพธ์จากการปฏิบัติการที่ได้จากโปรแกรมต้นฉบับและโปรแกรม mutant ซึ่งขั้นตอนดังกล่าวสามารถแสดงได้ดังภาพประกอบที่ 4.1



ภาพประกอบที่ 4.1 ขั้นตอนการประเมินประสิทธิภาพของกรณีทดสอบที่สร้างขึ้น

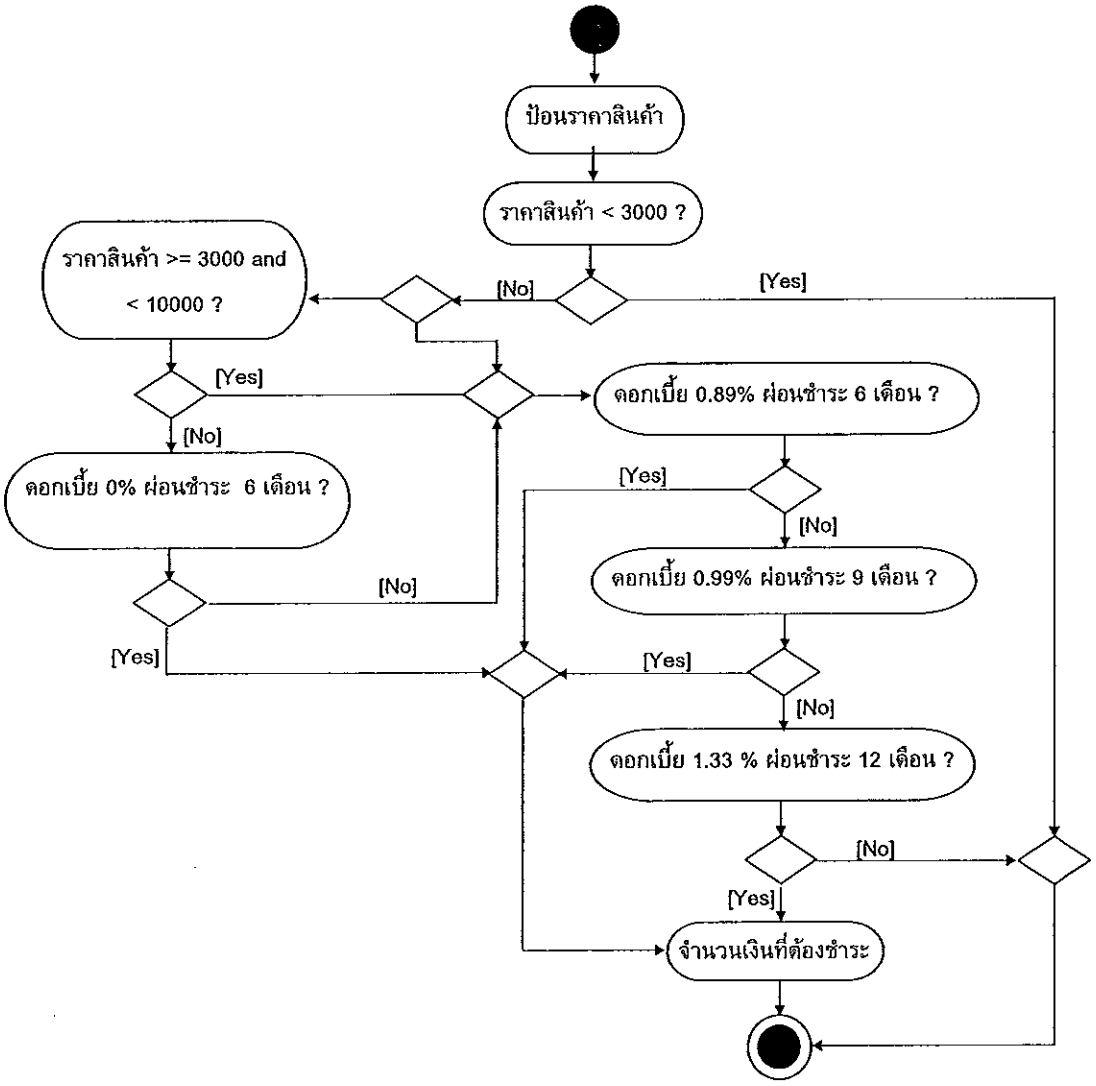
ขั้นตอนที่ 1 การสร้างกรณีทดสอบจากแผนภาพกิจกรรมด้วยหลักการ CCTM

เพื่อประเมินประสิทธิภาพของกรณีทดสอบที่สร้างจากแผนภาพกิจกรรมด้วยหลักการ CCTM ได้เลือกใช้แผนภาพกิจกรรมของโปรแกรมทั้งหมด 4 โปรแกรมได้แก่ 1) โปรแกรมคำนวณค่าจอดรถ 2) โปรแกรมคำนวณจำนวนเงินที่ต้องผ่อนชำระต่องวด 3) โปรแกรมการทำงานของตู้ ATM และ 4) โปรแกรมคำนวณหาราคาขายสินค้าสุทธิหลังหักส่วนลด แผนภาพกิจกรรมของโปรแกรมทั้ง 4 แสดงในภาพประกอบที่ 4.2 – ภาพประกอบที่ 4.5 ตารางที่ 4.1 แสดงจำนวนส่วนการตัดสินใจของแต่ละโปรแกรม เมื่อนำแผนภาพกิจกรรมของแต่ละโปรแกรมมาสร้างกรณีทดสอบโดยใช้หลักการ CCTM สามารถแสดงจำนวนของกรณีทดสอบที่ได้จากแต่ละแผนภาพกิจกรรมดังตารางที่ 4.2



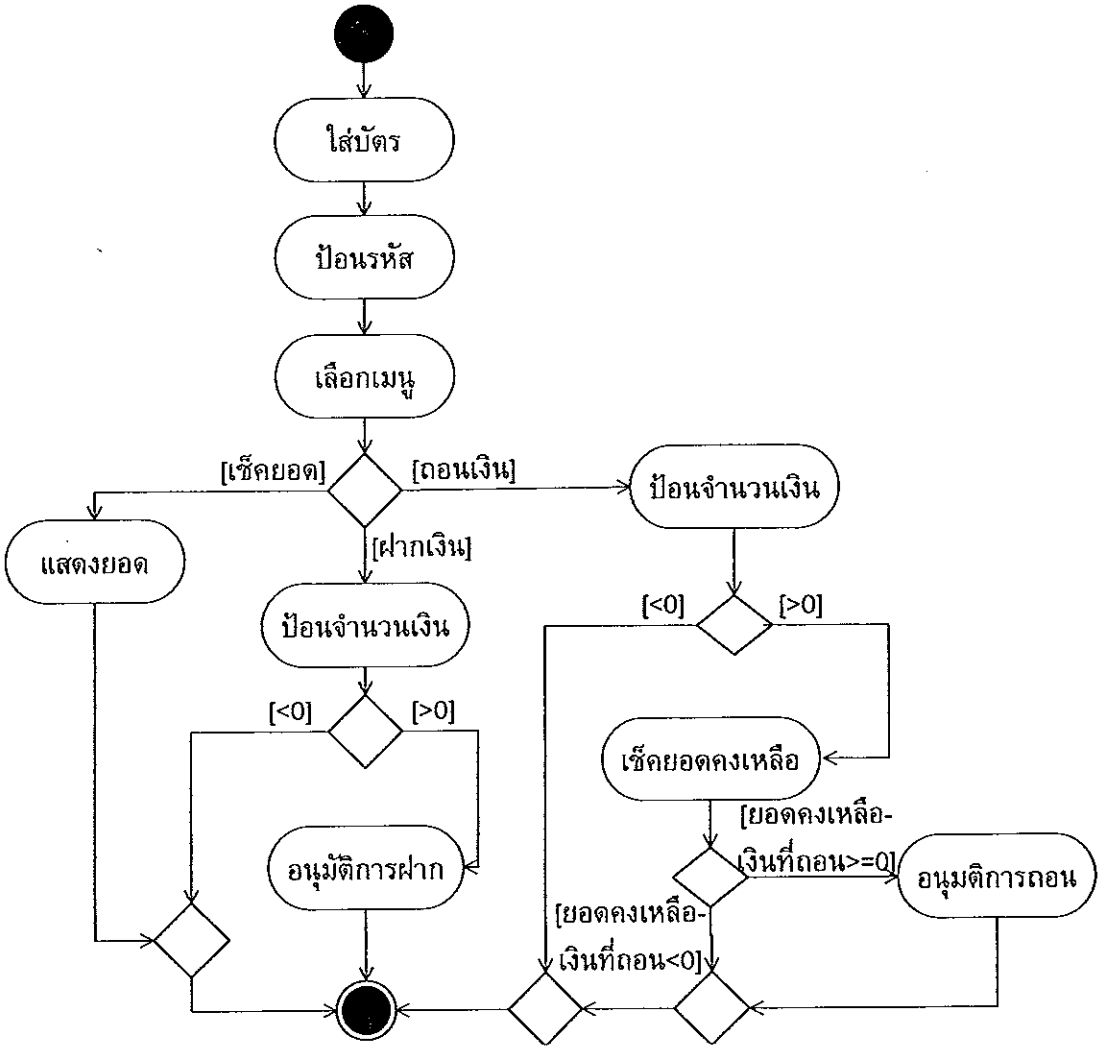
ภาพประกอบที่ 4.2 แผนภาพกิจกรรมของโปรแกรมคำนวณค่าจอดรถ

(พนิดา พานิชกุล, 2548)



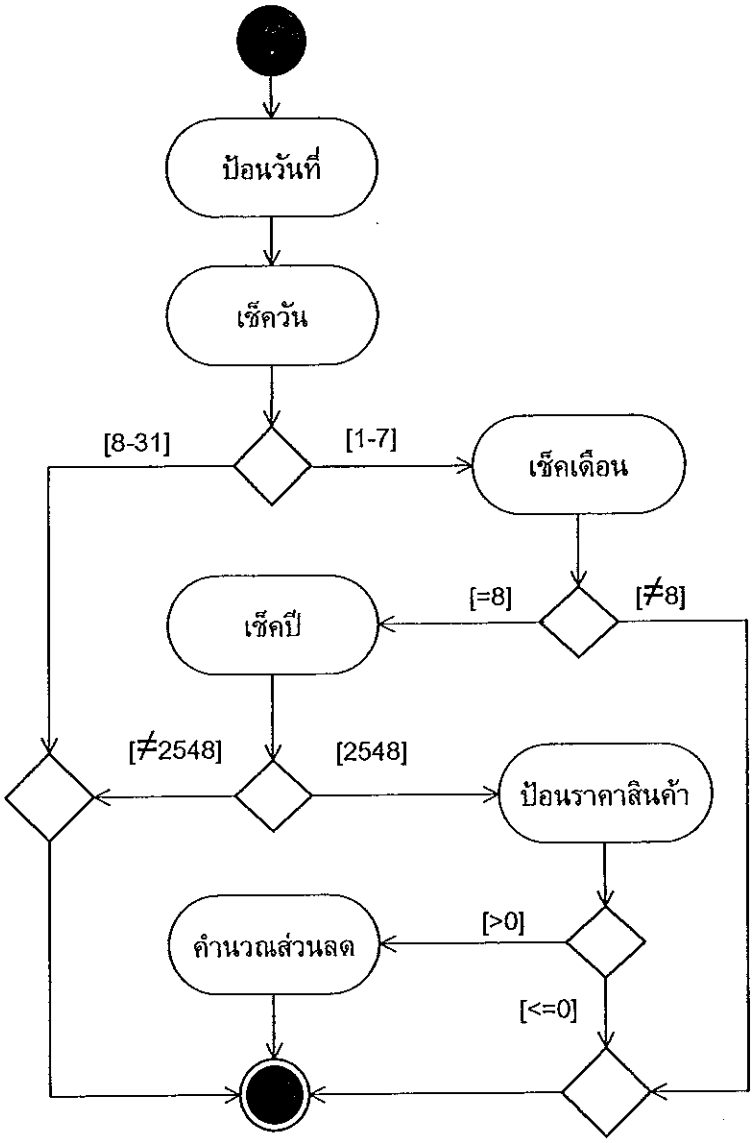
ภาพประกอบที่ 4.3 แผนภาพกิจกรรมของโปรแกรมคำนวณจำนวนเงินที่ต้องผ่อนชำระต่องวด

(พนิดา พานิชกุล, 2548)



ภาพประกอบที่ 4.4 แผนภาพกิจกรรมของโปรแกรมการทำงานของตู้ ATM

(พนิดา พานิชกุล, 2548)



ภาพประกอบที่ 4.5 แผนภาพกิจกรรมของโปรแกรมคำนวณหาราคาขายสินค้าสุทธิหลังหักส่วนลด (พนิดา พานิชกุล, 2548)

ตารางที่ 4.1 จำนวนส่วนการตัดสินใจที่ได้จากแผนภาพกิจกรรมของโปรแกรมตัวอย่าง

แผนภาพกิจกรรม	#decision
โปรแกรมคำนวณค่าจอดรถ	3
โปรแกรมคำนวณจำนวนเงินที่ต้องผ่อนชำระต่องวด	6
โปรแกรมการทำงานของตู้ ATM	3
โปรแกรมคำนวณหาราคาขายสินค้าสุทธิหลังหักส่วนลด	4

ตารางที่ 4.2 จำนวนกรณีทดสอบที่ได้จากแผนภาพกิจกรรมของโปรแกรมตัวอย่าง

แผนภาพกิจกรรม	#Test cases
โปรแกรมคำนวณค่าจอดรถ	4
โปรแกรมคำนวณจำนวนเงินที่ต้องผ่อนชำระต่องวด	10
โปรแกรมการทำงานของตู้ ATM	6
โปรแกรมคำนวณหาราคาขายสินค้าสุทธิหลังหักส่วนลด	5

ขั้นตอนที่ 2 การสร้างโปรแกรม mutant จากโปรแกรมต้นฉบับ

ขั้นตอนการสร้างโปรแกรม mutant จะดำเนินการโดยนำโปรแกรมต้นฉบับมาปรับเปลี่ยนโดยใช้ mutation operator ในงานวิจัยนี้จะใช้ operator สำหรับภาษาจาวาซึ่งได้ถูกนำเสนอในบทความเรื่อง Description of Method – level Mutation Operators for Java (Yu – Seung Ma, *et al.*, 2005) ในบทความนั้นได้กล่าวถึง operator ที่ใช้สำหรับภาษาจาวาในระดับ method ซึ่งมีทั้งหมด 12 operator ดังแสดงในตารางที่ 4.3

ตารางที่ 4.3 mutation operator สำหรับภาษาจาวา

Operator	Description
AOR	Arithmetic Operator Replacement
AOI	Arithmetic Operator Insertion
AOD	Arithmetic Operator Deletion
ROR	Relational Operator Replacement
COR	Conditional Operator Replacement
COI	Conditional Operator Insertion
COD	Conditional Operator Deletion
SOR	Shift Operator Replacement
LOR	Logical Operator Replacement
LOI	Logical Operator Insertion
LOD	Logical Operator Deletion
ASR	Assignment Operator Replacement

จากโปรแกรมต้นฉบับทั้ง 4 โปรแกรมสามารถนำมาสร้างโปรแกรม mutant ได้ดังตารางที่ 4.4

ตารางที่ 4.4 จำนวนโปรแกรม mutant ที่ได้จากการ mutate โปรแกรมต้นฉบับทั้ง 4 โปรแกรม

Program	#Program mutant
โปรแกรมคำนวณค่าจอดรถ	77
โปรแกรมคำนวณจำนวนเงินที่ต้องผ่อนชำระต่องวด	55
โปรแกรมการทำงานของตู้ ATM	35
โปรแกรมคำนวณหาราคาขายสินค้าสุทธิหลังหักส่วนลด	49

แต่ละโปรแกรม mutant จะได้จากการปรับเปลี่ยนโปรแกรมต้นฉบับ 1 ตำแหน่ง โดยใช้ mutation operator ตัวอย่างการสร้างโปรแกรม mutant โดยใช้ mutation operator ที่แตกต่างกันแสดงได้ดังนี้

ตัวอย่างที่ 1 สร้างโปรแกรม mutant โดยใช้ ROR operator ซึ่งแทน relational operator != ด้วย relational operator > แสดงรายละเอียดได้ดังภาพประกอบที่ 4.6

โปรแกรมต้นฉบับ	โปรแกรม mutant

inputTimeIn = stdin.readLine();
if((inputTimeIn.length()) **!=** 4) {
System.out.println
("ป้อนเวลาเข้าไม่ถูกต้อง
กรุณาตรวจสอบตามรูปแบบที่กำหนด");
System.exit(0); }
	inputTimeIn = stdin.readLine();
	if((inputTimeIn.length()) **>** 4) {
	System.out.println
	("ป้อนเวลาเข้าไม่ถูกต้อง
	กรุณาตรวจสอบตามรูปแบบที่กำหนด");
	System.exit(0); }

ภาพประกอบที่ 4.6 ตัวอย่างการสร้างโปรแกรม mutant โดยใช้ ROR operator

ตัวอย่างที่ 2 สร้างโปรแกรม mutant โดยใช้ AOR operator ซึ่งแทน arithmetic operator % ด้วย arithmetic operator + แสดงรายละเอียดได้ดังภาพประกอบที่ 4.7

โปรแกรมต้นฉบับ	โปรแกรม mutant

public static int calMin(int time) {
int min1 = time **%** 100;
int min2 = (time-min1)/100;
int min = (min2*60) +min1;
return min;
} // Method "calMin"
	public static int calMin(int time) {
	int min1 = time **+** 100;
	int min2 = (time-min1)/100;
	int min = (min2*60) +min1;
	return min;
	} // Method "calMin"

ภาพประกอบที่ 4.7 ตัวอย่างการสร้างโปรแกรม mutant โดยใช้ AOR operator

ตัวอย่างที่ 3 สร้างโปรแกรม mutant โดยใช้ ROR operator ซึ่งแทน relational operator < ด้วย relational operator > แสดงรายละเอียดได้ดังภาพประกอบที่ 4.8

```
if (price < 3000) {
System.out.println
("สินค้าราคาต่ำกว่า 3,000 ไม่สามารถซื้อเป็นเงินผ่อนได้"); }
else {
System.out.println
("กด 1 ดอกเบี้ย 0% ผ่อนชำระ 6 งวด (เฉพาะสินค้าราคา 10,000 บาทขึ้นไป)");
```

โปรแกรมต้นฉบับ

```
if (price > 3000) {
System.out.println
("สินค้าราคาต่ำกว่า 3,000 ไม่สามารถซื้อเป็นเงินผ่อนได้"); }
else {
System.out.println
("กด 1 ดอกเบี้ย 0% ผ่อนชำระ 6 งวด (เฉพาะสินค้าราคา 10,000 บาทขึ้นไป)");
```

โปรแกรม mutant

ภาพประกอบที่ 4.8 ตัวอย่างการสร้างโปรแกรม mutant โดยใช้ ROR operator

ตัวอย่างที่ 4 สร้างโปรแกรม mutant โดยใช้ COR operator ซึ่งแทน conditional operator && ด้วย conditional operator || แสดงรายละเอียดได้ดังภาพประกอบที่ 4.9

โปรแกรมต้นฉบับ	โปรแกรม mutant

if (choice == 1) {
if (price >= 3000 **&&** price <10000) {
System.out.print(" ");
System.out.println
("รูปแบบที่ 1 เฉพาะราคาสินค้า
10,000 บาทขึ้นไปเท่านั้น"); }
else
calPayment(price,0,6); }
	if (choice == 1) {
	if (price >= 3000 **||** price <10000) {
	System.out.print(" ");
	System.out.println
	("รูปแบบที่ 1 เฉพาะราคาสินค้า
	10,000 บาทขึ้นไปเท่านั้น"); }
	else
	calPayment(price,0,6); }

ภาพประกอบที่ 4.9 ตัวอย่างการสร้างโปรแกรม mutant โดยใช้ COR operator

ตัวอย่างที่ 5 สร้างโปรแกรม mutant โดยใช้ AOR operator ซึ่งแทน arithmetic operator + ด้วย arithmetic operator - แสดงรายละเอียดได้ดังภาพประกอบที่ 4.10

โปรแกรมต้นฉบับ
amount = (price1 **+** ((price1 * (interestRate/100))*numOfPayment))/numOfPayment; System.out.println("สินค้าราคา " +formatNumber.format(price1) + " บาท");
โปรแกรม mutant
amount = (price1 **-** ((price1 * (interestRate/100))*numOfPayment))/numOfPayment; System.out.println("สินค้าราคา " +formatNumber.format(price1) + " บาท");

ภาพประกอบที่ 4.10 ตัวอย่างการสร้างโปรแกรม mutant โดยใช้ AOR operator

ขั้นตอนที่ 3 นำกรณีทดสอบมาปฏิบัติการผ่านโปรแกรมต้นฉบับและโปรแกรม mutant

เมื่อนำโปรแกรมต้นฉบับและโปรแกรม mutant มาปฏิบัติการโดยใช้กรณีทดสอบที่สร้างขึ้น ผลลัพธ์จากการประเมินประสิทธิภาพของกรณีทดสอบที่สร้างขึ้นโดยการทำ mutation testing แสดงดังตารางที่ 4.5

ตารางที่ 4.5 ผลลัพธ์จากการประเมินประสิทธิภาพของกรณีทดสอบโดยการทำ mutation testing

Program	Total test cases	Total mutant	Killed mutant	Mutation score
โปรแกรมคำนวณค่าจอดรถ	4	77	64	0.83
โปรแกรมคำนวณค่างวด	10	55	48	0.87
โปรแกรมการทำงานของตู้ ATM	6	35	35	1
โปรแกรมคำนวณหาราคาขายสินค้าสุทธิหลังหักส่วนลด	5	49	40	0.82

จากผลการประเมินประสิทธิภาพของกรณีทดสอบดังกล่าวพบว่ากรณีทดสอบที่สร้างจากแผนภาพกิจกรรมของโปรแกรมการทำงานของตู้ ATM ให้ผล Mutation score เท่ากับ 1 ซึ่งหมายความว่ามีประสิทธิภาพที่ดีมาก และ Mutation score ที่ได้จากโปรแกรมอื่นๆ อีก 3 โปรแกรม มีค่า 0.83, 0.87 และ 0.82 ซึ่งเป็นค่าที่ค่อนข้างสูงแสดงให้เห็นว่าประสิทธิภาพของกรณีทดสอบที่นำมาใช้นั้นมีประสิทธิภาพที่อยู่ในเกณฑ์ที่ดี ซึ่งหมายความว่าหลักการสร้างกรณีทดสอบ CCTM ที่ได้นำเสนอในงานวิจัยนี้มีประสิทธิภาพ

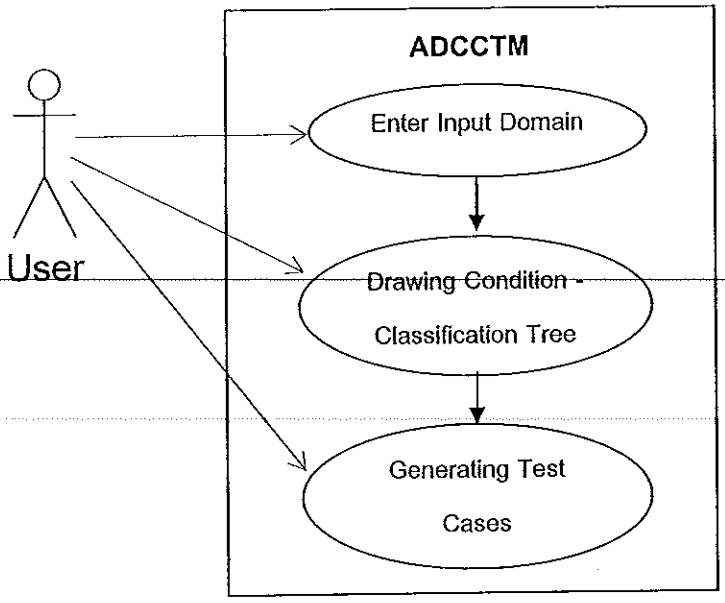
บทที่ 5

การออกแบบและพัฒนาต้นแบบเครื่องมือ ADCCTM

เนื้อหาในบทนี้กล่าวถึงการออกแบบต้นแบบเครื่องมือ Activity Diagram Condition - Classification Tree Model (ADCCTM) ซึ่งเป็นต้นแบบเครื่องมือที่พัฒนาขึ้นเพื่อสนับสนุนการสร้างกรณีทดสอบจากแผนภาพกิจกรรมตามหลักการ Condition - Classification Tree Model (CCTM) โดยใช้ภาษา Visual Basic ในการพัฒนา สำหรับเครื่องมือที่พัฒนามีองค์ประกอบหลักแบ่งเป็น 3 ส่วนได้แก่ ส่วนการป้อน input domain ส่วนการสร้างต้นไม้การจำแนกแบบมีเงื่อนไขและส่วนการสร้างกรณีทดสอบ

5.1 ความสัมพันธ์และกิจกรรมของระบบ

เครื่องมือ ADCCTM มีส่วนประกอบที่เกี่ยวข้องทั้งหมด 3 ส่วนได้แก่ ส่วนการป้อน input domain ส่วนการสร้างต้นไม้การจำแนกและส่วนการสร้างกรณีทดสอบ ดังแสดงภาพประกอบที่ 5.1 และมีรายละเอียดของแต่ละกิจกรรม (use case) ดังคำอธิบายในตารางที่ 5.1 – ตารางที่ 5.3



ภาพประกอบที่ 5.1 กิจกรรม (use case) และความสัมพันธ์ระหว่างกิจกรรมในระบบ ADCCTM

ตารางที่ 5.1 คำอธิบายกิจกรรม Enter Input Domain

ชื่อ	Enter Input Domain
เป้าหมาย	เพื่อให้ผู้ใช้ป้อน input domain ที่พิจารณาจากแผนภาพกิจกรรม
ผู้ดำเนินการ	ผู้ใช้งาน (user)
การดำเนินการ	ผู้ใช้งานป้อน input domain พร้อมทั้งเงื่อนไขของแต่ละ category เข้าสู่ระบบ
เงื่อนไขหรือข้อยกเว้น	-
ผลลัพธ์	categories และ options ของแผนภาพกิจกรรม

ตารางที่ 5.2 คำอธิบายกิจกรรม Drawing Condition-Classification Tree

ชื่อ	Drawing Condition-Classification Tree
เป้าหมาย	เพื่อให้ผู้ใช้สั่งงานให้ระบบวาดรูปต้นไม้การจำแนกแบบมีเงื่อนไข
ผู้ดำเนินการ	ผู้ใช้งาน (user)
การดำเนินการ	ผู้ใช้งานทำการเลือกสั่งงานให้ระบบวาดรูปต้นไม้การจำแนกแบบมีเงื่อนไขตาม input domain ที่ป้อนเข้าสู่ระบบ
เงื่อนไขหรือข้อยกเว้น	มีการป้อน input domain เข้าสู่ระบบ
ผลลัพธ์	ต้นไม้การจำแนกแบบมีเงื่อนไข

ตารางที่ 5.3 คำอธิบายกิจกรรม Generating Test Cases

ชื่อ	Generating Test Cases
เป้าหมาย	เพื่อให้ผู้ใช้สั่งงานให้ระบบสร้างกรณีทดสอบ
ผู้ดำเนินการ	ผู้ใช้งาน (user)
การดำเนินการ	ผู้ใช้งานทำการเลือกสั่งงานให้ระบบสร้างกรณีทดสอบจากต้นไม้การจำแนกแบบมีเงื่อนไขที่สร้างขึ้น
เงื่อนไขหรือข้อยกเว้น	มีการสั่งงานเพื่อวาดต้นไม้การจำแนกแบบมีเงื่อนไขมาก่อน
ผลลัพธ์	กรณีทดสอบ

5.2 ขั้นตอนการดำเนินงานของระบบ

จากการพิจารณากิจกรรมและความสัมพันธ์ระหว่างกิจกรรมในภาพประกอบที่ 5.1 สามารถอธิบายกระบวนการดำเนินงานหลักในระบบซึ่งแบ่งเป็น 3 ส่วนได้แก่ ส่วนการป้อน input domain ส่วนการสร้างต้นไม้การจำแนกและส่วนการสร้างกรณีทดสอบ ซึ่งสามารถอธิบายได้ดังนี้

5.2.1 ดำเนินการป้อน input domain ที่พิจารณามาจากแผนภาพกิจกรรมเข้าสู่ระบบ ซึ่ง input domain ที่ป้อนประกอบด้วย ข้อมูลของ category ซึ่งประกอบด้วยชื่อของ category แต่ละตัว และในแต่ละ category จะประกอบด้วย option และ condition ของ category นั้นๆ ซึ่งข้อมูลต่างๆ ที่ป้อนจะถูกจัดเก็บในโครงสร้างอาเรย์ดังนี้

```
category[1..m]{
    catname
    option[1..n]
    condition[1..k]
    }
```

5.2.2 หลังจากป้อน input domain ทั้งหมดเข้าสู่ระบบแล้ว ระบบจะนำข้อมูลที่ป้อนไปวาดต้นไม้การจำแนกแบบมีเงื่อนไข โดยลักษณะของต้นไม้จะแตกต่างกันขึ้นอยู่กับจำนวนของ category, option และ condition

5.2.3 ขั้นตอนการสร้างกรณีทดสอบ จะเริ่มจากการพิจารณาเงื่อนไขของ category

1) กรณีที่ไม่มีเงื่อนไขใน category หรือกรณีทดสอบยังไม่ถูกสร้างขึ้นให้เริ่มกำหนดกรณีทดสอบโดยกำหนดให้แต่ละกรณีทดสอบเท่ากับแต่ละ option ใน category นั้น

2) กรณีที่ใน category มีเงื่อนไขหรือมีกรณีทดสอบถูกสร้างขึ้นมาแล้วก่อนหน้า ให้ตรวจสอบเงื่อนไขใน category ว่าตรงกับเงื่อนไขในแต่ละ test cases หรือไม่

2.1) กรณีที่เงื่อนไขใน category ตรงกับเงื่อนไขใน test case ใดให้กำหนดกรณีทดสอบต่อใน test case นั้น

2.2) กรณีที่เงื่อนไขใน category ไม่ตรงกับเงื่อนไขใน test case ให้กำหนดกรณีทดสอบใหม่โดยให้เท่ากับ option ใน category และ option ในเงื่อนไขของ test case

รายละเอียดของขั้นตอนวิธีการสร้างกรณีทดสอบแสดงได้ดังภาพประกอบที่ 5.2

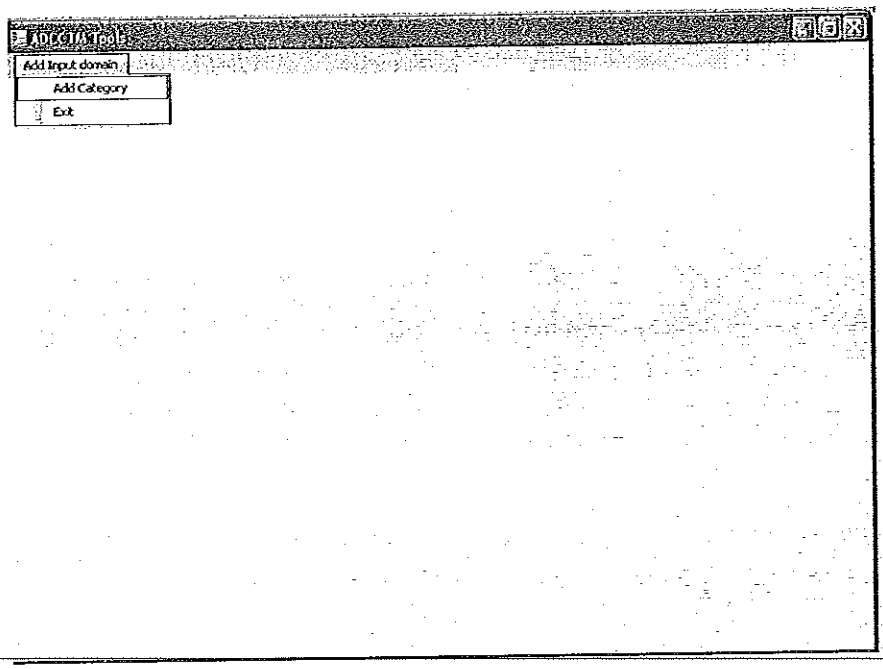
```
Input :
category[1...m]{
        catname// (A)
        option[1..n] // (a,aa)
        condition[1..k] // (A=a)  }
Generate test cases(category)
testcase[1...x]{
        position[1...y] // y is number of position in current test case
        condition[1...z] // z is number of condition in current test case  }
v=1, h=1, x=0, y=0, z=0
for i=1 to m // number of category
    for j=1 to n // number of option in category[i]
        if testcase is empty or category[i].condition is empty
            testcase[v].position[y=1+number of position in test case[v] ] = [v,j]
            testcase[v].condition[z=1+ number of condition in test case[v] ] = category[i].catname + "=" +  category[i].option[j]
            v++, h++, x++
        else
            for a=1 to x // total testcase
                for b=1 to z // number of condition in testcase[a]
                    if category[i].condition[b] = testcase[a].condition[b]
                        flag = true
                    else
                        flag = false
                if flag = true
                    testcase[a].position[y=1+number of position in test case[v] ] = [a,h]
                    testcase[a].condition[z=1+ number of condition in test case[v] ] = category[i].catname + "=" + category[i].option[j]
                    h++
                    exit loop for a
            if flag = false
                testcase[v].position[y=1+number of position in test case[v] ] = [v,h]
                testcase[v].condition[z=1+ number of condition in test case[v] ] = category[i].catname + "=" + category[i].option[j]
                length = 1
                for c=1 to k // number of condition in category[i]
                    for e=1 to i-1
                        for f=1 to n //  number of option in category[i]
                            pos= category[i].condition[c].find("=") // position of "=" in category[i].condition[c]
                            str= category[i].condition[c].substr(pos) // get text from "=" to end
                                if str = category[e].option[f]
                                    testcase[v].position[y=1+number of position in test case[v] ] = [v,length]
                                    testcase[v].condition[z=1+ number of condition in test case[v] ] = category[e].name + "=" + category[e].option[f]
                                    exit loop for e
                                else
                                    length++
                v++, h++, x++
```

ภาพประกอบที่ 5.2 ขั้นตอนการสร้างกรณีทดสอบด้วยหลักการ CCTM

5.3 การออกแบบส่วนติดต่อกับผู้ใช้

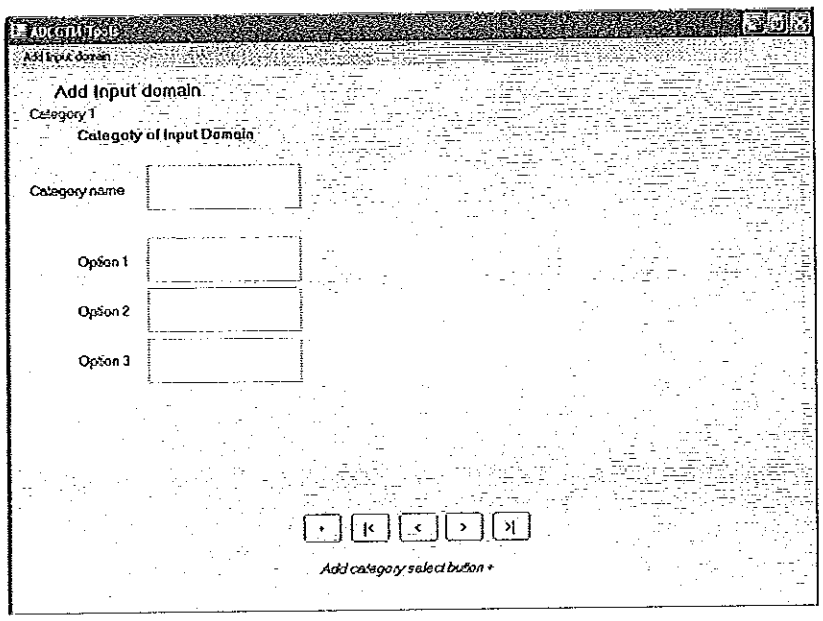
เครื่องมือ ADCCTM มีส่วนติดต่อผู้ใช้งานซึ่งออกแบบตามส่วนประกอบหลักของระบบทั้งหมด 3 ส่วน

ส่วนที่ 1 เป็นการรับ input domain ซึ่งหน้าจอการเริ่มต้นโปรแกรมจะแสดงแถบเมนู Add Category และเมนู Draw Classification Tree สำหรับการเริ่มต้นป้อน input domain เมื่อต้องการเริ่มป้อน input domain ให้เลือกเมนู Add Category หรือถ้าต้องการออกจากโปรแกรมให้เลือกเมนู Exit สำหรับเมนู Draw Classification Tree จะไม่สามารถทำงานได้จนกว่าจะมีการป้อน category ตั้งแต่ 2 ตัวขึ้นไป หน้าจอการเริ่มต้นโปรแกรมแสดงดังภาพประกอบที่ 5.3



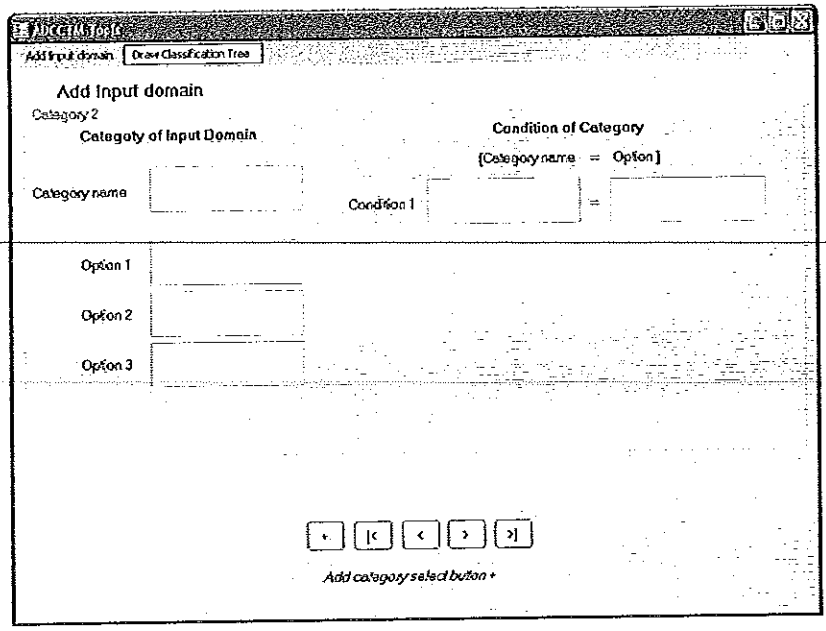
ภาพประกอบที่ 5.3 หน้าจอเริ่มต้นของโปรแกรม ADCCTM

เมื่อเข้าสู่หน้าจอการป้อน category ตัวที่ 1 ให้ผู้ใช้ป้อนข้อมูลของ category ตามช่องที่กำหนดซึ่งประกอบด้วย category name และ option ในแต่ละ category ส่วนด้านล่างของโปรแกรมจะแสดงแถบเครื่องมือในการเลื่อนแสดง category ในลำดับต่างๆ ที่ป้อน เมื่อต้องการเพิ่ม category ตัวถัดไปให้เลือกเครื่องมือ + ที่อยู่ด้านล่าง ดังภาพประกอบที่ 5.4



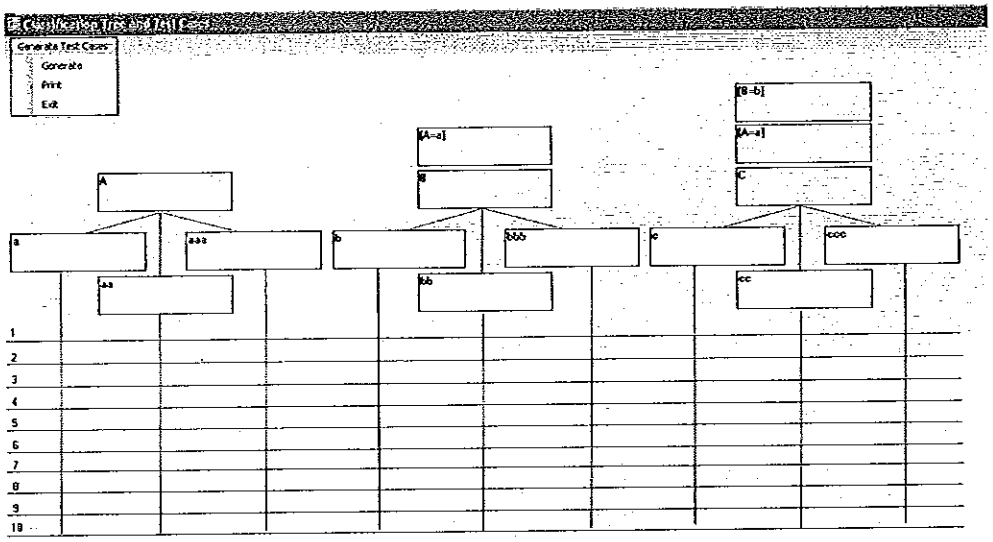
ภาพประกอบที่ 5.4 หน้าจอการป้อน input domain ของโปรแกรม ADCCTM

จากภาพประกอบที่ 5.4 เมื่อเลือกแถบเครื่องมือ + แล้วจะแสดงหน้าจอการป้อน category ตัวที่ 2 พร้อมด้วยช่องสำหรับป้อน condition อยู่ทางด้านขวามือ และสามารถใช้งานเมนู Draw Classification Tree ได้ในหน้าจอนี้ซึ่งแสดงดังภาพประกอบที่ 5.5



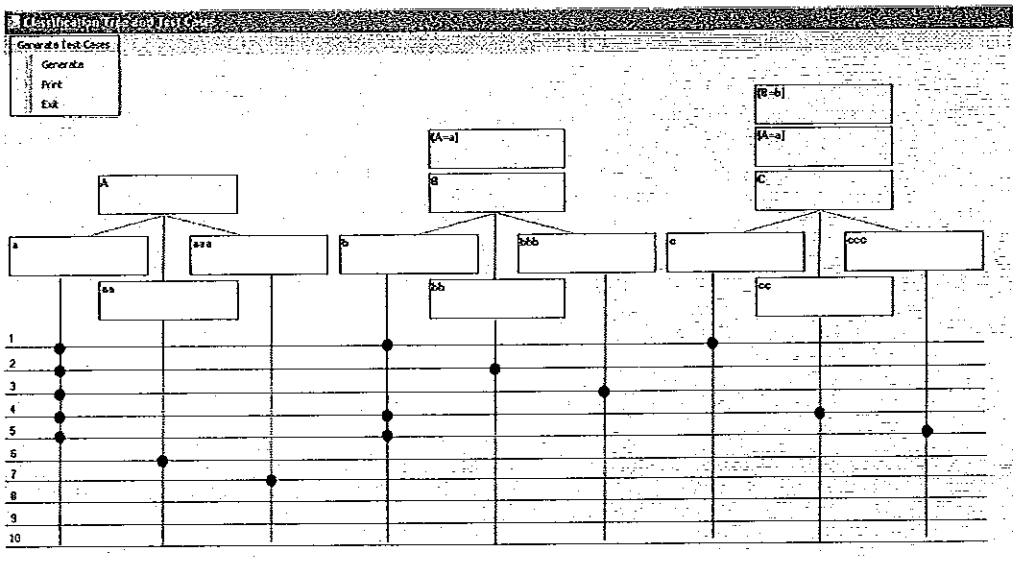
ภาพประกอบที่ 5.5 หน้าจอการป้อน category ตัวที่ 2 เข้าสู่โปรแกรม ADCCTM

เมื่อทำการป้อน input domain จนครบแล้วให้ผู้ใช้เลือกเมนู Draw Classification Tree เพื่อทำการสร้างต้นไม้การจำแนกแบบมีเงื่อนไข ซึ่งแสดงตัวอย่างต้นไม้การจำแนกแบบมีเงื่อนไขที่สร้างจาก input domain ที่ป้อน ได้ดังภาพประกอบที่ 5.6



ภาพประกอบที่ 5.6 ตัวอย่างต้นไม้การจำแนกแบบมีเงื่อนไข (condition-classification tree) ที่สร้างด้วยเครื่องมือ ADCCTM

จากภาพประกอบที่ 5.6 ผู้ใช้สามารถเลือกเมนู Generate เพื่อเริ่มสร้างกรณีทดสอบหรือเลือกเมนู Print เพื่อสั่งพิมพ์ต้นไม้การจำแนกแบบมีเงื่อนไข ซึ่งเมื่อเลือกเมนู Generate กรณีทดสอบจะถูกสร้างขึ้นภายในตารางด้านล่างต้นไม้การจำแนกแบบมีเงื่อนไขดังภาพประกอบที่ 5.7

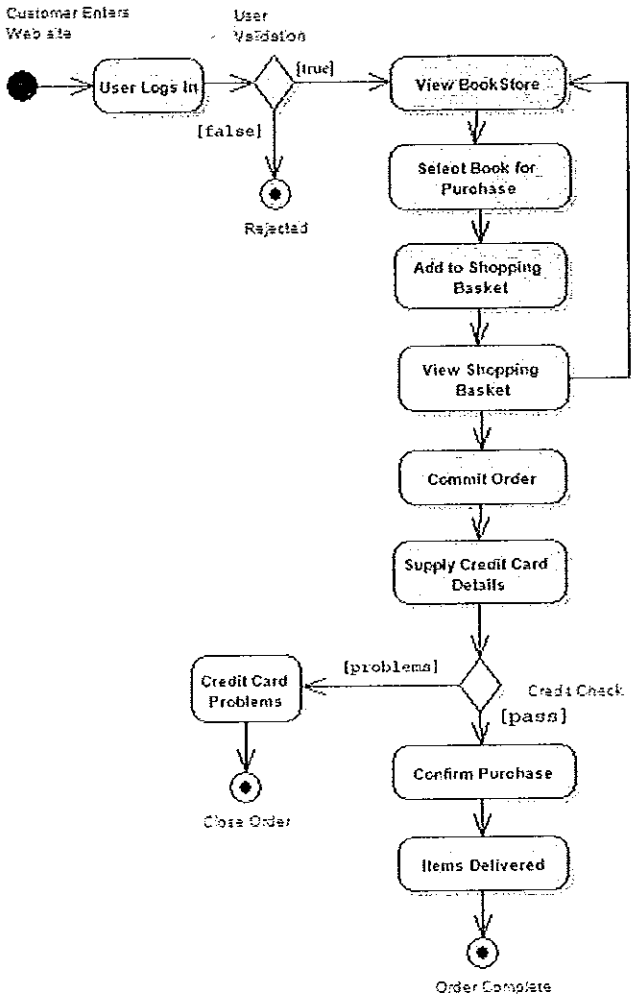


ภาพประกอบที่ 5.7 หน้าจอการสร้างกรณีทดสอบของโปรแกรม ADCCTM

5.4 การทดสอบประสิทธิภาพของต้นแบบเครื่องมือ ADCCTM

จากการใช้เครื่องมือ ADCCTM สร้างกรณีทดสอบจากแผนภาพกิจกรรม พบว่าเครื่องมือ ADCCTM สามารถสร้างกรณีทดสอบสำหรับแผนภาพทั้งหมดได้อย่างถูกต้อง ตัวอย่างการใช้เครื่องมือ ADCCTM ในการสร้างกรณีทดสอบจากแผนภาพกิจกรรมแสดงได้ดังต่อไปนี้

ตัวอย่างที่ 1 การสร้างกรณีทดสอบจากแผนภาพกิจกรรมร้านขายหนังสือ (Modern Analyst Community, 2006: Online) ซึ่งแสดงการทำงานในการสั่งซื้อหนังสือผ่านทางอินเตอร์เน็ตโดยเริ่มต้นจากการที่ลูกค้าเข้าสู่ระบบจากนั้นทำการเลือกหนังสือที่ต้องการซื้อสุดท้ายเป็นการชำระเงินและการจัดส่งให้แก่ลูกค้า รายละเอียดแสดงดังภาพประกอบที่ 5.8



ภาพประกอบที่ 5.8 แผนภาพกิจกรรมร้านขายหนังสือ

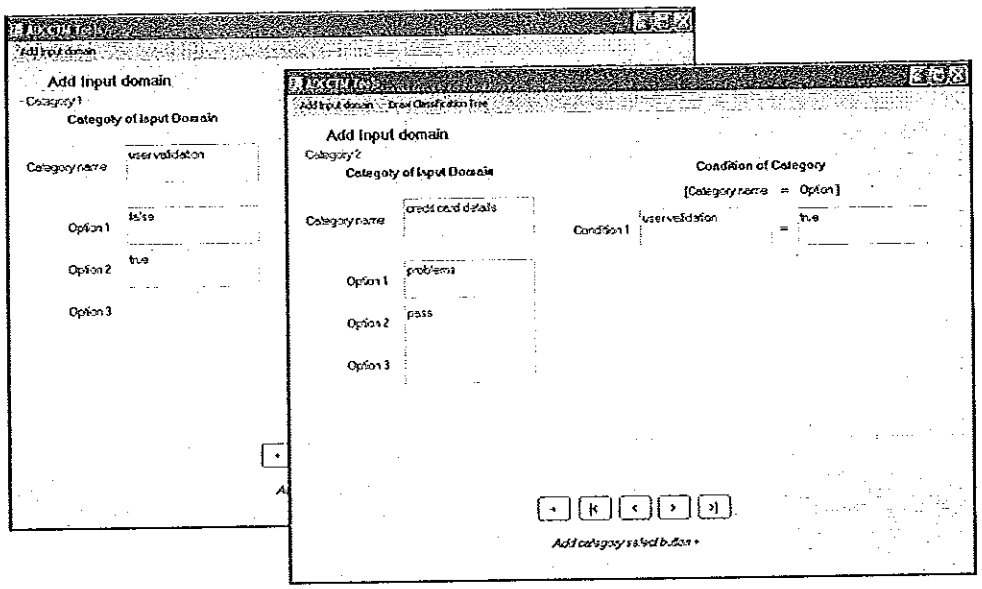
(Modern Analyst Community, 2006: Online)

จากแผนภาพกิจกรรมในภาพประกอบที่ 5.8 สามารถวิเคราะห์ input domain ออกเป็น category และ option ได้ดังตารางที่ 5.4

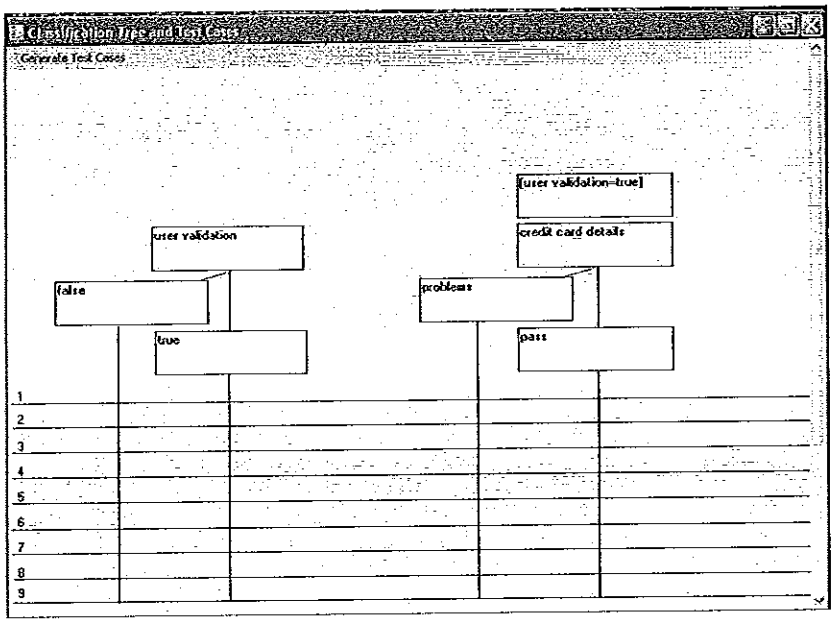
ตารางที่ 5.4 category และ option จากการวิเคราะห์แผนภาพกิจกรรมร้านขายหนังสือ

Category	Option	
1. user validation	false	true
2. credit card details	problems	pass

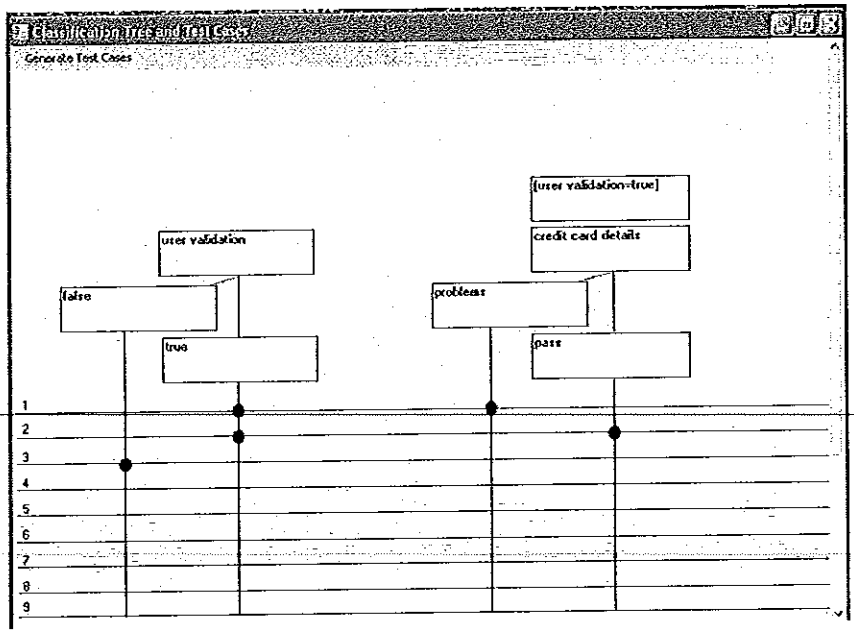
เมื่อพิจารณาแผนภาพกิจกรรมร้านขายหนังสือพบว่า ส่วนการตัดสินใจ credit card details ต้องผ่านเงื่อนไข [user validation= true] มาก่อน ดังนั้นจึงระบุเงื่อนไข user validation = true ลงในช่อง condition1 ใน category *credit card details* จากการพิจารณาจึงป้อนข้อมูลทั้งหมดเข้าสู่เครื่องมือ ADCCTM ดังภาพประกอบที่ 5.9 เมื่อเครื่องมือได้รับค่า category, option และ condition แล้วจะสร้างต้นไม้การจำแนกแบบมีเงื่อนไขและกรณีทดสอบได้ดังภาพประกอบที่ 5.10 และภาพประกอบที่ 5.11



ภาพประกอบที่ 5.9 หน้าจอการป้อน category, option และ condition เข้าสู่เครื่องมือ ADCCTM

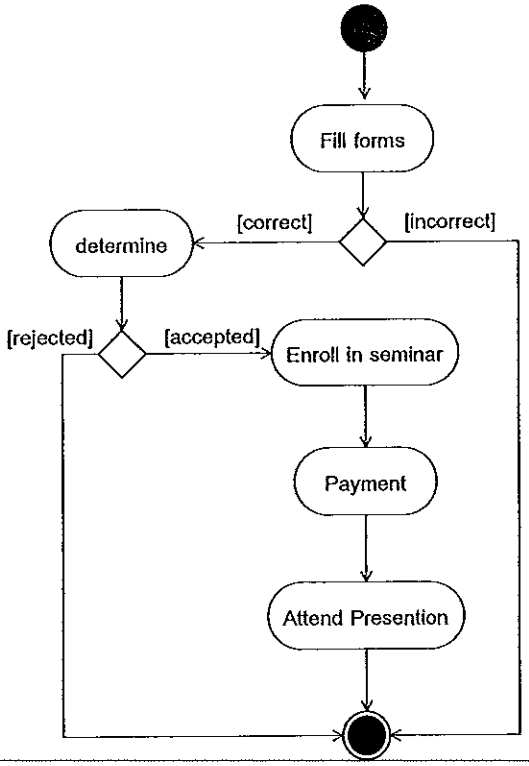


ภาพประกอบที่ 5.10 ต้นไม้การจำแนกแบบมีเงื่อนไขที่สร้างจากแผนภาพกิจกรรมร้านขายหนังสือ โดยใช้เครื่องมือ ADCCTM



ภาพประกอบที่ 5.11 กรณีทดสอบที่สร้างจากแผนภาพกิจกรรมร้านขายหนังสือ โดยใช้เครื่องมือ ADCCTM

ตัวอย่างที่ 2 การสร้างกรณีทดสอบจากแผนภาพกิจกรรมการส่งผลงานเข้าร่วมสัมมนา (Dong Xu, *et al.*, 2009) ซึ่งเริ่มต้นด้วยการให้ผู้สนใจกรอกข้อมูลลงในแบบฟอร์มการสมัครเข้าสู่ระบบ จากนั้นระบบจะทำการตรวจสอบความถูกต้องของแบบฟอร์ม หากแบบฟอร์มถูกต้องจะส่งผลงานเข้าสู่กระบวนการพิจารณาเพื่อเลือกรับผลงานหรือไม่หากผลงานได้รับเลือกจะเข้าสู่ขั้นตอนการลงทะเบียนและเข้าร่วมนำเสนอผลงานเป็นลำดับต่อไป ในกรณีที่แบบฟอร์มไม่ถูกต้องหรือไม่ผ่านการพิจารณาระบบจะตีกลับผลงานนั้น ซึ่งรายละเอียดแสดงดังภาพประกอบที่ 5.12



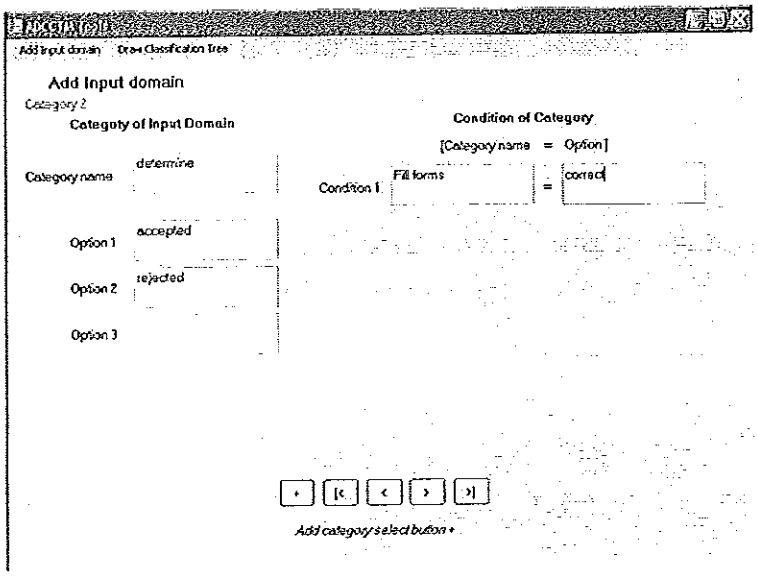
ภาพประกอบที่ 5.12 แผนภาพกิจกรรมการส่งผลงานเข้าร่วมสัมมนา (Dong Xu, *et al.*, 2009)

จากแผนภาพกิจกรรมในภาพประกอบที่ 5.12 สามารถวิเคราะห์ input domain ออกเป็น category และ option ได้ดังตารางที่ 5.5

ตารางที่ 5.5 category และ option จากการวิเคราะห์แผนภาพกิจกรรมการส่งผลงานเข้าร่วมสัมมนา

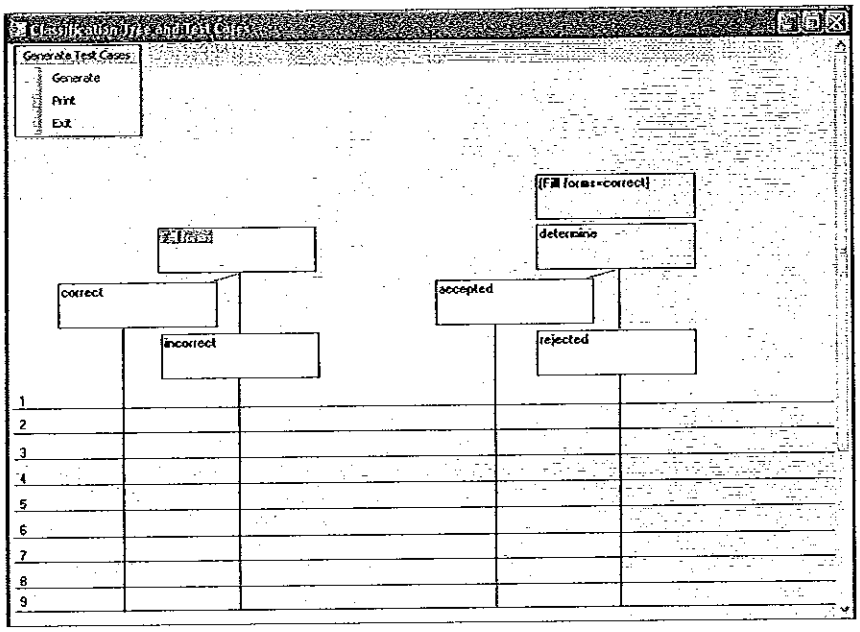
Category	Option	
1. fill forms	correct	incorrect
2. determine	rejected	accepted

เมื่อพิจารณาแผนภาพกิจกรรมการส่งผลงานเข้าร่วมสัมมนา พบว่าส่วนการตัดสินใจ determine ต้องผ่านเงื่อนไข [fill forms = correct] มาก่อน ดังนั้นจึงระบุเงื่อนไขนี้ให้กับ category *determine* ขั้นตอนต่อมาคือนำข้อมูลทั้งหมดป้อนเข้าสู่เครื่องมือ ADCCTM ดังภาพประกอบที่ 5.13

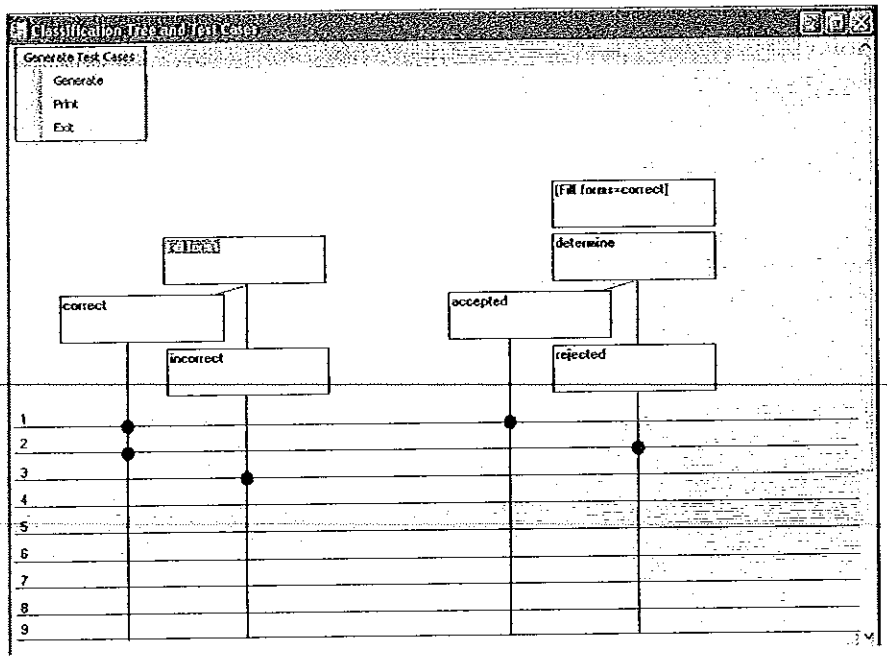


ภาพประกอบที่ 5.13 การป้อน category, option และ condition เข้าสู่เครื่องมือ ADCCTM

เมื่อโปรแกรมได้รับค่า category, option และ condition แล้วจะสร้างต้นไม้การจำแนกแบบมีเงื่อนไขดังภาพประกอบที่ 5.14 และสร้างกรณีทดสอบได้ดังภาพประกอบที่ 5.15

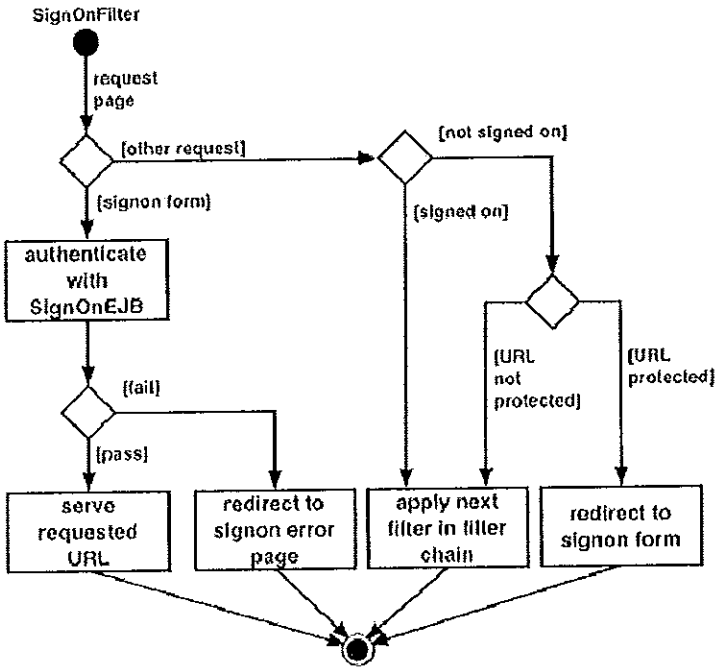


ภาพประกอบที่ 5.14 ต้นไม้การจำแนกแบบมีเงื่อนไขที่สร้างจากแผนภาพกิจกรรมการส่งผลงานเข้าร่วมสัมมนาโดยใช้เครื่องมือ ADCCTM



ภาพประกอบที่ 5.15 กรณีทดสอบที่สร้างจากแผนภาพกิจกรรมการส่งผลงานเข้าร่วมสัมมนาโดยใช้เครื่องมือ ADCCTM

ตัวอย่างที่ 3 การสร้างกรณีทดสอบจากแผนภาพกิจกรรมการตรวจสอบการใช้ระบบ (Beth Stearns, *et al.*, 2002) ซึ่งแสดงการทำงานของการตรวจสอบการใช้ระบบในการเข้าสู่ระบบหรือการเรียกใช้ส่วนอื่นๆของระบบ ซึ่งรายละเอียดแสดงดังภาพประกอบที่ 5.16



ภาพประกอบที่ 5.16 แผนภาพกิจกรรมการตรวจสอบการใช้ระบบ

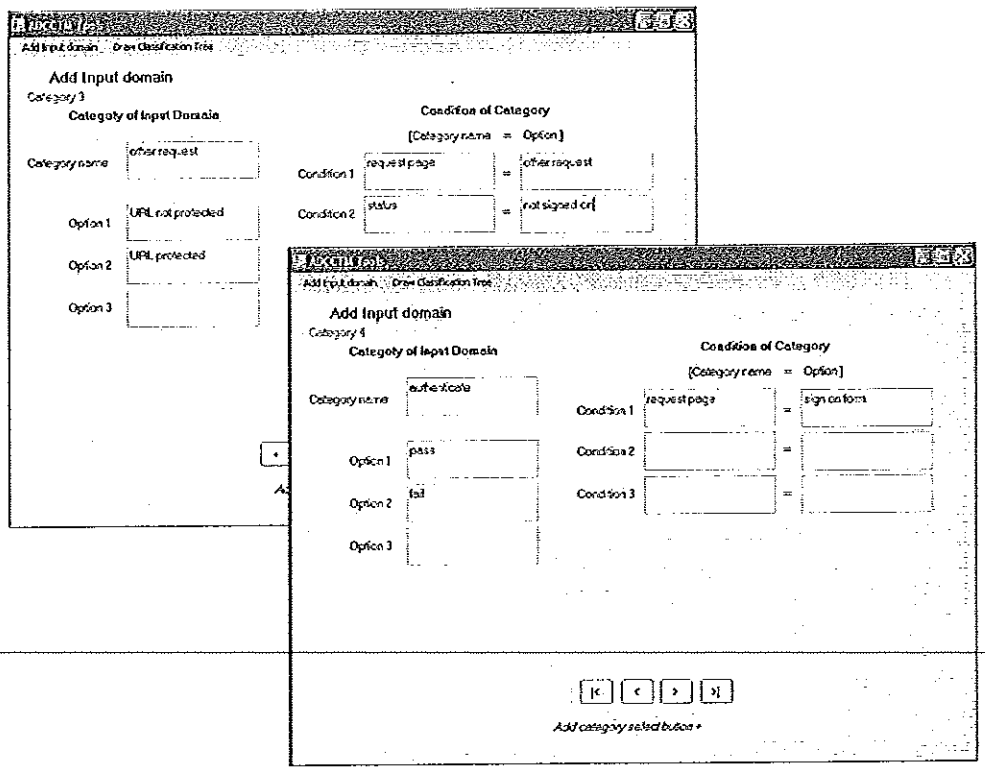
(Beth Stearns, *et al.*, 2002)

จากแผนภาพกิจกรรมในภาพประกอบที่ 5.16 สามารถวิเคราะห์ input domain ออกเป็น category และ option ได้ดังตารางที่ 5.6

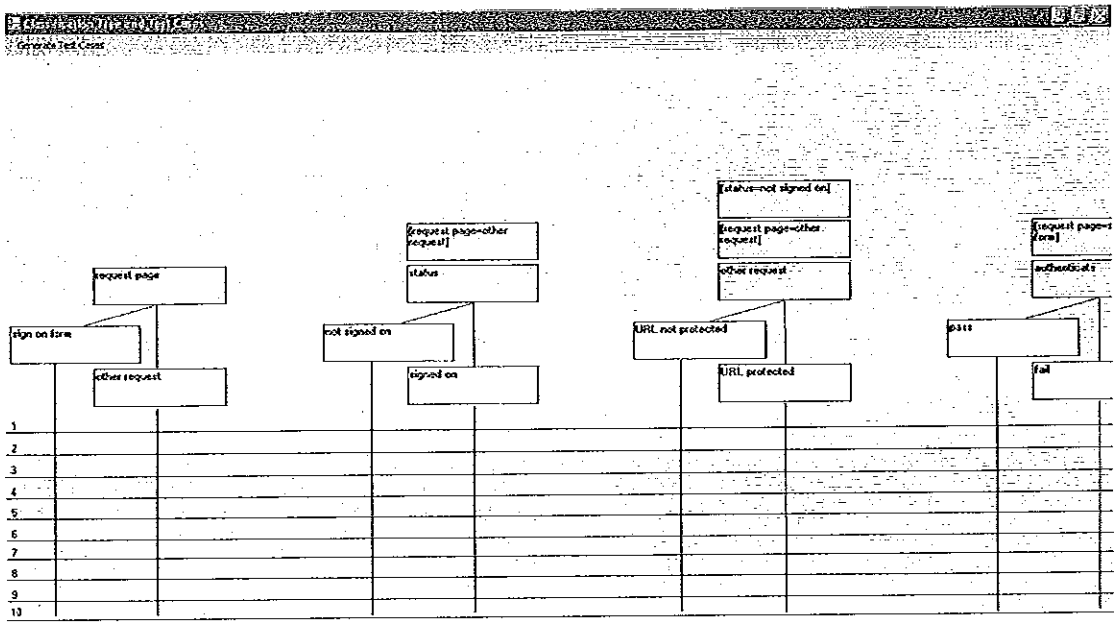
ตารางที่ 5.6 category และ option จากการวิเคราะห์แผนภาพกิจกรรมการตรวจสอบการใช้ระบบ

Category	Option	
1. request page	sign on form	other request
2. status	not signed on	signed on
3. other request	URL not protected	URL protected
4. authenticate	pass	fail

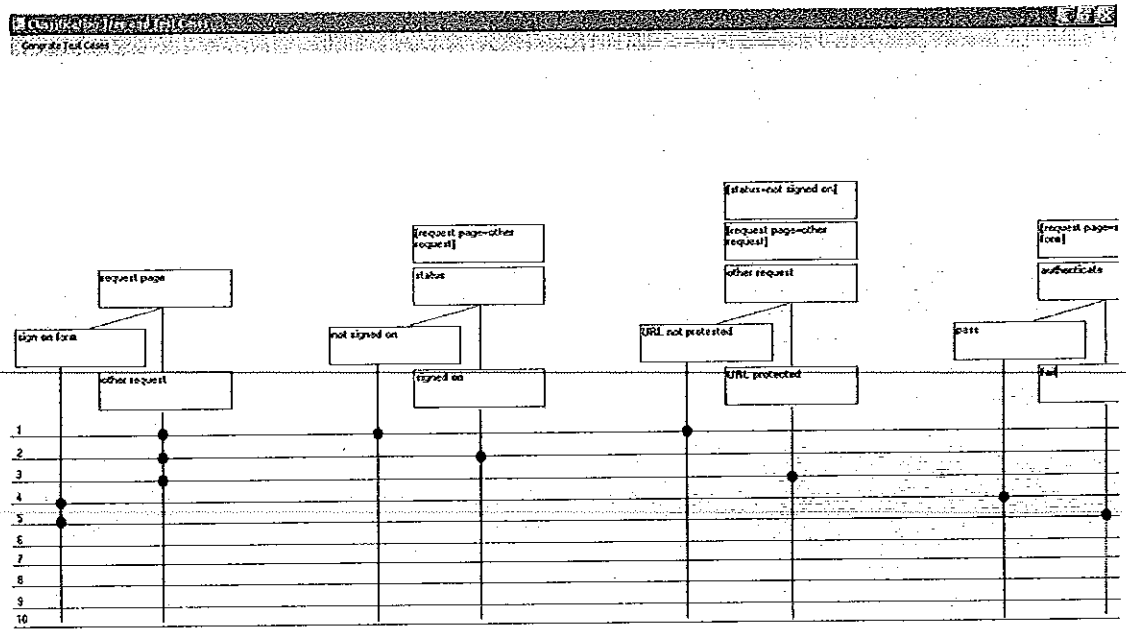
เมื่อพิจารณาแผนภาพกิจกรรมการตรวจสอบการใช้ระบบพบว่า ส่วนการตัดสินใจ status ต้องผ่านเงื่อนไข [request page = other request] มาก่อน ดังนั้นจึงระบุเงื่อนไขดังกล่าวให้กับ category *status* ส่วนการตัดสินใจ other request ต้องผ่านเงื่อนไข [request page = other request] และ [status = not signed on] มาก่อน ดังนั้นจึงระบุเงื่อนไขดังกล่าวให้กับ category *other request* และส่วนการตัดสินใจ authenticate ต้องผ่านเงื่อนไข [request page = signon form] มาก่อน ดังนั้นจึงระบุเงื่อนไขดังกล่าวให้กับ category *authenticate* ขั้นตอนต่อมาคือนำข้อมูลทั้งหมดป้อนเข้าสู่เครื่องมือ ADCCTM ดังภาพประกอบที่ 5.17 เมื่อโปรแกรมได้รับค่า category, option และ condition แล้วจะสร้างต้นไม้การจำแนกแบบมีเงื่อนไขและกรณีทดสอบได้ดังภาพประกอบที่ 5.18 และภาพประกอบที่ 5.19



ภาพประกอบที่ 5.17 การป้อน category, option และ condition เข้าสู่เครื่องมือ ADCCTM



ภาพประกอบที่ 5.18 ต้นไม้การจำแนกแบบมีเงื่อนไขที่สร้างจากแผนภาพกิจกรรมการตรวจสอบการใช้ระบบโดยใช้เครื่องมือ ADCCTM



ภาพประกอบที่ 5.19 กรณีทดสอบที่สร้างจากแผนภาพกิจกรรมการตรวจสอบการใช้ระบบโดยใช้เครื่องมือ ADCCTM

บทที่ 6

บทสรุปและข้อเสนอแนะ

จากเอกสารในบทที่ผ่านมา ได้กล่าวถึงความเป็นมาของงานวิจัยและศึกษาเอกสารรวมทั้งแนวคิดที่เกี่ยวข้อง และหลักการที่ได้นำเสนอ จนถึงการพัฒนาเครื่องมือต้นแบบรวมทั้งการทดสอบการใช้งานเครื่องมือ สำหรับในบทนี้จะกล่าวถึงผลสรุปในการทำวิจัย ปัญหาและอุปสรรคในการวิจัย รวมถึงข้อเสนอแนะสำหรับผู้ที่สนใจในอนาคต

6.1 บทสรุป

การทดสอบซอฟต์แวร์เป็นขั้นตอนที่มีค่าใช้จ่ายสูงโดยเฉพาะ การสร้างกรณีทดสอบ หากการทดสอบซอฟต์แวร์กระทำได้เร็วและมีจำนวนกรณีทดสอบที่ไม่มากจนเกินไปจะสามารถช่วยลดค่าใช้จ่ายในขั้นตอนการทดสอบซอฟต์แวร์ลงได้เป็นจำนวนมาก

วิทยานิพนธ์นี้ได้นำเสนอแบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไข (Condition - Classification Tree Model: CCTM) สำหรับสร้างกรณีทดสอบโดยใช้แผนภาพกิจกรรมของ UML แบบจำลองนี้ถูกสร้างจาก input domain ที่รวบรวมได้จากแผนภาพกิจกรรม ซึ่งประกอบด้วยชื่อของ category และ option ต่างๆใน category รวมทั้งเงื่อนไขในแต่ละ category จากนั้นนำ input domain มาสร้างต้นไม้การจำแนกแบบมีเงื่อนไขและใช้ต้นไม้ที่ได้นี้สร้างตารางกรณีทดสอบ ขั้นตอนสุดท้ายเป็นการสร้างกรณีทดสอบที่สอดคล้องกับเงื่อนไขต่างๆ โดยการพิจารณาเงื่อนไขในแต่ละ category

กรณีทดสอบจากวิธีการที่นำเสนอจะถูกประเมินประสิทธิภาพโดยใช้วิธี mutation testing ผลจากการประเมินพบว่า mutation score ที่ได้จากการทำ mutation testing มีค่าค่อนข้างสูง แสดงให้เห็นว่ากรณีทดสอบที่สร้างจากวิธีการที่นำเสนอเป็นกรณีทดสอบที่มีประสิทธิภาพสูง นอกจากนี้วิทยานิพนธ์นี้ได้พัฒนาต้นแบบเครื่องมือในการสร้างกรณีทดสอบเพื่อสนับสนุนแนวคิดที่นำเสนอชื่อเครื่องมือ ADCCTM (Activity Diagram Condition – Classification Tree Model) ซึ่งผลลัพธ์จากการใช้เครื่องมือในการสร้างกรณีทดสอบจากแผนภาพกิจกรรม พบว่ากรณีทดสอบที่ได้จากการใช้เครื่องมือ ADCCTM เป็นกรณีทดสอบที่มีประสิทธิภาพ

ผลลัพธ์ที่ได้จากวิธีการที่ได้นำเสนอนี้ช่วยให้ 1) การทดสอบเริ่มต้นได้ตั้งแต่ขั้นตอนการออกแบบโดยใช้ specification 2) ช่วยลดเวลาในการวิเคราะห์เอกสารความต้องการ เนื่องจากการวิเคราะห์ input domain จากแผนภาพกิจกรรมทำได้ง่ายกว่า 3) ช่วยลด

ความซับซ้อนในการสร้างกรณีทดสอบเนื่องจากมีการระบุเงื่อนไข ซึ่งจะทำให้กรณีทดสอบที่มีความเป็นไปได้ต่ำถูกกำจัดออก ทำให้ได้จำนวนกรณีทดสอบลดน้อยลง

6.2 ปัญหาและอุปสรรค

เนื่องจากวิทยานิพนธ์นี้ได้นำเสนอหลักการในการสร้างกรณีทดสอบโดยประยุกต์ใช้หลักการ CTM (Classification Tree Method) ดังนั้นจึงต้องมีการศึกษาเกี่ยวกับหลักการ CTM อย่างมาก แต่เนื่องจากหลักการ CTM ถูกนำเสนอมานานแล้วจึงทำให้การติดตามเอกสารทำได้ค่อนข้างยากและมีจำนวนน้อย

6.3 ข้อเสนอแนะและงานในอนาคต

6.3.1 หลักการที่นำเสนอเหมาะกับแผนภาพกิจกรรมที่ไม่มีการวนลูปที่ซับซ้อน เพราะมีการเก็บเงื่อนไขเพียงเงื่อนไขเดียวจากแต่ละ category แต่จะพบว่าในแผนภาพกิจกรรมอาจจะเกิดกรณีที่มีการวนลูปกลับไปยัง category ที่เคยผ่านมาได้ ในอนาคตควรมีการปรับปรุงหลักการให้สามารถใช้กับแผนภาพกิจกรรมที่มีการวนลูปที่ซับซ้อนได้

6.3.2 เครื่องมือที่นำเสนอต้องอาศัยผู้ทดสอบในการวิเคราะห์ input domain จากแผนภาพกิจกรรม ดังนั้นในอนาคตควรพัฒนาเครื่องมือสำหรับสร้างกรณีทดสอบจากแผนภาพกิจกรรมได้โดยอัตโนมัติ

บรรณานุกรม

พนิดา พานิชกุล. 2548. การเขียนโปรแกรมคอมพิวเตอร์เบื้องต้นด้วยภาษา Java. พิมพ์ครั้งที่ 3. เคทีพี: กรุงเทพฯ.

Beth Stearns, Inderjeet Singh, Mark Johnson, and The Enterprise Team. 2002. Designing Enterprise Applications with the Java™ 2, Enterprise Edition, Sun Microsystems, Inc. California.

Dong Xu, Huaikou Miao and Nduwimfura Philbert. 2009. Model Checking UML Activity Diagrams in FDR. 2009 Eigth IEEE/ACIS International Conference on Computer and Information Science. IEEE Computer Society, 2009. pp 1035 – 1040.

Grochtmann, M., Grimm, K. and Wegener, J.1993. 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, October 1993. pp 25 – 28.

Ilene Burnstein. 2003. Practical software testing : a process-oriented approach. Springer-Verlag. USA.

J.Glenford and Myers. 2004. The Art of Software Testing. Second Edition. John Wiley & Sons, Inc. USA.

Modern Analyst Community. 2006. UML Diagrams. http://www.modernanalyst.com (accessed 7/7/2010).

Object Management Group. 2003. Unified modeling language. Specification v1.5 formal/2003-03-01, Object Management Group, Mar 2003.

P.Ammann and J.Offutt. 2008. Introduction to Software Testing. Cambridge University Press. USA.

Randy Miller. 2003. Practical UML A Hands-On Introduction for Developers. http://gp.codegear.com/authors/edit/661.aspx (accessed 20/5/2010).

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

T. Y. Chen , Pak-Lok Poon , Sau-Fun Tang and T. H. Tse. 2005. 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.

W. Sinzhang, Y. Jiesong, Y. Xiaofeng, H. Jun, L. Xuandong, and Z. Guoliang. 2004. 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.

Yu – Seung Ma and J.Offutt. 2005. Description of Method – level Mutation Operators for Java. http://cs.gmu.edu/~offutt/mujava/mutopsMethod.pdf (accessed 11/5/2010).

ภาคผนวก

ภาคผนวก ก.

ผลงานตีพิมพ์

เรื่อง	การสร้างกรณีทดสอบจาก UML Activity Diagram แบบอัตโนมัติด้วยวิธีการต้นไม้การจำแนก
งานประชุมวิชาการ	การประชุมทางวิชาการระดับชาติด้านคอมพิวเตอร์และเทคโนโลยีสารสนเทศ (NCCIT 2010) ครั้งที่ 6
สถานที่	กรุงเทพมหานคร ประเทศไทย
วันที่	3 - 5 มิถุนายน 2553

NCCIT2010-37

การสร้างกรณีทดสอบจาก UML Activity Diagram แบบอัตโนมัติด้วยวิธีการต้นไม้การจำแนก

Automatic Test Case Generation for UML Activity Diagrams using Classification Tree Method (CTM)

ปรัชญานีย์ ไทยเกิด (Pratyanee Thaikerd)[1] และ สุภาภรณ์ กานต์สมเกียรติ (Supaporn Kansomkeat)[2]

[1,2]SEA Lab, ภาควิชาวิทยาการคอมพิวเตอร์, คณะวิทยาศาสตร์, มหาวิทยาลัยสงขลานครินทร์

s5110220041@psu.ac.th, supaporn.k@psu.ac.th

บทคัดย่อ

การทดสอบซอฟต์แวร์เป็นขั้นตอนที่สำคัญในการพัฒนาซอฟต์แวร์ เนื่องจากค่าใช้จ่ายส่วนใหญ่จะถูกใช้ในขั้นตอนนี้ การสร้างกรณีทดสอบโดยอัตโนมัติจะช่วยลดค่าใช้จ่ายในการทดสอบซอฟต์แวร์ ในปัจจุบันซอฟต์แวร์ส่วนใหญ่ถูกพัฒนาขึ้นโดยใช้เทคโนโลยีเชิงวัตถุ และมีภาษามากมายที่ถูกพัฒนาขึ้นเพื่อสนับสนุนการออกแบบซอฟต์แวร์เชิงวัตถุ ภาษาที่ได้รับความนิยมเป็นอย่างมากคือ ภาษาการออกแบบเชิงโมเดล (Unified Modeling Language: UML) แผนภาพกิจกรรม (Activity Diagram) เป็นแผนภาพประเภทหนึ่งของ UML ซึ่งแสดงพฤติกรรมของซอฟต์แวร์ บทความนี้เสนอวิธีการสร้างกรณีทดสอบจากแผนภาพกิจกรรมโดยใช้หลักการต้นไม้ การจำแนก (Classification Tree Method: CTM) ซึ่งวิธีที่เสนอทำให้การทดสอบซอฟต์แวร์กระทำได้ตั้งแต่ช่วงต้นของกระบวนการพัฒนาซอฟต์แวร์และช่วยลดเวลาในการวิเคราะห์เอกสารความต้องการของซอฟต์แวร์ได้อย่างมาก

คำสำคัญ: หลักการต้นไม้การจำแนก ภาษาการออกแบบเชิงโมเดล แผนภาพกิจกรรม

Abstract

Software testing plays a major role in software development because it accounts for a large part of development cost. Automatic test case generation can reduce the testing cost. Most software today is developed using object-oriented technology, many languages have been developed to support object-oriented software design, most notably the Unified Modeling Language (UML). An activity diagram is one type of UML diagram that models the behavior of software. This paper proposes an approach to generate test cases directly from UML activity diagram using classification tree method. The proposed approach can help to generate test cases early in the development process and reduce the time needed to analyze the software specifications.

Keyword: Classification Tree Method, UML, Activity Diagram, Test Case

1. บทนำ

การทดสอบซอฟต์แวร์ เป็นขั้นตอนที่สำคัญและมีค่าใช้จ่ายสูงถึงร้อยละ 50 ของค่าใช้จ่ายทั้งหมดในการผลิตซอฟต์แวร์ [1] หากการสร้างกรณีทดสอบสามารถทำได้โดยอัตโนมัติจะทำให้ค่าใช้จ่ายการสร้างกรณีทดสอบลดลงเป็นจำนวนมาก

ในปัจจุบันการพัฒนาโปรแกรมเชิงวัตถุ (Object-Oriented Software Development) กำลังได้รับความนิยม

เป็นอย่างมาก เนื่องจากแนวคิดเชิงวัตถุที่ใกล้เคียงกับความเป็นจริงทำให้สามารถพัฒนาระบบที่มีความซับซ้อนและมีขนาดใหญ่ได้ นอกจากความสะดวกรวดเร็วแล้วยังสามารถนำกลับมาแก้ไขเพิ่มเติมในภายหลังได้อย่างง่ายดาย จึงมีการสร้างเครื่องมือ มาช่วยในการวิเคราะห์และออกแบบระบบซอฟต์แวร์เชิงวัตถุเพื่อให้ดำเนินงานได้อย่างสะดวกและรวดเร็ว เช่น ภาษาการออกแบบเชิงโมเดล (Unified Modeling Language: UML) [2] ดังนั้นการนำโมเดลจากขั้นตอนนี้มาใช้ในการสร้างกรณีทดสอบจะทำให้สามารถสร้างกรณีทดสอบได้ในขั้นตอนเริ่มต้นของขั้นตอนการออกแบบซอฟต์แวร์ แผนภาพกิจกรรม (Activity Diagram) เป็นแผนภาพหนึ่งของภาษาการออกแบบเชิงโมเดล ซึ่งเป็นแผนภาพที่แสดงขั้นตอนการทำงานของซอฟต์แวร์ตั้งแต่เริ่มต้นจนสิ้นสุด ส่วนสำคัญส่วนหนึ่งของแผนภาพกิจกรรมที่มีผลในการกำหนดกิจกรรมถัดไป และเป็นตัวกำหนดเส้นทางการทำกิจกรรมต่างๆ คือ ส่วนการตัดสินใจ (decision point) พบว่าเส้นทางการตัดสินใจที่จะทำกิจกรรมถัดไปหลังออกจากส่วนการตัดสินใจจะขึ้นอยู่กับข้อมูลนำเข้า (input) ที่ได้รับจากกิจกรรมที่เกิดก่อนหน้าการตัดสินใจ โดยข้อมูลนำเข้าจะถูกกำหนดเป็นเงื่อนไขสำหรับทางเลือกแต่ละเส้นทางที่แยกออกจากส่วนการตัดสินใจ ดังนั้นจากเงื่อนไขการตัดสินใจของเส้นทางต่างๆสามารถใช้เพื่อระบุขอบเขตของข้อมูลนำเข้า(input domain) ที่สำคัญต่อการทำงานของซอฟต์แวร์ได้

ดังนั้นบทความนี้จึงเสนอวิธีการสร้างกรณีทดสอบจากแผนภาพกิจกรรมโดยใช้หลักการต้นไม้การจำแนก (Classification Tree Method: CTM) ซึ่งช่วยลดเวลาในการวิเคราะห์เอกสารความต้องการเพื่อระบุขอบเขตของข้อมูลนำเข้า โดยวิธีการที่นำเสนอสามารถระบุขอบเขตของข้อมูลนำเข้า ได้โดยอัตโนมัติจากแผนภาพกิจกรรม จากนั้นนำข้อมูลนำเข้า ที่ระบุได้มาสร้างเป็นต้นไม้การจำแนกและตารางกรณีทดสอบ (test case table) เพื่อหากรณีทดสอบที่เหมาะสม

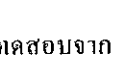
บทความนี้ประกอบด้วยหัวข้อดังต่อไปนี้ หัวข้อที่ 2 นำเสนอทฤษฎีและงานวิจัยที่เกี่ยวข้อง หัวข้อที่ 3 อธิบายวิธีการสร้างกรณีทดสอบจากแผนภาพกิจกรรมโดยใช้หลักการ CTM หัวข้อที่ 4 เสนอตัวอย่างการสร้างกรณีทดสอบจากแผนภาพกิจกรรมโดยหลักการ CTM และหัวข้อที่ 5 บทสรุปและงานในอนาคต

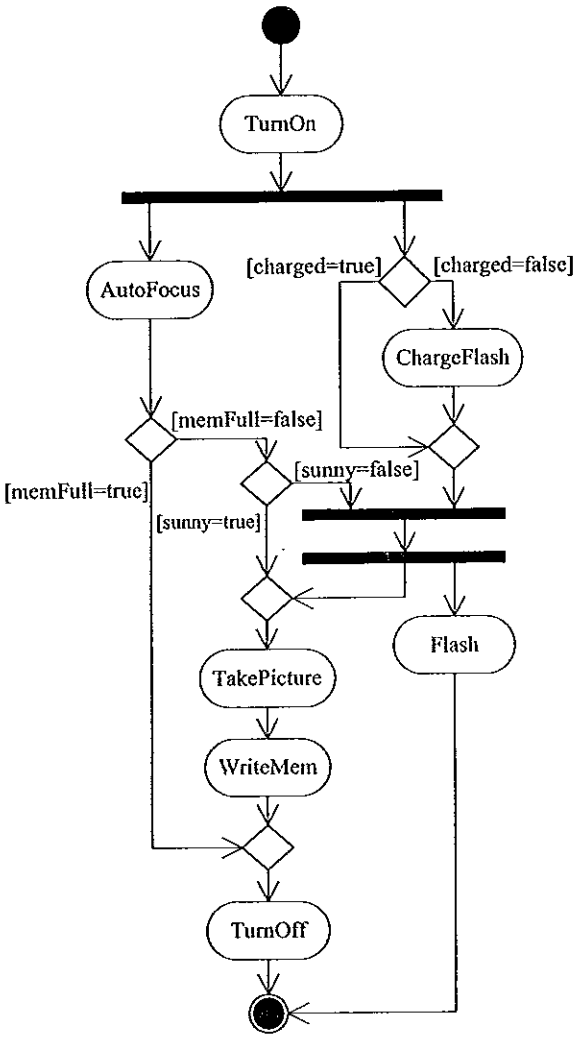
2. ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1 The Classification Tree Method (CTM)

วิธีการ Classification Tree Method เสนอโดย Grochtmann และ Grimm [3] เป็นวิธีหนึ่งที่ใช้เทคนิคการทดสอบแบบ black-box เทคนิคของวิธี CTM คือการแบ่ง input domain ออกเป็นหมวดหมู่แล้วแยกย่อยแต่ละหมู่ของ input domain ออกเป็น class ภายใต้เงื่อนไขเดียวกันโดยเขียนอยู่ในรูปของต้นไม้ ซึ่งเรียกว่าต้นไม้การจำแนก จากนั้นสร้างตารางกรณีทดสอบจากต้นไม้การจำแนกที่ได้ การสร้างกรณีทดสอบโดยใช้วิธีการ CTM จะได้กรณีทดสอบโดยการรวมโหนดของต้นไม้แต่ละกิ่ง โดยเลือกกรณีทดสอบที่เป็นไปได้โดยที่แต่ละกรณีทดสอบประกอบด้วยโหนดใบจากต้นไม้การจำแนกทุกหมวดหมู่ หมวดหมู่ละ 1 โหนด

2.2 UML Activity Diagram

แผนภาพกิจกรรม (activity diagram) [2] เป็นแผนภาพที่แสดงพฤติกรรมการทำงานของระบบซอฟต์แวร์ โดยอธิบายลำดับและขั้นตอนการทำงานที่เกิดขึ้นในซอฟต์แวร์มีลักษณะคล้ายกับผังงาน (flowchart) โดยจะเริ่มต้นด้วยสัญลักษณ์วงกลมทึบ ● หรือเรียกว่า Initial State และเชื่อมการทำงานต่างๆ ในแผนภาพด้วยลูกศรมีทิศทาง → และแสดงกิจกรรมการทำงานอยู่ภายในสัญลักษณ์  หรือเรียกว่า State ส่วนสำคัญส่วนหนึ่งที่มีผลในการกำหนดกิจกรรมถัดไป และเป็นตัวกำหนดเส้นทางการทำกิจกรรมต่างๆ คือส่วนการตัดสินใจ (decision point) ซึ่งแสดงโดยใช้ สัญลักษณ์ ◇ หรือ decision หากมีกิจกรรมที่ดำเนินไป พร้อมกันจะแสดงเส้นทางของกิจกรรมด้วยสัญลักษณ์  หรือเรียกว่า Transition (Join) และสัญลักษณ์  หรือเรียกว่า Transition (Fork) จุดสิ้นสุดของแผนภาพจะแทนด้วยสัญลักษณ์ ◉ หรือเรียกว่า Final State และมีสัญลักษณ์กรอบสี่เหลี่ยมหรือ swim lane ใช้แบ่งการทำงานในแต่ละส่วนออกจากกันตัวอย่างแผนภาพกิจกรรมดังแสดงในภาพที่ 1 แผนภาพกิจกรรมการทำงานของกล้องดิจิตอล [4]



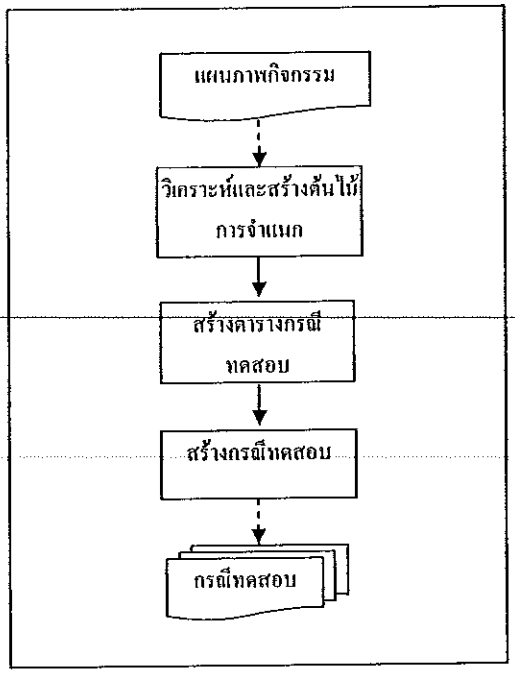
ภาพที่ 1: แผนภาพกิจกรรมการทำงานของกล้องดิจิตอล

2.3 งานวิจัยที่เกี่ยวข้อง

ปัจจุบันมีงานวิจัยที่นำเสนอวิธีเกี่ยวกับการสร้างกรณีทดสอบจากแผนภาพกิจกรรมหลายงานด้วยกัน อาทิเช่น Chen และ Poon [5] เสนองานวิจัยเรื่อง "Identification of Categories and Choices in Activity Diagrams" ซึ่งนำเสนอวิธีการสร้างกรณีทดสอบจาก specification ที่อยู่ในรูปของแผนภาพกิจกรรมโดยการกำหนดกลุ่มและ choice ของ input domain โดยผู้วิจัย ตามวิธีการแบ่งกลุ่ม (category – partition method) ตำแหน่งในแผนภาพกิจกรรมที่ช่วย ในการกำหนดกลุ่มและ choice คือส่วนการตัดสินใจ (decision -point) กรณีทดสอบที่ได้จากขั้นตอนวิธีนี้มีความถูกต้องและ ความเป็นไปได้สูง Chen Mingsong และคณะ [6] เสนองานวิจัยเรื่อง "Automatic Test Case Generation for UML Activity Diagram" ซึ่งเป็นการสร้างกรณีทดสอบจากแผนภาพกิจกรรมโดยใช้หลักการครอบคลุมแผนภาพกิจกรรม 3 ลักษณะคือ 1) State Coverage 2) Transition Coverage และ 3) Simple Path Coverage คณะวิจัยได้พัฒนาเครื่องมือ ที่เรียกว่า AGTCG ซึ่งช่วยในการสร้างกรณีทดสอบ แบบอัตโนมัติ เช่นเดียวกับ Wang Linzhang และคณะ [7] ที่นำเสนองานวิจัยเรื่อง "Generating Test Cases from UML Activity Diagram based on Gray-Box Method" ซึ่งสร้างกรณีทดสอบจากแผนภาพกิจกรรมโดยใช้หลักการ Gray-Box ที่พิจารณาเส้นทางการทำงานที่เป็นไปได้จากแผนภาพกิจกรรมพร้อมกับพิจารณา input/output ของระบบควบคู่ไปด้วย

3. การสร้างกรณีทดสอบจากแผนภาพกิจกรรมโดยใช้หลักการ CTM

งานวิจัยนี้เสนอการสร้างกรณีทดสอบจากแผนภาพกิจกรรมโดยใช้หลักการ Classification Tree Method (CTM) โดยกระบวนการสร้างกรณีทดสอบมีขั้นตอนดังภาพที่ 2 ซึ่งมีรายละเอียดขั้นตอน ดังต่อไปนี้



ภาพที่ 2: กระบวนการสร้างกรณีทดสอบจากแผนภาพกิจกรรม

ขั้นตอนที่ 1 วิเคราะห์และสร้างต้นไม้การจำแนก

1.1) วิเคราะห์แผนภาพกิจกรรมแต่ละส่วนการตัดสินใจ (decision point) ที่เป็นทางเลือก ส่วนการตัดสินใจแต่ละส่วนคือลักษณะของ input domain ที่เป็นไปได้ 1 หมวดหมู่ (category)

1.2) นำลักษณะของ input domain 1 หมวดหมู่ มาสร้างต้นไม้การจำแนก 1 ต้น ซึ่งมีขั้นตอนการสร้างดังนี้

1) สร้างโหนดเริ่มต้นสำหรับลักษณะของ input domain หมวดหมู่นี้ซึ่งพิจารณาจากส่วนการตัดสินใจ

2) สร้างโหนดลูกสำหรับ input domain ในหมวดหมู่นี้ เรียกว่า class โดย class ต่างๆ พิจารณาจากเส้นทางที่ออกจากส่วนการตัดสินใจ

ขั้นตอนที่ 2 สร้างตารางกรณีทดสอบ

2.1) นำต้นไม้การจำแนกที่มีความสัมพันธ์กัน มารวมกันให้เป็นต้นเดียว โดยพิจารณาความสัมพันธ์จากแผนภาพกิจกรรม

2.2) นำต้นไม้แต่ละต้นจากขั้นตอนที่ 2.1 มารวมกันเป็นต้นใหญ่โดยเพิ่มโหนดเริ่มต้นเพื่อเชื่อมต้นไม้เหล่านั้นเข้าด้วยกัน

2.3) สร้างตารางโดยลากเส้นคอลัมน์จากโหนดใบของต้นไม้การจำแนกแต่ละหมวดหมู่

2.4) ลากเส้นแถวของตารางเพื่อใช้กำหนดกรณีทดสอบ

ขั้นตอนที่ 3 สร้างกรณีทดสอบ

3.1) กำหนดกรณีทดสอบแต่ละกรณีโดยทำสัญลักษณ์บนตำแหน่งที่เส้นแถวลากตัดกับเส้นคอลัมน์จากโหนดใบของต้นไม้การจำแนกทุกหมวดหมู่ หมวดหมู่ละ 1 โหนด

3.2) เส้นแถวที่ทำสัญลักษณ์ 1 เส้น คือกรณีทดสอบ 1 กรณี

4. ตัวอย่างการสร้างกรณีทดสอบโดยใช้หลักการ CTM

เพื่อแสดงตัวอย่างการสร้างกรณีทดสอบโดยวิธีที่ได้นำเสนอ ในงานวิจัยนี้ได้เลือกใช้แผนภาพกิจกรรมการทำงานของ กล้องดิจิตอล ดังภาพที่ 1 ซึ่งถูกนำเสนอเป็นตัวอย่างในการสร้างกรณีทดสอบจากบทความเรื่อง Automatic Verification and Performance Analysis of Time-Constrained SysML Activity Diagrams [5] จากแผนภาพกิจกรรมการทำงานของกล้องดิจิตอลข้างต้นสามารถนำมาสร้างกรณีทดสอบด้วยหลักการ CTM ได้ดังต่อไปนี้

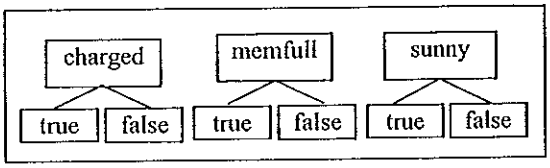
1) ขั้นตอนการวิเคราะห์และสร้างต้นไม้การจำแนก

1.1) จากแผนภาพกิจกรรมการทำงานของกล้องดิจิตอล จะพบว่ามีส่วนการตัดสินใจปรากฏอยู่ 3 ตำแหน่ง ซึ่งสามารถแบ่ง input domain ออกเป็น 3 category ซึ่งแต่ละ category ประกอบด้วย class ดังตารางที่ 1

ตารางที่ 1: category และ class สำหรับแผนภาพกิจกรรมการทำงาน ของกล้องดิจิตอล

category	class	
1. charged	true	false
2. memfull	true	false
3. sunny	true	false

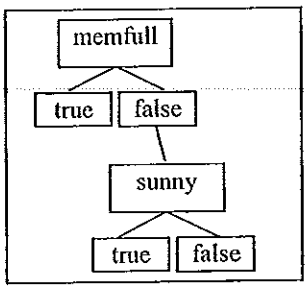
1.2) จาก category และ class สามารถสร้างต้นไม้การจำแนกได้ทั้งหมด 3 ต้น ดังภาพที่ 3



ภาพที่ 3: ต้นไม้การจำแนกจากแผนภาพกิจกรรมการทำงานของกล้องดิจิตอล

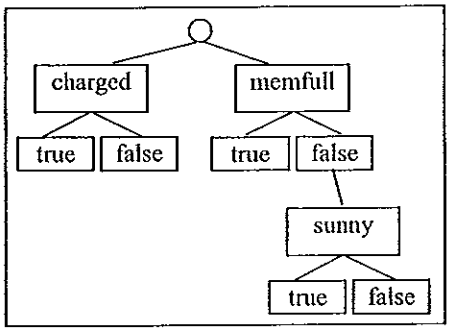
2) ขั้นตอนการสร้างตารางกรณีทดสอบ

2.1) พิจารณาความสัมพันธ์ของต้นไม้การจำแนกแต่ละต้นจากแผนภาพกิจกรรมพบว่า ต้นไม้ sunny สัมพันธ์กับต้นไม้ memfull ผ่านทางโหนด false จึงรวมต้นไม้ sunny เข้ากับต้นไม้ memfull ส่วนต้นไม้ charged ไม่มีความสัมพันธ์กับต้นไม้ใดเลย ต้นไม้ที่เกิดจากการรวมกันของต้นไม้การจำแนก sunny และต้นไม้การจำแนก memfull ตามความสัมพันธ์แสดงได้ดังภาพที่ 4



ภาพที่ 4: ต้นไม้ที่เกิดจากการรวมกันของต้นไม้การจำแนก sunnyและต้นไม้การจำแนก memfull

2.2) เพิ่มโหนดเริ่มต้นเพื่อรวมต้นไม้การจำแนก charged และต้นไม้ที่เกิดจากการรวมกันของต้นไม้การจำแนก memfull และ ต้นไม้การจำแนก sunny เป็นต้นเดียวกัน ซึ่งต้นไม้ที่รวมกันเป็นต้นเดียวสามารถแสดงได้ดังภาพที่ 5



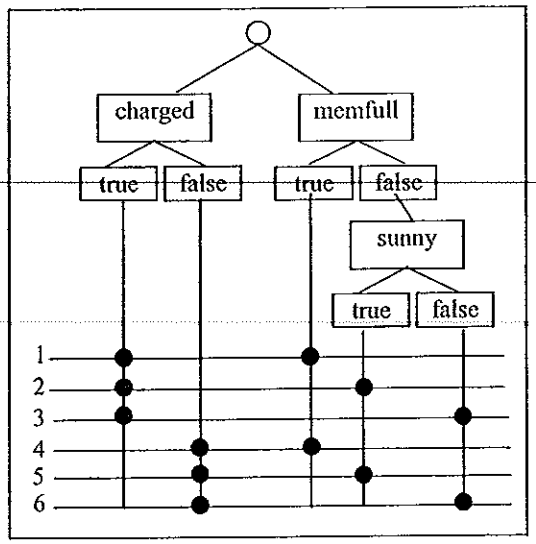
ภาพที่ 5: ต้นไม้การจำแนกที่รวมกันตามความสัมพันธ์

2.3) ลากเส้นคอลัมน์จากโหนดใบของต้นไม้การจำแนกแต่ละโหนด

2.4) ลากเส้นแถวของตารางเพื่อใช้กำหนดกรณีทดสอบ

3) ขั้นตอนการสร้างกรณีทดสอบ

กำหนดกรณีทดสอบแต่ละกรณีโดยทำสัญลักษณ์บนเส้นแถวของตาราง โดยแต่ละแถวที่ทำสัญลักษณ์คือ 1 กรณีทดสอบซึ่งประกอบด้วยโหนดใบจากต้นไม้การจำแนกแต่ละต้น ต้นละ 1 โหนด ซึ่งกรณีทดสอบสำหรับการทำงานของกล้องดิจิตอล ที่ได้จากการขั้นตอนนี้แสดงดังภาพที่ 6



ภาพที่ 6: กรณีทดสอบที่ได้จากแผนภาพกิจกรรมโดยใช้วิธี CTM

5. บทสรุปและงานในอนาคต

การสร้างกรณีทดสอบเป็นขั้นตอนหนึ่งที่สำคัญในการทดสอบซอฟต์แวร์ งานวิจัยนี้ได้นำเสนอวิธีการสร้างกรณีทดสอบจากแผนภาพกิจกรรมโดยใช้หลักการ CTM ซึ่งผลที่ได้รับคือสามารถเริ่มต้นการทดสอบซอฟต์แวร์ได้ตั้งแต่ขั้นตอนการออกแบบและช่วยลดเวลาในการวิเคราะห์เอกสารความต้องการได้อย่างมากเพราะใช้แผนภาพกิจกรรม ในอนาคตผู้วิจัยจะปรับปรุงวิธีการที่นำเสนอเพื่อประยุกต์ใช้กับแผนภาพกิจกรรมที่มีการวนลูป พร้อมทั้งจะทำการประเมินประสิทธิภาพของกรณีทดสอบที่สร้างขึ้นโดยใช้วิธี mutation testing นอกจากนี้ผู้วิจัยจะเพิ่มเทคนิคอื่นๆ เช่น เพิ่มการพิจารณาเงื่อนไขในแผนภาพกิจกรรมร่วมด้วย เพื่อให้กรณีทดสอบ มีจำนวนลดลงและจะพัฒนาเครื่องมือสนับสนุนวิธีการที่นำเสนอเพื่อการสร้างกรณีทดสอบโดยอัตโนมัติและนำไปใช้ทดสอบระบบซอฟต์แวร์ที่มีขนาดใหญ่และซับซ้อนมากขึ้น

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] M.Grochtmann, K.Grimm, and J.Wegener, "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, pp25 – 28, October 1993.

[4] Jarraya, Yosr; Soeanu, Andrei; Debbabi, Mourad; Hassaine, and Fawzi, "Automatic Verification and Performance Analysis of Time-Constrained SysML Activity Diagrams," in *Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)*, pp 515 – 522, 2007.

[5] 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.

[6] C. Mingsong, Q. Xiaokang, and L. Xuandong, "Automatic Test Case Generation for UML Activity Diagrams, In International Conference on Software Engineering", *Proceedings of the 2006 International workshop on Automation of Software Test*, ACM, May 2006.

[7] L.Wang, J.Yuan, X.Yu, J.Hu, X.Li, and G.Zheng, "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.

ภาคผนวก ข.

ผลงานตีพิมพ์

เรื่อง	Generating Test Cases from UML Activity Diagrams Using the Condition – Classification Tree Method
งานประชุมวิชาการ	2010 2nd International Conference on Software Technology and Engineering (ICSTE 2010)
สถานที่	San Juan, Puerto Rico, USA
วันที่	3 - 5 ตุลาคม 2553

2010 2nd International Conference on Software Technology and Engineering(ICSTE)

Generating Test Cases from UML Activity Diagrams using the Condition-Classification Tree Method

Supaporn Kansomkeat, Phachayanee Thiket
Department of Computer Science
Faculty of Science, Prince of Songkla University
Hat Yai, Songkhla, 90112, Thailand
email: {supaporn.k,s511022041}@psu.ac.th

Jeff Offutt
Software Engineering
George Mason University
Fairfax, VA 22030, USA
email: offutt@gmu.edu

***Abstract*—A key technical challenge in software testing is the design of useful test cases. Test design can be based on a variety of software artifacts, including requirements, designs, or even the implementation. The Unified Modeling Language (UML) is now widely used to describe object-oriented designs. This paper focuses on one UML diagram, the activity diagram, which is used to model software behavior. This paper proposes the Condition-Classification Tree Method for generating test cases from activity diagrams. Activity diagrams are used to generate condition-classification trees, which are then used to create test case tables and test cases. The paper presents experimental data that show the proposed method can help generate a relatively small number of test cases at reasonable cost, early in development.**

Keywords- software testing; test generation; UML activity diagrams

I. INTRODUCTION

Software testing [1] usually involves executing a program on a set of tests and comparing the actual output results with the expected outputs. A key technical activity in software testing is the design of test cases. The goal of this research is to give testers a way to design and generate effective tests early during development. In particular, we focus on developing tests during design specification.

The Unified Modeling Language (UML) [2] is a collection of diagrams for specifying, visualizing and documenting software. The languages are primarily intended to be used with object-oriented software. By using the UML, complex systems are designed and modeled through a collection of views of a model. The UML defines nine separate diagrams, including activity diagrams, which are used to model software behavior. In activity diagrams, the flow of control is based on decision points represented by diamonds. The transitions from decision points contain guard conditions, which are used to determine which path from the decision point is taken. Decisions along with their guard conditions define alternate paths through a work flow. Chen et al. [3] introduced a technique for identifying categories and choices from activity diagrams to generate test cases. They focus on decision points and guard conditions in activity diagrams to identify categories and choices. The paper did not describe a complete generation processes. Mingsong [4] proposed generating tests from activity diagrams as design specifications. They get relatively small test suites that meet the test adequacy criterion.

This paper proposes a Condition-Classification Tree Method (CCTM) that extends the Classification-Tree Method by using activity diagrams. First, an activity diagram is analyzed to gather control flow information based on decision points and guard conditions. Then, this information is used to derive Condition-Classification Trees. Finally, the trees are used to generate a test case table and then test cases. Generating tests from activity diagrams allow test cases to be generated early, during software design, greatly increasing the chances of finishing testing before the software is scheduled to ship.

The rest of the paper is organized as follows. Section 2 introduces the background. Section 3 presents the Condition-Classification Tree Method. Section 4 discusses a case study that was conducted to evaluate the CCTM method. Finally, conclusions and future work are presented in section 5.

II. BACKGROUND

The *Condition-Classification Tree Method* extends the Classification-Tree Method (CTM) and generates Condition-Classification Trees from UML activity diagrams. This section provides a brief overview of these topics.

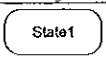
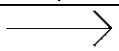
A. UML Activity Diagram

Activity diagrams are used to model the workflow behavior of a system. The diagrams describe behavior by modeling the sequence of activities performed. Figure 1 shows a UML activity diagram for a small sales order processing system for a retail shop called SALES [5]. For each purchase, SALES accepts the customer and purchase details and decides whether this purchase should be approved. The basic elements in a UML Activity Diagram are activities and transitions. An activity is a function that the software performs, and is represented by a rounded rectangle. Each activity diagram has one start activity, where the sequence of actions begins, and one finish activity, where the sequence of actions ends. The start activity is indicated by a solid circle and the finish activity by a bull's eye. Transitions show control flow among activities, and are drawn as drawn directed arrows. An activity diagram can be used to describe complex sequences of activities, with support for both conditional and parallel behavior. Conditional behavior is described by a branch and a merge. A branch has a single

978-1-4244-8665-6/10/$26.00 © 2010 IEEE

incoming transition and at least two guarded outgoing transitions. A merge has at least two incoming but only one outgoing transition. Parallel behavior is indicated by a fork and a join. A solid thick bar is called a synchronization bar. The fork has one transition entering into a synchronization bar and at least two transitions exiting, all of which will be taken. The join represents the end of the parallel behavior and at least two transitions entering into a synchronization bar, and only one leaving. A Swim lane groups related activities into one column. Table I shows these activity diagram symbols.

TABLE I. BASIC ACTIVITY DIAGRAM SYMBOLS

Symbols	Names
	Start activity
	Final activity
	Activity
	Branch, Merge
	Transition
	Swim lane
	Join
	Fork

B. Classification-tree Method

Grochtmann and Grimm [6] proposed the classification-tree method as an improvement on the category-partition method. The classification-tree method helps testers construct test cases from specifications by using classification trees. Classifications are criteria for partitioning the input domain of the program to be tested, and classes are the disjoint subsets of values for each classification. A classification tree organizes the classifications and classes into a hierarchical structure according to the specification. The classification-tree method has four major steps:

1) Analyze the software specification and identify all the classifications and the associated classes.

2) Construct a classification tree from the classifications and classes.

3) Draw a test case table under the classification tree.

4) Identify all possible combinations of classes from the test case table. Each combination of classes represents one test case.

III. GENERATING TEST CASES FROM ACTIVITY DIAGRAMS USING THE CONDITION-CLASSIFICATION TREE METHOD

The CCTM generates test cases from UML activity diagrams. Figure 2 overviews the test generation methodology.

The CCTM method works in three steps (A, B and C), as described below.

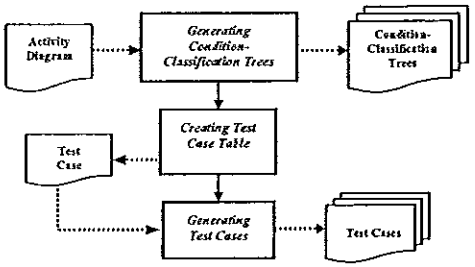


Figure 2. Condition-Classification Tree Method

Step A: Generate condition-classification trees using UML activity diagram

The UML activity diagrams are analyzed to extract behaviors of the system at the decision points and guard conditions. Each decision point and its guard conditions are used to create a condition-classification tree as follows:

1) A decision point is placed into a category that is represented by a start node called the category node. For example, in Figure 3, the decision point B is represented by the category node B, as shown in Figure 4.

2) Transitions that start at the decision point being considered are analyzed to create child nodes of the category node. These child nodes are called option nodes. In Figure 3, for example, the decision point considered is B. The two transitions out of B are considered to generate two option nodes b1 and b2, as illustrated in Figure 4.

3) To identify conditions in the tree, decision points before the decision point being considered are analyzed to collect their guard conditions along with path that was taken. Each collected guard condition is enclosed by square brackets [ ] and put in the top of the tree. For example, Figure 3 shows that the decision point A with guard condition a1 must be taken before decision point B. Therefore, we put the condition "[a1]" in top of the tree shown as Figure 4.

Step B: Create test case table

1) The condition-classification trees created in step A are sorted by the number of conditions from small to large. If some trees have the same number of conditions,

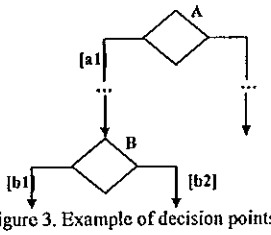


Figure 3. Example of decision points

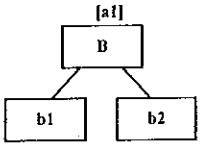


Figure 4. A simple Condition-Classification Tree

they can be sorted in any sequence. For example, seven condition-classification trees are created for SALES. The tree Customer status has one condition and the tree Whether transaction has two conditions, thus the tree Whether transaction follows the tree Customer status. Figure 5 shows sorted condition-classification trees for SALES.

2) The grids of the test case table are drawn under the sorted condition-classification trees. The columns of the table correspond to the option nodes of the sorted condition-classification trees. The rows represent potential test cases. Figure 5 shows the test case table for SALES.

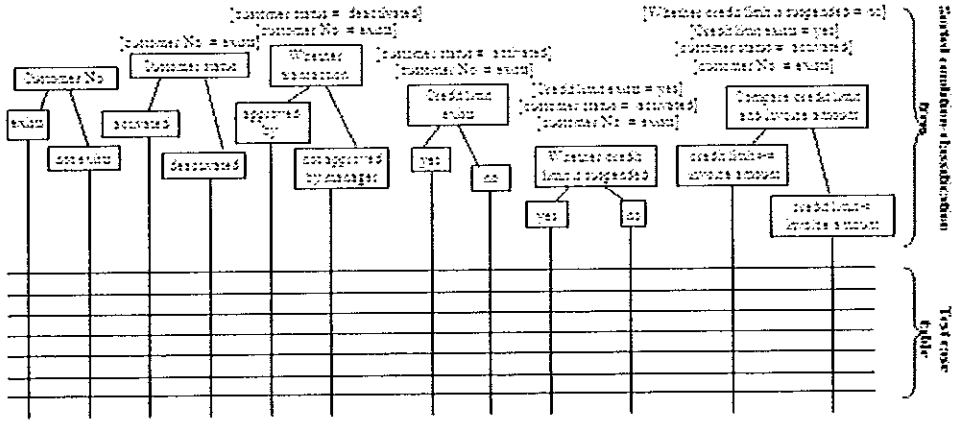


Figure 5. The sorted condition-classification trees and test case table for SALES

Step C: Generate test cases

Each row of the test case table represents a test case. A test case is constructed in the table by marking a combination of option nodes that depends on the conditions of the condition-classification trees. The marking is carried out follows:

1) A series of sorted condition-classification trees that have conditions is considered one by one from left to right. Each option node of the considered tree is marked with option nodes of left trees that correspond to conditions of the tree under consideration. In Figure 5, for example, the considered tree *Customer status* has the condition *[customer No. = exists]* that corresponds to the left tree *Customer No.*, thus two test cases are marked:

1. The option node *activated* of the tree *Customer Status* is marked with the option node *exists* of the tree *Customer No.*, and
2. The option node *deactivated* of the tree *Customer Status* is marked with the option node *exists* of the tree *Customer No.* as shown in Figure 6.

2) If the conditions of the considered tree are the same as the conditions of the tree to its immediate left, each option node of the tree under consideration will be marked by using the test case marked on the previous tree. For example, the tree *whether transaction* has the conditions *[customer status = deactivated]* and *[customer No. = exists]*. By the condition *[customer status = deactivated]*, each option node of the tree *whether transaction* should be marked with the option node *deactivated* from the tree *Customer status*.

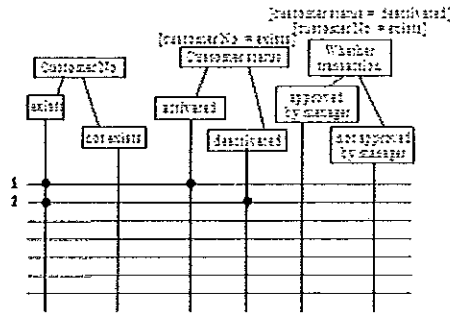


Figure 6. Test case generation by considering *Customer status* tree

Because the condition *[customer No. = exists]* of the tree *whether transaction* is the same as in the tree *Customer status*, this marking should be done continually by using the second test case as shown in Figure 7.

3) If any option nodes from the condition-classification trees are not marked, each option node is marked to be a test case. Test case 7 in Figure 8 is an example.

Figure 8 shows seven test cases generated by the above steps for SALES. The original Classification-tree method [5] generated 30 test cases for SALES, more than four times the number generated by our CCTM method.

IV. CASE STUDY

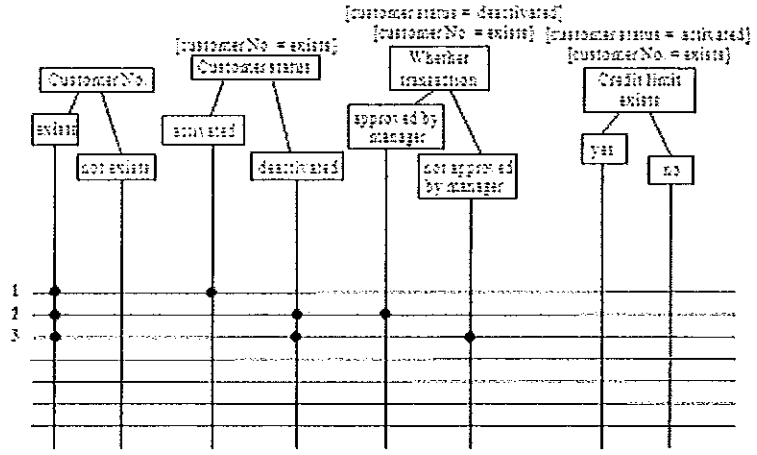


Figure 7. Test case generation by considering *Whether transaction* tree

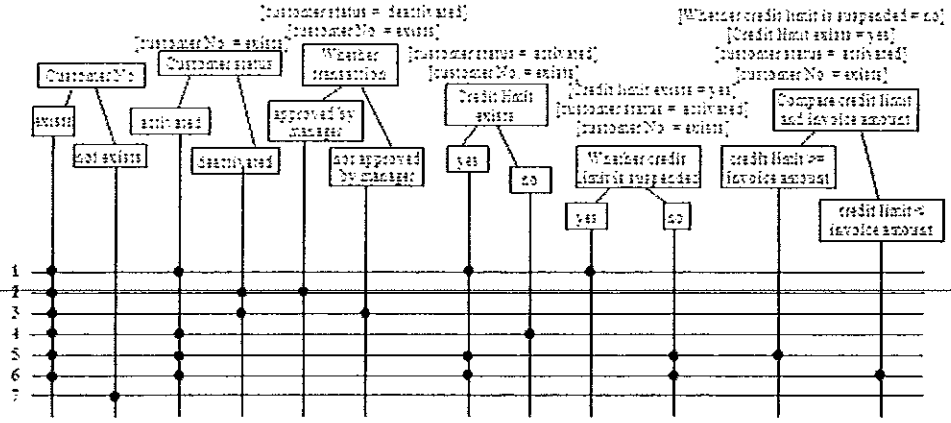


Figure 8. Test cases for SALES

This section presents two small applications, Payment and Car Parking, which are used to evaluate CCTM's ability to generate test cases. The Payment application calculates monthly payments when a user buys goods. The Car Parking application manages car parking times. These applications were designed with the help of activity diagrams and implemented in Java (Payment class and CarParking class). We generated ten tests for Payment and four for Car Parking. Table 2 shows the description of each class. Column "SLOC" shows the source lines of code. Column "#Methods" shows the number of methods.

TABLE II. Description of each class used in case study

Class name	SLOC	# Methods
Payment	74	3
CarParking	80	4

We used mutation analysis [1,7] to introduce faults into the Java classes. Mutation analysis is a fault-based testing strategy that starts with a program to be tested and makes numerous small syntactic changes into the original program (or the specification). The modified programs are called mutants. Mutation analysis is used to evaluate test sets, based on the number of mutants that fail, or are killed. The ratio of killed mutants to total mutants is called the mutation score. Mutants are obtained by applying mutation operators that introduce simple changes into the original program. For example, an arithmetic operator may be changed from addition to subtraction.

This study used a set of replacement operators designed for Java by Ma and Offutt [8,9] (not all muJava operators were used, just the replacement operators). Mutants were created by hand. There are 55 mutants for Payment and 77 for CarParking, as shown in Table 3.

TABLE III. Case study results on the Payment and CarParking classes

Class name	# Test Cases	# Mutants	Mutants Killed	Mutation Score
Payment	10	55	48	0.873
CarParking	4	77	64	0.831

V. CONCLUSIONS

This paper provided a method to automatically gather control flow information from decision points and guard conditions in activity diagrams. This information is used to construct condition-classification trees. These trees are then used to generate a test case table and test cases. Experimental data show that tests generated by the CCTM method have strong ability to detect faults. We have implemented a test case generation tool that supports CCTM. In the future, we intend to improve the tool to automate the entire test process and evaluate our method with more complex systems.

REFERENCES

[1] P. Ammann and J. Offutt, "Introduction to Software Testing," Cambridge University Press, USA, 2008, ISBN 0-52188-038-1.

[2] G. Booch, J. Rumbaugh, and I. Jacobson, "The Unified Modeling Language User Guide," Object Technology Series, Addison-Wesley Longman, Inc, 1998.

[3] T. Y. Chen, Pak-Lok Poon , Sau-Fun Tang and T. H. Tse, "Identification of Categories and Choices in Activity Diagrams," Proceedings of the 5th International Conference on Quality Software (QSIC 2005), IEEE Computer Society, Sep. 2005, pp. 55-63.

[4] C. Mingsong, Q. Xiaokang, and L. Xuandong, "Automatic Test Case Generation for UML Activity Diagrams," International Conference on Software Engineering, Proceedings of the 2006 International Workshop on Automation of Software Test (AST 2006), ACM, May 2006, pp. 2-8.

[5] T. Y. Chen and P. L. Poon, "Teaching Black Box Testing," Proceedings of the 1998 International Conference on Software Engineering: Education and Practice, IEEE Computer Society Press, Jan. 1998, pp 324-329.

[6] M. Grochtmann and K. Grimm, "Classification Trees for Partition Testing," Software Testing, Verification & Reliability, vol. 3, Jun. 1993, pp. 63-82,.

[7] R. A. Demillo, R. J. Lipton, and R. G. Sayward, "Hints on Test Data Selection: Help for the Practicing Programmer," IEEE Computer, vol. 11, Apr. 1978, pp. 34-41.

[8] J. Offutt, Y. S. Ma and Y. R. Kwon, "The Class-Level Mutants of MuJava," International Conference on Software Engineering, Proceedings of the 2006 International Workshop on Automation of Software Test (AST 2006), ACM, May 2006, pp. 74-84.

[9] Y. S. Ma and J. Offut, "Description of Method-level Mutation Operators for Java," http://cs.gmu.edu/~offutt/mujava/mutopsMethod.pdf (accessed 29 April 2010).

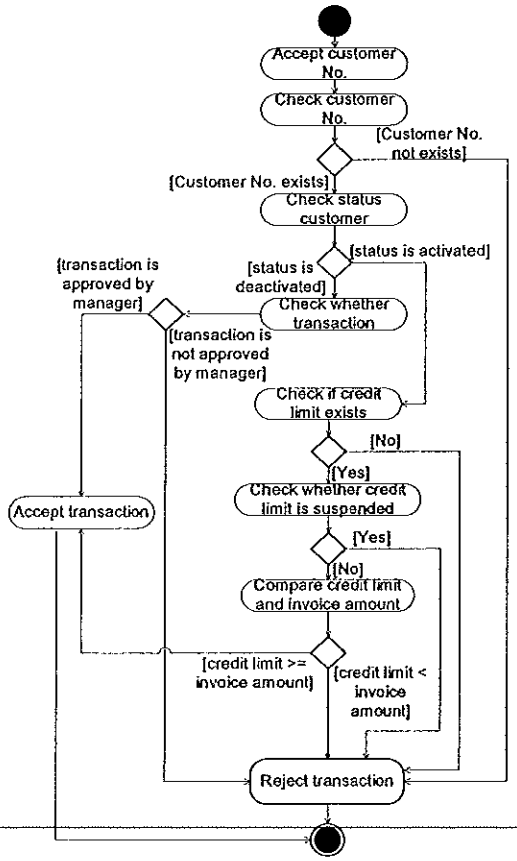


Figure 1. An activity diagram for SALES

ภาคผนวก ค.

ผลงานตีพิมพ์

เรื่อง	แบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไขสำหรับสร้างกรณีทดสอบบนพื้นฐานของแผนภาพกิจกรรมของ UML
งานประชุมวิชาการ	การประชุมวิชาการระดับประเทศด้านเทคโนโลยีสารสนเทศ (NCIT2010) ครั้งที่ 3
สถานที่	กรุงเทพมหานคร ประเทศไทย
วันที่	28 - 29 ตุลาคม 2553

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 พบว่าแผนภาพกิจกรรมถูกนำมาใช้ในการะบวนการทดสอบซอฟต์แวร์ในขั้นตอนก่อนการสร้างกรณีทดสอบเท่านั้น 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

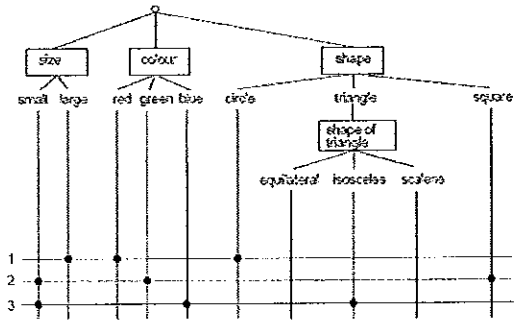
1) วิเคราะห์ specification ของซอฟต์แวร์และแบ่ง input domain ออกเป็นหมวดหมู่พร้อมทั้งระบุ class ที่เป็นไปได้ในแต่ละหมวดหมู่

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

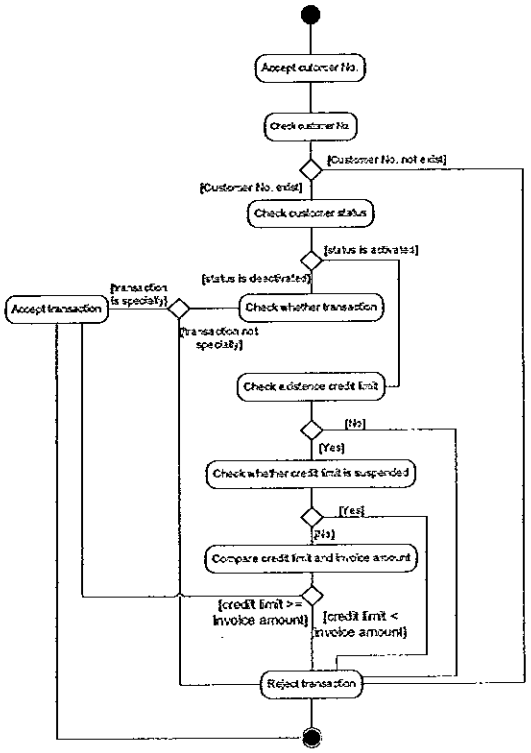
ตัวอย่างการใช้วิธี CTM สร้างกรณีทดสอบจาก requirement specification ของระบบ computer vision [7] ซึ่งกำหนด input domain ออกเป็น 3 กลุ่มได้แก่ size, color และ shape จากนั้นระบุ input domain ย่อยที่เป็นไปได้ในแต่ละกลุ่มแล้วสร้างต้นไม้การจำแนกและตารางกรณีทดสอบ ดังแสดงรายละเอียดดังรูปที่ 1



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

3. UML Activity Diagram

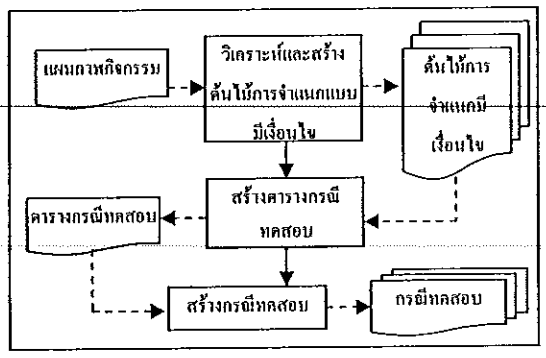
แผนภาพกิจกรรม (activity diagram) [2] เป็นแผนภาพที่แสดงพฤติกรรมการทำงานของระบบซอฟต์แวร์ โดยอธิบายลำดับและขั้นตอนการทำงานที่เกิดขึ้นในซอฟต์แวร์ แผนภาพกิจกรรมมีลักษณะคล้ายกับ ผังงาน (flowchart) โดยจะเริ่มต้นด้วยสัญลักษณ์วงกลมทึบ ● เรียกว่า Initial State และเชื่อมการทำงานต่างๆ ในแผนภาพด้วยลูกศรมีทิศทาง → 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

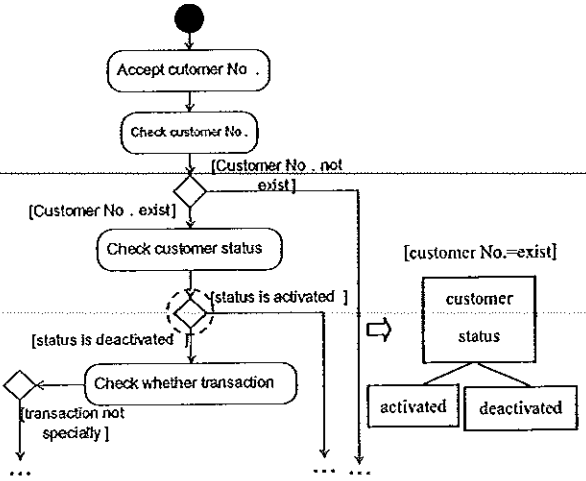
1.1) วิเคราะห์แผนภาพกิจกรรมเพื่อรวบรวมลำดับขั้นตอนการทำกิจกรรมต่างๆ โดยพิจารณาในส่วนการตัดสินใจ (decision point) ที่มีเงื่อนไขกำหนดทางเลือกกิจกรรมถัดไป โดยส่วนการตัดสินใจแต่ละส่วนแสดงลักษณะของ input domain ที่เป็นไปได้ 1 กลุ่ม (category)

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 ส่วนใดมาบ้างกรณีที่พบให้นำเงื่อนไขการตัดสินใจเหล่านั้นระบุไว้บนต้นไม้การจำแนกภายในเครื่องหมายวงเล็บล้ำมปู [ ] ดังตัวอย่างในรูปที่ 4 พบว่าก่อนส่วนการตัดสินใจ check customer statusได้ผ่านส่วนการตัดสินใจ check customer No. มาก่อนโดยผ่านทางเงื่อนไข exist ดังนั้นจึงระบุเงื่อนไข "[customer No.=exist]" ดังแสดงในรูปที่ 4



รูปที่ 4 ตัวอย่างการสร้างต้นไม้การจำแนกโดยพิจารณาส่วนการตัดสินใจ *check customer status*

ขั้นตอนที่ 2 สร้างตารางกรณีทดสอบ

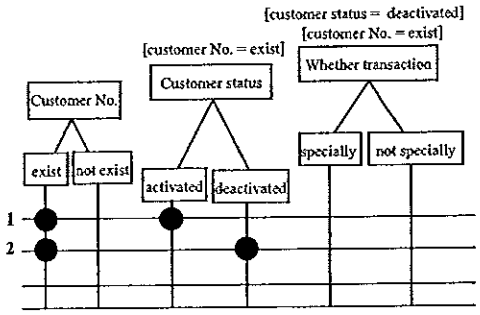
2.1) นำต้นไม้การจำแนกที่ระบุเงื่อนไขทั้งหมดมาวางเรียงกัน โดยเรียงลำดับตามจำนวนเงื่อนไขที่ระบุจากจำนวนน้อยไปหาจำนวนมาก กรณีที่จำนวนเงื่อนไขของต้นไม้การจำแนกต้นใดมีค่าเท่ากันให้วางต้นใดก่อนหรือหลังก็ได้ ตัวอย่างจากแผนภาพกิจกรรมการขายสินค้าในรูปที่ 2 สามารถสร้างต้นไม้การจำแนกที่ระบุเงื่อนไขได้ทั้งหมด 6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

2.2) สร้างตารางกรณีทดสอบโดยลากเส้นกลอัลกน์จากโหนดลูกของต้นไม้การจำแนกที่ระบุเงื่อนไขแต่ละต้นและลากเส้นแถวของตารางเพื่อใช้กำหนดกรณีทดสอบ รูปที่ 5 ส่วนล่างแสดงตารางกรณีทดสอบของการขายสินค้า

ขั้นตอนที่ 3 สร้างกรณีทดสอบ

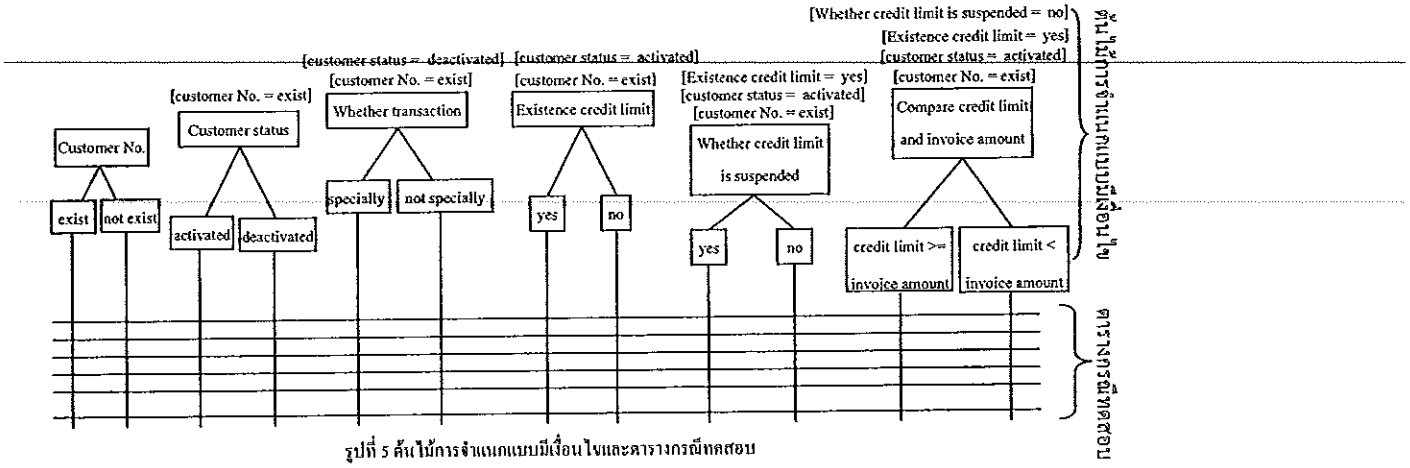
3.1) กำหนดกรณีทดสอบแต่ละกรณีโดยทำสัญลักษณ์ลงบนเส้นแถวของตาราง โดยพิจารณาจากต้นไม้การจำแนกที่มีเงื่อนไขระบุทีละต้นจากทางด้านซ้ายไปทางด้านขวา ซึ่งเงื่อนไขจะถูกใช้เพื่อระบุว่าแต่ละโหนดในต้นไม้นั้นสามารถเลือกจับกับโหนดลูกในต้นไม้การจำแนกต้นใดได้บ้าง กรณีที่สามารถจับคู่ได้จะทำสัญลักษณ์ลงบนเส้นแถวในตารางดังตัวอย่างในรูปที่ 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

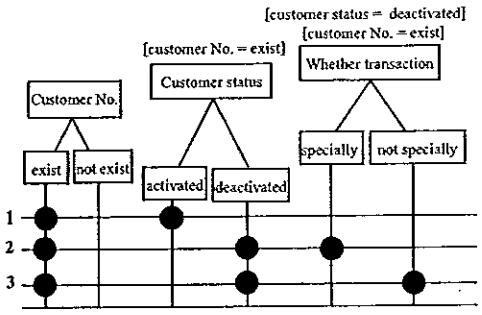


รูปที่ 6 กรณีทดสอบที่ได้จากการพิจารณาเงื่อนไขบนต้นไม้การจำแนก *customer status*

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



รูปที่ 5 ต้นไม้การจำแนกแบบมีเงื่อนไขและตารางกรณีทดสอบ



รูปที่ 7 กรณีทดสอบที่ได้จากการพิจารณาเงื่อนไขบนต้นไม้การจำแนก *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.

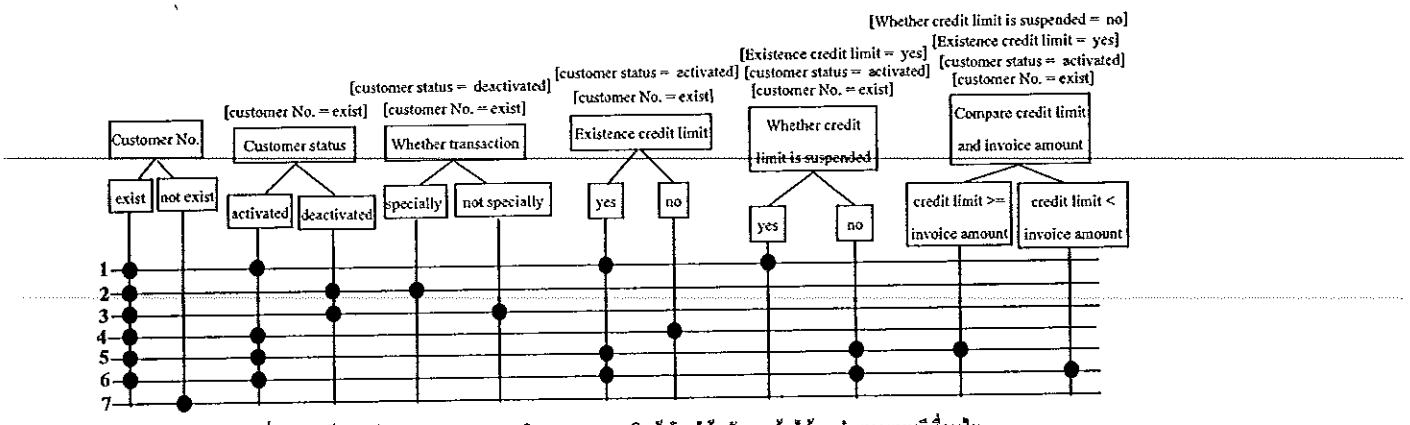
[4] W. Sinzhang, Y. Jiesong, Y. Xiaofeng, H. Jun, L. Xuandong, and Z. Guoliang, "Generating Test Cases from UML Activity Diagram based on Gray-Box Method", In Proceedings of the 11th Asia-Pacific Software Engineering Conference(APSEC), IEEE Computer Society, November. 2004.

[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

ประวัติผู้เขียน

ชื่อ สกุล นางสาวปรัชญานีย์ ไทยเกิด

รหัสประจำตัวนักศึกษา 5110220041

วุฒิการศึกษา

วุฒิ	ชื่อสถาบัน	ปีที่สำเร็จการศึกษา
วท.บ. (วิทยาการคอมพิวเตอร์)	มหาวิทยาลัยราชภัฏยะลา	2548

ทุนการศึกษา

ทุนการศึกษาในโครงการส่งเสริมการผลิตครูที่มีความสามารถพิเศษด้านวิทยาศาสตร์และคณิตศาสตร์ (สควค.)

ตำแหน่งและสถานที่ทำงาน

ตำแหน่ง ครู คศ.1

สถานที่ทำงาน โรงเรียนชะอวด อำเภอชะอวด จังหวัดนครศรีธรรมราช

การตีพิมพ์เผยแพร่ผลงาน

ปรัชญานีย์ ไทยเกิด และสุภาภรณ์ กานต์สมเกียรติ. 2553. การสร้างกรณีทดสอบจาก UML Activity Diagram แบบอัตโนมัติด้วยวิธีการต้นไม้การจำแนก. การประชุมทางวิชาการระดับชาติด้านคอมพิวเตอร์และเทคโนโลยีสารสนเทศ (NCCIT 2010) ครั้งที่ 6. กรุงเทพมหานคร ประเทศไทย, 3-5 มิถุนายน 2553. หน้า 349-354.

ปรัชญานีย์ ไทยเกิด สุภาภรณ์ กานต์สมเกียรติ และอภิรดา ธาดาเดช. 2553. แบบจำลองต้นไม้การจำแนกแบบมีเงื่อนไขสำหรับสร้างกรณีทดสอบบนพื้นฐานของแผนภาพกิจกรรมของ UML. การประชุมวิชาการระดับประเทศด้านเทคโนโลยีสารสนเทศ (NCIT2010) ครั้งที่ 3. กรุงเทพมหานคร ประเทศไทย, 28-29 ตุลาคม 2553.

Supaporn Kansomkeat, Pratyanee Thaikerd and Jeff Offutt. 2010. Generating Test Cases from UML Activity Diagrams Using the Condition – Classification Tree Method. 2010 2nd International Conference on Software Technology and Engineering (ICSTE 2010). San Juan, Puerto Rico, USA. October 3–5, 2010. pp 62 - 66.