การสร้างกรณีทดสอบสำหรับการทดสอบระดับรวมหน่วยแบบเพิ่มทีละหน่วย โดยอัตโนมัติจากกรณีทดสอบระดับหน่วย

นายสกรณ์ บุษบง

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
มหาวิทยาลัยเทคโนโลยีสุรนารี
ปีการศึกษา 2556

AUTOMATED INCREMENTAL INTEGRATION TEST CASE GENERATION FROM UNIT TEST CASE

Zagon Budsabong

A Thesis Submitted in Partial Fulfillment of the Requirements for the

Degree of Master of Engineering in Computer Engineering

Suranaree University of Technology

Academic Year 2013

การสร้างกรณีทดสอบสำหรับการทดสอบระดับรวมหน่วยแบบเพิ่มทีละหน่วยโดยอัตโนมัติจากกรณีทดสอบระดับหน่วย

มหาวิทยาลัยเทคโนโลยีสุรนารี อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

คณะกรรมการสอบวิทยานิพนธ์

_____

(รศ. ดร.กิตติศักดิ์ เกิดประสพ)
ประธานกรรมการ

_____

(ผศ. ดร.พิชโยทัย มหัทธนาภิวัฒน์)
กรรมการ (อาจารย์ที่ปรึกษาวิทยานิพนธ์)

_____

(ผศ. ดร.คะชา ชาญศิลป์)
กรรมการ

_____

(ศ. ดร.ชูกิจ ลิมปีจำนงค์)
รองอธิการบดีฝ่ายวิชาการและนวัตกรรม

_____

(รศ. ร.อ. ดร.กนต์ธร ชำนิประศาสน์)
คณบดีสำนักวิชาวิศวกรรมศาสตร์

สกรณ์ บุษบง : การสร้างกรณีทดสอบสำหรับการทดสอบระดับรวมหน่วยแบบเพิ่มทีละหน่วยโดยอัตโนมัติจากกรณีทดสอบระดับหน่วย(AUTOMATED INCREMENTAL INTEGRATION TEST CASE GENERATION FROM UNIT TEST CASE)
อาจารย์ที่ปรึกษา : ผู้ช่วยศาสตราจารย์ ดร. พิชโยทัย มหัทธนาภิวัฒน์, 75 หน้า.

ในการทดสอบซอฟต์แวร์นั้นเป็นกระบวนการที่ประกอบไปด้วยขั้นตอนที่มีความซับซ้อน ใช้เวลาและค่าใช้จ่ายสูงและการทดสอบระดับรวมหน่วยนั้นเป็นการทดสอบที่สำคัญในกระบวนการทดสอบซอฟต์แวร์เนื่องจากเป็นการทดสอบที่สามารถระบุข้อผิดพลาดจากการทำงานระหว่างโมดูล ซึ่งมีความสำคัญและมีผลกระทบต่อคุณภาพของซอฟต์แวร์ที่พัฒนาขึ้นมาอย่างมาก ในกระบวนการสร้างกรณีทดสอบทั่วไปนั้นจะต้องสร้างกรณีทดสอบจากกรณีการทำงาน (Use Case) จึงต้องมีการพิจารณากรณีจำนวนมากเพื่อวิเคราะห์กรณีที่จำเป็นในการสร้างกรณีทดสอบของแต่ละส่วนของซอฟต์แวร์ที่กำลังทำการทดสอบโดยเฉพาะการสร้างกรณีทดสอบระดับรวมหน่วย (Integration Testing) นั้นมักจะมีขั้นตอนที่มีความซับซ้อน เนื่องจากเป็นการทำงานของโมดูลที่มีความสัมพันธ์กันตั้งแต่สองโมดูลขึ้นไป ทำให้ในบางกรณีเกิดความยุ่งยากในกระบวนการสร้างกรณีทดสอบ

งานวิจัยนี้ได้เสนอแนวทางแก้ไขปัญหาความล่าช้าและยุ่งยากของการสร้างกรณีทดสอบระดับรวมหน่วย โดยพัฒนาเครื่องมือที่ใช้ในการสร้างกรณีทดสอบระดับรวมหน่วยอัตโนมัติจากกรณีทดสอบระดับหน่วยโดยใช้วิธีเพิ่มทีละหน่วยจากกรณีทดสอบระดับหน่วย (Unit Test Case) โดยที่งานวิจัยชิ้นนี้มุ่งเน้นไปที่การสร้างความสัมพันธ์ระหว่างโมดูลเพื่อลดความยุ่งยาก ซับซ้อนและเวลาในการสร้างกรณีทดสอบ

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนักศึกษา_____

ปีการศึกษา 2556 ลายมือชื่ออาจารย์ที่ปรึกษา_____

ZAGON BUDSABONG : AUTOMATED INCREMENTAL INTEGRATION TEST CASE GENERATION FROM UNIT TEST CASE.
THESIS ADVISOR : ASST. PROF. DR. PICHAYOTAI MAHATHANAPAWAT, Ph.D., 75 PP.

Software testing is a process comprising the steps that are complex, spend more time and expensive. Integration testing is an important process in software testing because it can identify errors between operations of the modules. This is important and affects the quality of software that developed greatly. In the process of general test case generation, it must generate test cases from Use Case and consider many test cases to analyze necessary cases for each of the software under test. Integration testing generation is usually complex process, because it tests the operation of modules that are related from 2 or above. In some cases, making difficulty in the process of creating test cases.

This thesis proposes how to reduce time and difficulty of creating integration test cases, by development of tools for automated incremental integration test case generation from unit test case and using abstract syntax tree to identify relation between modules. This work focuses on creating the relationship between modules to reduce the complexity and time to generate test cases.

School of Computer Engineering Student's Signature_____

Academic Year 2013 Advisor's Signature_____

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จลุล่วงด้วยดี ผู้วิจัยขอกราบขอบพระคุณบุคคล และกลุ่มบุคคลต่าง ๆ ที่ได้กรุณาให้คำปรึกษา แนะนำ และช่วยเหลือเป็นอย่างดี ทั้งในด้านวิชาการ และด้านการดำเนินงานวิจัยดังต่อไปนี้

ผู้ช่วยศาสตราจารย์ ดร.พิชโยทัย มหัทธนาภิวัฒน์ อาจารย์ที่ปรึกษาวิทยานิพนธ์ที่ให้คำแนะนำปรึกษาในการทำงานวิจัย ช่วยแก้ไขปัญหาและให้กำลังใจแก่ผู้วิจัยตลอดมา รวมทั้งช่วยตรวจทาน และแก้ไขวิทยานิพนธ์เล่มนี้จนเสร็จสมบรูณ์การจัดการรูปแบบ และช่วยตรวจทานความถูกต้องของวิทยานิพนธ์

คุณกัลญา พับโพธิ์ เลขานุการสาขาวิชาวิศวกรรมคอมพิวเตอร์ และคุณดารณี ทิพย์ทอง ผู้ช่วยสอนสาขาวิชาวิศวกรรมคอมพิวเตอร์ ที่ให้ความช่วยเหลือในงานด้านเอกสาร เพื่อนๆพี่ๆ น้องๆ นักศึกษาบัณฑิตสาขาวิชาวิศวกรรมคอมพิวเตอร์ และนักศึกษาบัณฑิตสาขาวิชาเทคโนโลยีสารสนเทศ ทุกท่านที่คอยให้คำปรึกษาและช่วยเหลือมาโดยตลอด

นอกจากนี้ ขอขอบคุณครู อาจารย์ทั้งในอดีตและปัจจุบันที่ให้ความรู้แก่ผู้วิจัยจนประสบความสำเร็จในชีวิต

ขอกราบขอบพระคุณ บิดา มารดา ที่ให้กำเนิด อบรม เลี้ยงดูด้วยความรัก และส่งเสริมการศึกษาเป็นอย่างดีโดยตลอด ทำให้ผู้วิจัยมีความรู้ ความสามารถ มีจิตใจที่เข็มแข็ง รวมทั้งเป็นกำลังใจที่ยิ่งใหญ่แก่ผู้วิจัย จนทำให้ผู้วิจัยประสบความสำเร็จในชีวิตเรื่อยมา

สำหรับคุณงามความดีอันใดที่เกิดจากวิทยานิพนธ์เล่มนี้ ผู้วิจัยขอมอบให้กับบิดา มารดา ซึ่งเป็นที่รักและเคารพยิ่ง ตลอดจนครูอาจารย์ที่เคารพทุกท่าน ที่ได้ประสิทธิ์ประสาทความรู้และถ่ายทอดประสบการณ์ที่ดีให้แก่ผู้วิจัยตลอดมา จนทำให้ประสบความสำเร็จในชีวิตตลอดมา

สกรณ์ บุษบง

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญรูป

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของปัญหาการวิจัย

การพัฒนาซอฟต์แวร์ (Software Development) ในปัจจุบันนั้นมีเป้าหมายสูงสุดคือ การพัฒนาให้ซอฟต์แวร์มีคุณภาพ ทำงานได้ตรงต่อความต้องการและไม่มีข้อผิดพลาดภายในซอฟต์แวร์ จึงได้มีกระบวนการทดสอบซอฟต์แวร์ (Software Testing) เพื่อตรวจสอบความถูกต้องทั้งในแง่ของความถูกต้องของระบวนการทำงานและ ตรงต่อความต้องการของลูกค้า การทดสอบซอฟต์แวร์นั้นมีหลายขั้นตอน ไม่ว่าจะเป็น การทดสอบระดับหน่วย (Unit Testing) การทดสอบระดับรวมหน่วย (Integration Testing) การทดสอบระบบ (System Testing) และการทดสอบการยอมรับ (Acceptance testing) ตามลำดับ ซึ่งทุกๆขั้นตอนนั้นมุ่งเน้นไปที่การระบุหรือค้นหาข้อผิดพลาดของซอฟต์แวร์ที่ซุกซ่อนอยู่ให้ปรากฏออกมาเพื่อทำการแก้ไขต่อไป

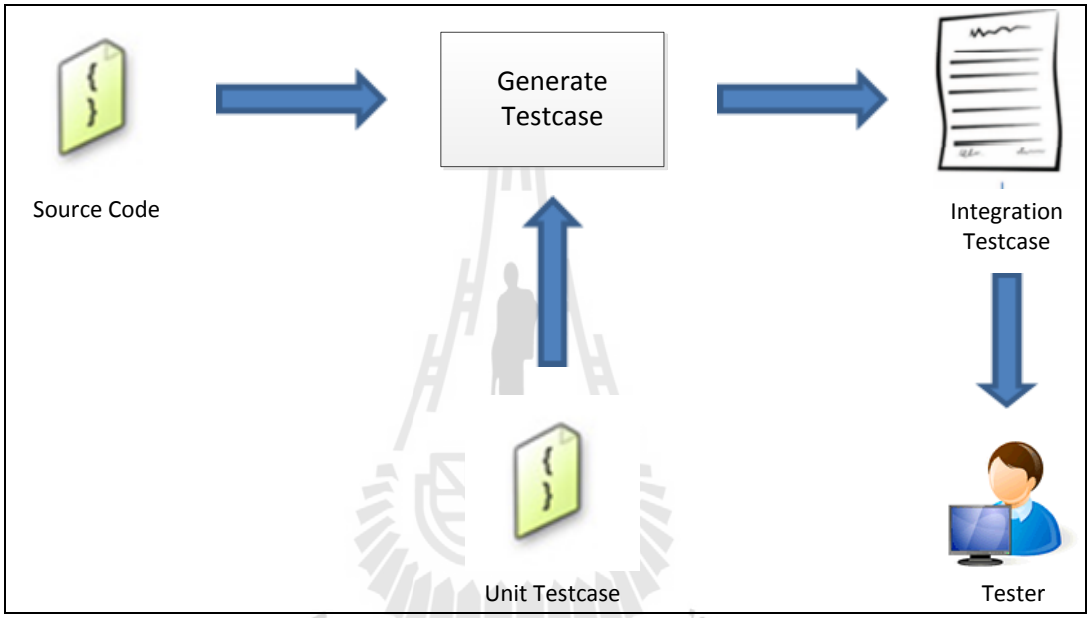
การทดสอบระดับรวมหน่วย (Integration Testing) เป็นขั้นตอนหนึ่งที่มีความสำคัญ ในการทดสอบแต่ละโมดูลที่มีการทำงานร่วมกันเพื่อค้นหาและระบุข้อผิดพลาดที่เกิดขึ้นระหว่างโมดูล ไม่ว่าจะเป็นกรณีที่โมดูลทำงานผิดพลาด หรือกรณีที่โมดูลไม่สามารถทำงานได้เมื่อมีการทำงานร่วมกัน ซึ่งการทดสอบระดับรวมหน่วยนั้นจะต้องสร้างกรณีทดสอบ (Test Case) จำนวนมากเพื่อนำมาใช้ทดสอบกับทุกๆโมดูลที่มีความสัมพันธ์กัน ซึ่งการทดสอบระดับรวมหน่วยนั้นมักจะมีความซับซ้อนมากยิ่งขึ้น เมื่อมีการนำมาใช้ร่วมกับการเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming)

การสร้างกรณีทดสอบ (Test Case) โดยทั่วไปนั้นจะสร้างจากกรณีการทำงาน (Use Case) จึงจำเป็นจะต้องพิจารณาจากกรณีการทำงานจำนวนมากเพื่อสร้างกรณีทดสอบโดยเฉพาะส่วนของการทดสอบระดับรวมหน่วย เนื่องจากมีความสัมพันธ์ของหลายๆกรณีการทำงานในการสร้างกรณีทดสอบเดียว ทำให้กระบวนการสร้างกรณีทดสอบนั้นเป็นกระบวนการที่ต้องใช้เวลาและทรัพยากรเป็นอย่างมาก

หากพิจารณาจากกรณีทดสอบระดับหน่วย (Unit Test Case) ของซอฟต์แวร์ที่พัฒนาขึ้นมานั้น จะเห็นได้ว่ามีข้อมูลที่มีประโยชน์ต่อการนำมาประยุกต์ใช้ร่วมกับกระบวนการสร้างกรณีทดสอบระดับรวมหน่วย เนื่องจากการทดสอบระดับหน่วยนั้นได้มีการตรวจสอบแล้วว่าแต่ละ

โมดูลนั้นสามารถทำงานได้ตรงต่อความต้องการ จึงสามารถนำมาใช้ตรวจสอบในกรณีที่มีโมดูลมีความสัมพันธ์กันจะยังสามารถทำงานได้หรือไม่

ในงานวิจัยนี้ได้พัฒนาเครื่องมือเพื่อช่วยในการสร้างกรณีทดสอบระดับรวมหน่วย (Test Case) แบบอัตโนมัติโดยมุ่งประเด็นไปที่การพิสูจน์ความสัมพันธ์ระหว่างโมดูล เพื่อช่วยให้กระบวนการสร้างกรณีทดสอบระดับรวมหน่วย(Integration Testing) นั้นมีความสะดวกและรวดเร็วยิ่งขึ้น และใช้กรณีทดสอบระดับหน่วยในการสร้างกรณีทดสอบ โดยโครงสร้างจะแสดงในรูปที่ 1.1



รูปที่ 1.1 กรอบแนวคิดงานวิจัยเรื่อง การสร้างกรณีทดสอบของการทดสอบระดับรวมหน่วยอัตโนมัติ

1.2 วัตถุประสงค์ของงานวิจัย

งานวิจัยนี้มีวัตถุประสงค์เพื่อศึกษาและพัฒนาวิธีการสร้างกรณีทดสอบระดับรวมหน่วยแบบอัตมัติโดยมีจุดประสงค์ดังนี้

1.2.1 เพื่อศึกษาและพัฒนากระบวนการแจกแจงความสัมพันธ์ระหว่างโมดูลภายในคลาสจากซอร์สโค้ดภาษาจาวา

1.2.2 เพื่อศึกษาและพัฒนาขั้นตอนและเทคนิคในการสร้างกรณีทดสอบระดับรวมหน่วย (Integration Test Case) จากกรณีทดสอบระดับหน่วย (Unit Test Cases)

1.2.3 เพื่ออำนวยความสะดวกและลดระยะเวลาในกระบวนการสร้างกรณีทดสอบระดับรวมหน่วย

1.2.4 เพื่อนำกรณีทดสอบระดับหน่วยกลับมาใช้ใหม่

1.3 ข้อตกลงเบื้องต้น

งานวิจัยนี้เป็นการศึกษาและพัฒนาระบบเพื่อสร้างกรณีทดสอบสำหรับการทดสอบระดับรวมหน่วยอัตโนมัติ เพื่อสร้างกรณีทดสอบระดับรวมหน่วยจากกรณีทดสอบระดับ หน่วยส่งผลให้การสร้างกรณีทดสอบมีความรวดเร็วขึ้น โดยจะมีคุณสมบัติดังนี้

1. ใช้กรณีทดสอบระดับหน่วยของซอฟต์แวร์เดียวกัน
2. ใช้ซอร์สโค้ดเป็นไฟล์สกุลจาวา
3. โปรแกรมจะสร้างกรณีทดสอบระดับรวมหน่วยจากกรณีทดสอบระดับหน่วย
4. โปรแกรมจะสร้างกรณีทดสอบในรูปแบบ กรณีทดสอบและผลลัพธ์ โดยที่ผลลัพธ์จะแสดงถึงผลที่จะเกิดขึ้นหลังการทดสอบ
5. ใช้วิธีการทดสอบแบบบนลงล่าง (Top-Down Testing)

1.4 ขอบเขตของการวิจัย

งานวิจัยนี้เป็นการศึกษาและพัฒนาเครื่องมือเพื่อใช้ในการสร้างกรณีทดสอบสำหรับการทดสอบระดับรวมหน่วยอัตโนมัติ ซึ่งจะสร้างกรณีทดสอบระดับรวมหน่วยจากกรณีทดสอบระดับหน่วยโดยที่จะนำเข้าข้อมูลทั้งโฟลเดอร์ ซึ่งโฟลเดอร์นี้จะประกอบไปด้วยไฟล์สกุลจาวาของโปรเจคที่จะทำการทดสอบ โดยจะมีขอบเขตดังนี้

1. การสร้างกรณีทดสอบจะมุ่งเน้นไปที่การพิสูจน์ความสัมพันธ์ระหว่างโมดูลและตรวจสอบสถานะการทำงานร่วมกันระหว่างโมดูล
2. เครื่องมือจะรอบรับเฉพาะโมดูลที่ใช้ตัวแปรพื้นฐานของภาษาจาวา
3. การสร้างกรณีทดสอบนี้จะไม่เข้าถึงกระบวนการภายในโมดูล เป็นเพียงการตรวจหาว่ามีการเรียกโมดูลใดอยู่ภายในโมดูลเท่านั้น

งานวิจัยชิ้นนี้จะมีการทดสอบเพื่อตรวจสอบประสิทธิภาพในการสร้างกรณีทดสอบโดยการกำหนดโมดูลในรูปแบบต่างๆ รวมถึง ความสัมพันธ์ของโมดูลในรูปแบบต่างๆเพื่อประเมินความถูกต้องของกรณีทดสอบที่ถูกสร้างขึ้นจากเครื่องมือนี้

1.5 ประโยชน์ที่คาดว่าจะได้รับ

จากการศึกษาและพัฒนางานวิจัยนี้ผู้วิจัยคาดว่าจะเกิดประโยชน์ต่อผู้ใช้ในการใช้งานซอฟต์แวร์เพื่อสร้างกรณีทดสอบระดับรวมหน่วยอัตโนมัติดังนี้

1. อำนวยความสะดวกและลดทรัพยากรในการสร้างกรณีทดสอบระดับรวมหน่วย
2. ผู้ใช้สามารถมองเห็นโครงสร้างของโมดูลที่มีความสัมพันธ์กันได้
3. กรณีทดสอบที่สร้างขึ้นมานั้นจะตรงต่อวัตถุประสงค์ของซอฟต์แวร์นั้นๆ
4. เป็นเครื่องมือต้นแบบ สามารถนำไปพัฒนาต่อได้

บทที่ 2
ปริทัศน์วรรณกรรมและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงงานวิจัยที่เกี่ยวข้องโดยมีรายละเอียดของการทดสอบซอฟต์แวร์ (Software Testing) ระดับของการทดสอบซอฟต์แวร์ (Software Testing Levels) การเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming) และงานวิจัยที่เกี่ยวข้อง

2.1 การทดสอบซอฟต์แวร์ (Software Testing)

การทดสอบซอฟต์แวร์ (Software Testing) เป็นขั้นตอนที่มีความสำคัญและเป็นกระบวนการที่จะต้องทำตลอดทั้งกระบวนการพัฒนาซอฟต์แวร์ (Software Development) โดยการทดสอบซอฟต์แวร์นั้นจะมุ่งเน้นไปที่การประเมินซอฟต์แวร์ (Software) หรือระบบ (System) เพื่อให้ตรงต่อความต้องการและระบุข้อผิดพลาดที่เกิดขึ้นเพื่อทำการแก้ไขและได้ซอฟต์แวร์หรือระบบที่มีคุณภาพตามที่ต้องการ

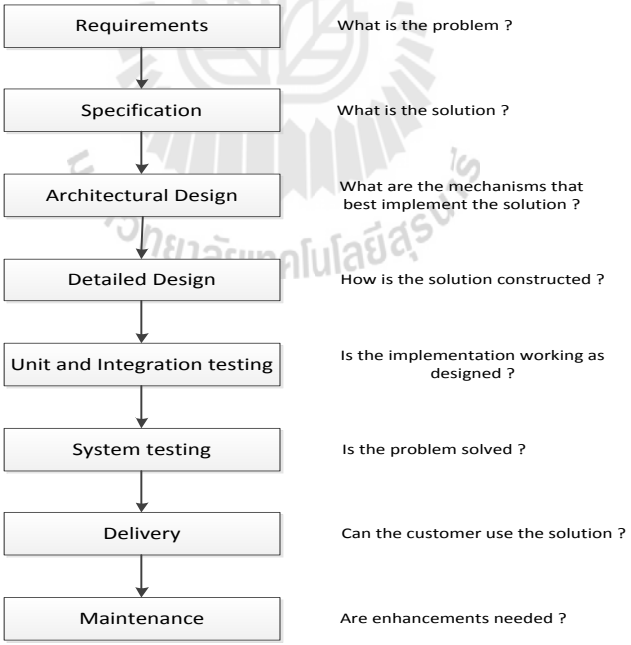
การทดสอบซอฟต์แวร์นั้นขึ้นอยู่กับวิธีการที่ใช้ทดสอบซึ่งสามารถดำเนินการได้ตลอดทั้งกระบวนการพัฒนาซอฟต์แวร์ ซึ่งแต่เดิมนั้นการทดสอบจะเกิดเมื่อขั้นตอนการเขียนโปรแกรมเสร็จเรียบร้อยแล้ว แต่ในวิธีการแบบคล่องตัว (Agile Method) การทดสอบจะดำเนินไปพร้อมกับกระบวนการพัฒนา ดังนั้น วิธีการแบบคล่องตัว เป็นวิธีในการพัฒนาซอฟท์แวร์ โดยมีหลักการในการพัฒนาระบบที่เน้นการทำงานที่รวดเร็ว มีการเตรียมพร้อมที่จะตอบสนองต่อความเปลี่ยนแปลงที่อาจเกิดขึ้นได้ตลอดเวลา โดยเฉพาะความเปลี่ยนแปลงที่เกิดจากความต้องการของผู้ใช้ วิธีพัฒนาแบบนี้ถือเป็นการพัฒนาแบบทำซ้ำ ที่จะต้องมีการพบปะสนทนากับผู้ใช้อยู่ตลอดเวลา และในขณะที่พบปะกันนั้นก็จะถือเป็นช่วงระยะเวลาของการส่งงานไปในตัวด้วย การส่งงานแต่ละครั้งของวิธีการพัฒนาแบบนี้ โดยส่วนใหญ่จะเป็นการพัฒนาเฉพาะส่วนย่อย ๆ จากนั้นจะทยอยส่งให้กับผู้ใช้ เมื่อผู้ใช้ได้ทดสอบหรือประเมินระบบแล้ว ถ้าต้องการปรับเปลี่ยนตรงส่วนใดก็สามารถทำได้โดยที่ไม่ต้องรื้อระบบใหม่ทั้งหมด

หลังจากได้ทำการทดสอบความถูกต้องของแต่ละโมดูลและพบข้อผิดพลาดแล้ว การแก้ปัญหาหรือแก้ไขข้อบกพร่องนั้นอาจมีหลายวิธีที่สามารถใช้ในการแก้ไขได้ จึงมีการรวบรวมวิธีการดังกล่าวเพื่อนำไปปรึกษาในทีมพัฒนาและทีมทดสอบเพื่อหาวิธีการที่ดีที่สุด และนำเทคนิคที่เหมาะสมนั้นมาใช้ในการทดสอบ เพื่อทำการแก้ไขต่อไป เมื่อตราจสอบถูกต้องของแต่ละโมดูลแล้ว

ขั้นตอนต่อไปจะนำแต่ละโมดูลย่อยต่างๆมารวมเป็นระบบใหญ่ การทดสอบนั้นจะกระทำทั้งระบบ โดยเลือกวิธีที่เหมาะสมโดยเปรียบเทียบถึงจุดเด่นและจุดด้อยของแต่ละเทคนิค และจะนำเทคนิคที่เหมาะสมและมีประสิทธิภาพที่สุดมาใช้ในการทดสอบ นอกจากนั้นผู้ทดสอบยังสามารถนำเครื่องมืออัตโนมัติมาช่วยในการทดสอบได้อีกด้วย

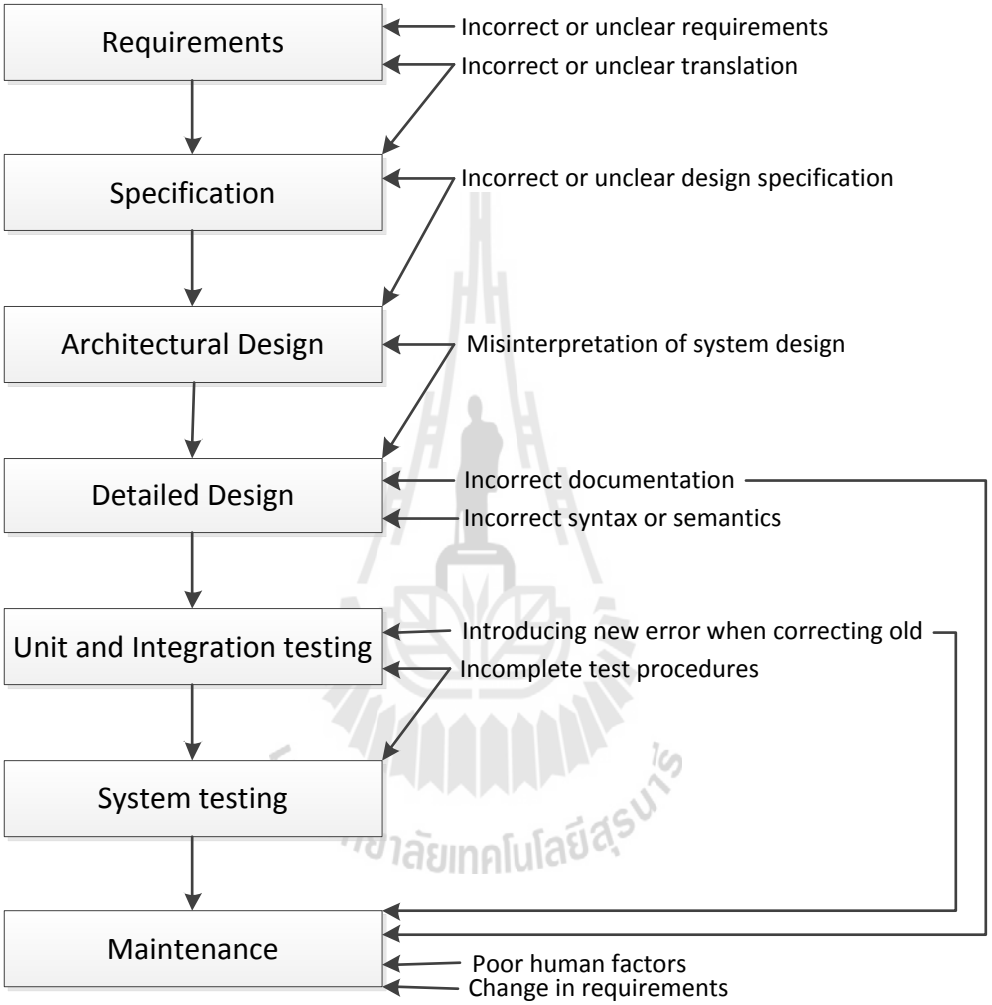
การเขียนโปรแกรมที่ไม่มีความผิดพลาดเลยนั้นเป็นไปได้ยากมาก แม้ว่าโปรแกรมเมอร์จะมีความเชี่ยวชาญและมีประสบการณ์สูง ความผิดพลาดที่เกิดขึ้นนั้นถือเป็นเรื่องปกติเพียงแต่จะมีมากหรือน้อยเท่านั้น เนื่องจากในระบบงานนั้นมีปัจจัยมากมายที่จะทำให้เกิดข้อผิดพลาดได้

ความผิดพลาดนั้นสามารถเกิดขึ้นได้ทุกๆระยะของขั้นตอนการพัฒนาซอฟต์แวร์แสดงในรูปที่ 2.1 ความผิดพลาดอาจเกิดจากลูกค้าไม่แน่ใจถึงความต้องการของตนเอง หรือการตีความหมายความต้องการของลูกค้าผิดพลาด ซึ่งทั้งสองกรณีนั้นจะทำให้ส่งผลถึงระบบ ทำให้ไม่สามารถทำตามความต้องการของลูกค้าได้ การสื่อสารของทีมงานนั้นก็อาจเป็นสาเหตุของความผิดพลาดได้ เนื่องจากการตีความหมายที่ผิดพลาดตั้งแต่ต้นส่งผลกระทบต่อกระบวนการการออกแบบระบบ ทำให้การออกแบบระบบนั้นไม่ถูกต้อง



รูปที่ 2.1 The System Development Process (http://e-book.ram.edu,2556)

ความผิดพลาดทั้งหมดนั้นสามารถเกิดขึ้นได้ทั้งจากลูกค้า นักออกแบบระบบ นักออกแบบโปรแกรม โปรแกรมเมอร์ ทีมงานทดสอบ รวมถึงทีมงานในการบำรุงรักษาระบบอีกด้วย ดังแสดงในรูปที่ 2.2 ซึ่งจะแสดงถึงข้อผิดพลาดที่อาจเกิดขึ้นในการพัฒนาซอฟต์แวร์



รูปที่ 2.2 ข้อผิดพลาดที่เกิดขึ้นในการพัฒนาซอฟต์แวร์ (http://e-book.ram.edu,2556)

2.1.1 วัตถุประสงค์ของการทดสอบ

วัตถุประสงค์หลักของการทดสอบคือการค้นหาข้อผิดพลาดที่อยู่ภายในซอฟต์แวร์เพื่อระบุและแก้ไข การทดสอบนั้น เป็นไปไม่ได้เลยที่จะสร้างโมดูล (Module) เพื่อนำมาทดสอบได้ทุกส่วนของซอฟต์แวร์ แต่สามารถที่จะทดสอบทีละส่วนของซอฟต์แวร์ได้

การทดสอบซอฟต์แวร์จะใช้หลักการที่เรียกว่า การทวนสอบและทดสอบเพื่อรับรองผล (Verification and Validation)ในการประเมินซอฟต์แวร์ที่ทำการทดสอบมีรายละเอียดดังนี้

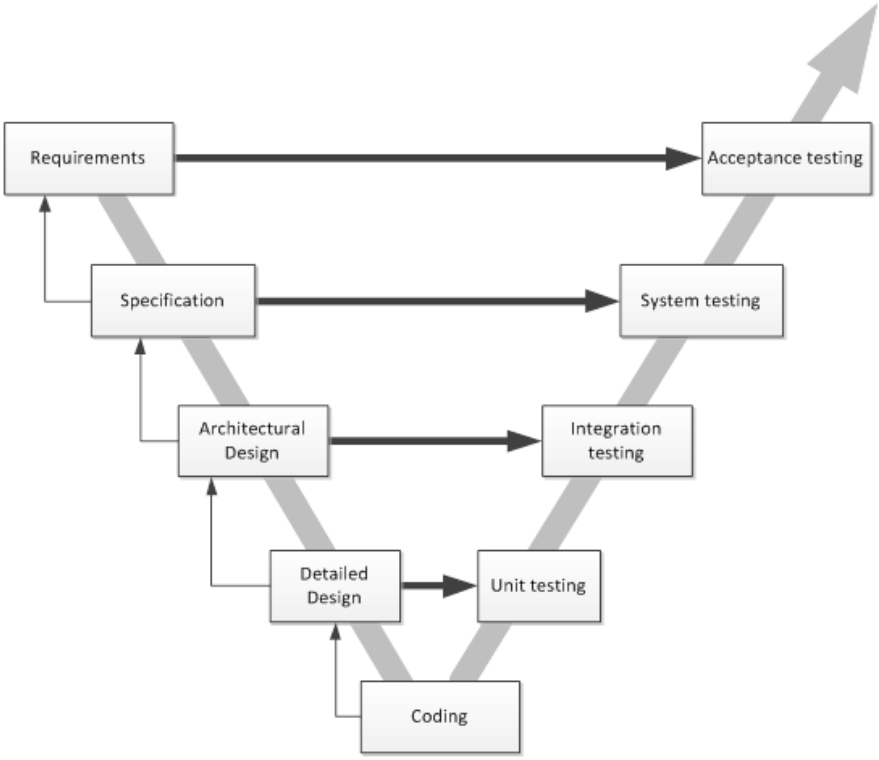
1) การทวนสอบ (Verification) คือกระบวนการประเมินว่าซอฟต์แวร์นั้นทำงานได้อย่างถูกต้องตามขั้นตอนกระบวนการทำงานหรือไม่ ซึ่งตรงกับทฤษฎีการทดสอบแบบเป็นระดับ โดยมีการทดสอบหน่วย (Unit Testing) การทดสอบรวม (Integration Testing) และการทดสอบระบบ (System Testing)

2) ทดสอบเพื่อรับรองผล (Validation) คือกระบวนการประเมินว่าซอฟต์แวร์ (Software) นั้นตรงต่อความต้องการของผู้ใช้ (User Requirement) มักทำในกระบวนการช่วงหลังของการพัฒนาซึ่งตรงกับการทดสอบระดับ การทดสอบการยอมรับ (Acceptance Testing)

2.2 ระดับการทดสอบซอฟต์แวร์ (Software Testing Levels)

ในการพัฒนาซอฟต์แวร์เพื่อให้ได้ซอฟต์แวร์ที่มีคุณภาพ จะมีการทดสอบหลายระดับที่ทำได้อย่างละเอียดและมีประสิทธิภาพ ซึ่งจะเป็นการทดสอบที่เริ่มจากส่วนที่น้อยที่สุดและขยายขึ้นเรื่อยๆจนกลายเป็นการทดสอบทั้งระบบ ไปจนถึงการทดสอบการยอมรับจากลูกค้า

วีโมเดล (V-Model) เป็นแบบจำลองที่แสดงความสัมพันธ์ระหว่างวงจรการพัฒนา (Development life cycle) และวงจรการทดสอบ (Testing life cycle) โดยที่แต่ละกระบวนการพัฒนาซอฟต์แวร์จะมีการจัดวางโครงสร้างของกรณีทดสอบไว้ ซึ่งแสดงในรูปที่ 2.3



รูปที่ 2.3 V-Model (http://kb.tsu.ac.th,2556)

การทดสอบซอฟต์แวร์ที่พัฒนาขึ้นมามีวัตถุประสงค์คือการแก้ไขข้อผิดพลาดต่างๆที่เกิดขึ้นในระหว่างการพัฒนา ตรวจสอบคุณสมบัติและทดสอบความสามารถในการทำงานในด้านต่างๆของซอฟต์แวร์ให้เป็นไปตามรูปแบบและข้อกำหนดต่างๆที่ได้ออกแบบไว้ ขั้นตอนการทดสอบซอฟต์แวร์นั้นประกอบไปด้วย

1. การทดสอบระดับหน่วย (Unit Testing)

เป็นการทดสอบระดับที่ย่อยที่สุดของซอฟต์แวร์ ซึ่งเป็นขั้นตอนแรกของการทดสอบหลังจากที่ผู้พัฒนาได้เขียนโปรแกรมเรียบร้อยแล้ว เพื่อประเมินหาข้อผิดพลาดในภายในคำสั่งอาจเป็นรูปแบบภาษา ความหมาย สูตร การคำนวณ ลำดับของการทำงาน โดยทำการเปรียบเทียบกับเอกสารที่ได้ระบุความต้องการ,เอกสารการออกแบบระบบและเอกสารการออกแบบโปรแกรม การทดสอบในขั้นแรกจะทำการกำหนดกรณีทดสอบขึ้นมาหลายๆชุดแล้วนำมาทดสอบเพื่อตรวจดูผลลัพธ์ว่าได้ตรงตามความต้องการหรือไม่

การทดสอบระดับหน่วยนี้จะทำการทดสอบในด้านต่างๆของแต่ละโมดูล ซึ่งจะสามารถทดสอบหลายๆโมดูลไปพร้อมๆกันได้ โดยจะต้องทดสอบดังนี้

1. ตัวประสาน (Interface) คือการกำหนดความสามารถของวัตถุในการเขียนโปรแกรมเชิงวัตถุซึ่ง คลาส ใดที่ได้ implement interface ถือว่าการระบุว่าคลาสนั้นจะมีความสามารถตามที่ interface กำหนดไว้ ดังนั้นการติดต่อกับวัตถุที่สร้างขึ้นจาก คลาส ดังกล่าวต้องผ่าน interface เท่านั้น

2. โครงสร้างข้อมูล (Data Structure) โครงสร้างข้อมูลในระดับหน่วยก็คือโครงสร้างข้อมูลท้องถิ่น (Local) ที่มีขอบเขตอยู่ภายในหน่วยย่อย เพื่อดูว่าข้อมูลที่ถูกจัดเก็บชั่วคราว มีความสมบูรณ์หรือถูกต้องในระหว่างการทำงานของอัลกอริธึมหรือไม่

3. เงื่อนไขขอบเขต (Boundary Condition) คือ ขอบเขตค่าของข้อมูลที่โปรแกรมต้องประมวลผล โปรแกรมจะต้องทำงานภายใต้ขอบเขตข้อมูลที่กำหนดจึงจะถูกต้อง หาก ข้อมูลมีค่าต่ำหรือสูงกว่าขอบเขตที่กำหนด โปรแกรมที่ดีจะต้องส่งไปยังเส้นทางการทำงานอื่นเพื่อจัดการกับความผิดพลาดนี้ต่อไป หากต้องการทดสอบให้พบข้อผิดพลาดควรป้อนข้อมูลที่มีค่าต่ำหรือสูงกว่าขอบเขต ก็จะทราบว่าโปรแกรมนั้นมีคุณภาพหรือไม่

4. เส้นทางการประมวลผลอิสระ (Independent Process Path) คือเส้นทางการทำงานที่แตกต่างกันตามเงื่อนไขที่ได้รับ เมื่อนำข้อมูลเข้าสู่โปรแกรมจะต้องมีเส้นทางอย่างน้อย 1 เส้นทาง ที่จะถูกใช้งาน

5. เส้นทางการประมวลผลข้อผิดพลาดและการแสดงข้อผิดพลาด เพื่อแจ้งเตือนผู้ใช้และให้คำแนะนำเพื่อทำงานต่อไป การทดสอบเส้นทางดังกล่าวก็จำเป็นสำหรับคุณภาพของโปรแกรม และเป็นอีกหนึ่งเส้นทางที่จะทำให้พบข้อผิดพลาดได้มากที่สุด

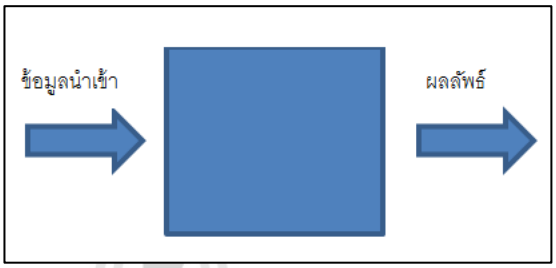
การทดสอบระดับหน่วยนั้นจะต้องเริ่มจาการออกแบบกรณีการทดสอบขึ้นมาก่อน ซึ่งกรณีทดสอบจะต้องสามารถค้นพบข้อผิดพลาดของโปรแกรมให้ได้มากที่สุด สำหรับการทดสอบระดับหน่วยมีให้เลือกใช้ 2 วิธีคือ กล่องขาว (White-Box Testing) กล่องดำ (Black-Box Testing) (http://e-book.ram.edu,2556)

1. การทดสอบแบบกล่องดำ (Black-Box Testing) คือการทดสอบโดยที่ไม่สนใจกระบวนการทำงานภายในระบบหรือโปรแกรมโดยมุ่งเน้นไปที่ผลลัพธ์ที่ได้ออกมาจากแต่ละโมดูลของระบบหรือโปรแกรม เพื่อให้แน่ใจว่าระบบหรือซอฟต์แวร์ทำงานได้ถูกต้องตามที่กำหนดไว้ได้หรือไม่ จุดมุ่งหมายของการ

ทดสอบแบบกล่องดำคือ การทดสอบประสิทธิภาพของซอฟต์แวร์และเงื่อนไขขอบเขตของข้อมูลนำเข้าด้วย

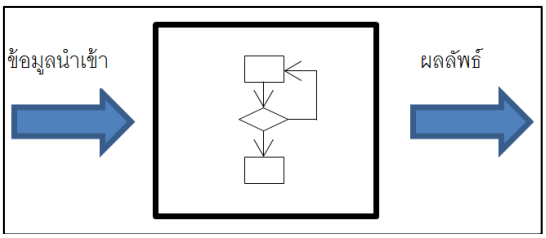
ข้อผิดพลาดที่สามารถพบได้จากการทดสอบแบบกล่องดำ มีดังนี้

- กระบวนการทำงานที่ผิดพลาด หรือขาดหายไป
- ข้อผิดพลาดของส่วนประสาน (Interface) กับระบบอื่นๆ
- ข้อผิดพลาดของการทำงานต่อหรือหยุดการทำงาน
- ข้อผิดพลาดจากการประมวลผล เป็นต้น



รูปที่ 2.4 รูปแสดงการทดสอบแบบกล่องดำ (Black-Box Testing)
(http://e-book.ram.edu,2556)

2. การทดสอบแบบกล่องขาว (White-Box Testing) คือการทดสอบโดยที่จะคำนึงถึงโครงสร้างภายในระบบหรือโปรแกรม เป็นการทดสอบย่อยที่ละโมดูลหรือการทดสอบการทำงานระหว่างโมดูลและมักทำการทดสอบโดยผู้พัฒนา (Developer) โดยใช้ผลลัพธ์ทางตรรกะที่ได้จากการทำงานของซอฟต์แวร์เป็นตัวชี้วัด ซึ่งการทดสอบแบบกล่องขาวนั้นจะต้องสร้างผังงาน (Flow Chart) เพื่อกำหนดเส้นทางทุกๆเส้นทางที่ผ่านการทดสอบอย่างน้อย 1 ครั้ง การทดสอบแบบกล่องขาวจะต้องทดสอบสิ่งต่อไปนี้
   - ทุกๆเส้นทางในกระบวนการจะต้องทำงานได้ถูกต้องสมบูรณ์
   - ทดสอบการตัดสินใจทางตรรกะ ต้องทดสอบ ทั้งค่าที่เป็นจริงและเท็จ
   - ทดสอบการทำงานภายในลูป (Loop) ทุกลูป ตามจำนวนครั้งของการวนรอบ
   - ทดสอบโครงสร้างของข้อมูลภายในให้ถูกต้อง ก่อนที่จะส่งไปประมวลผลต่อไป



รูปที่ 2.5 รูปแสดงการทดสอบแบบกล่องขาว (White-Box Testing)
(http://e-book.ram.edu,2556)

2. การทดสอบระดับรวมหน่วย (Integration Testing)

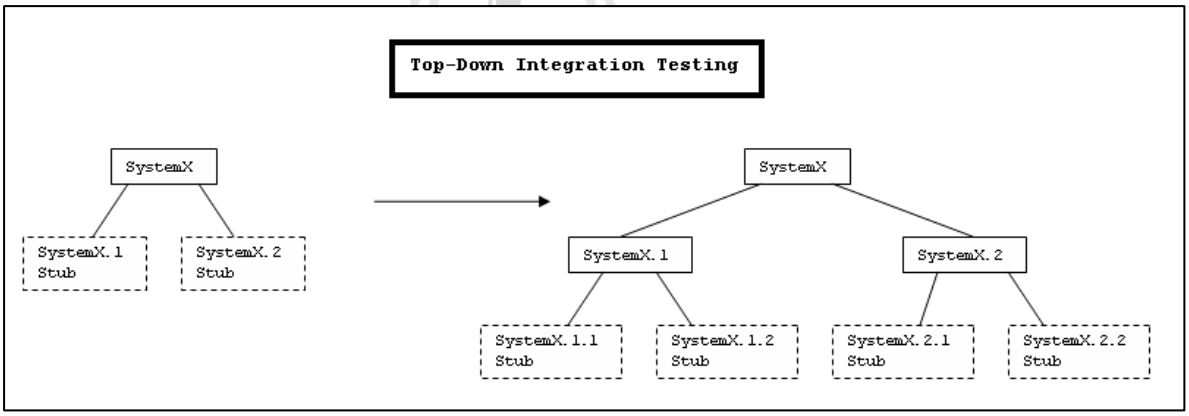
เป็นการทดสอบที่จะตรวจสอบการทำงานระหว่างกลุ่มของโปรแกรมหรือโมดูลย่อยต่างๆที่ทำงานร่วมกัน เพื่อค้นหาข้อผิดพลาดที่อาจเกิดขึ้นได้ แม้ว่าแต่ละโมดูลจะผ่านการตรวจสอบมาแล้วก็ตาม เมื่อแต่ละโมดูลนั้นมีการทำงานร่วมกันกับโมดูลอื่นก็อาจเกิดข้อผิดพลาดบางอย่างขึ้นได้ จึงได้มีการทดสอบระดับรวมหน่วยขึ้นมาเพื่อทดสอบส่วนประสานการทำงานระหว่างโมดูล

การทดสอบระดับรวมหน่วยนั้นสามารถกระทำได้ใน 2 ลักษณะ คือ แบบรวมหน่วยทั้งหมดในครั้งเดียว (Big Bang) และแบบเพิ่มทีละหน่วยหรือโมดูล (Incremental) แต่การทดสอบแบบรวมหน่วยทั้งหมดในครั้งเดียวนั้นมีความเสี่ยงต่อการเกิดข้อผิดพลาดสูงมาก เนื่องจากการทดสอบทั้งหมดจะถูกทดสอบเพียงครั้งเดียวหลังจากการรวมแต่ละโมดูลเข้าด้วยกันแล้ว ดังนั้นการทดสอบแบบรวมหน่วยทั้งหมดจึงไม่ค่อยเป็นที่นิยมเท่า การทดสอบแบบเพิ่มทีละหน่วยหรือโมดูล เนื่องจาการทดสอบแบบเพิ่มที่ละหน่วยหรือโมดูลนั้นจะทำการทดสอบทุกครั้งที่มีการเพิ่มโมดูลเข้าไปร่วมการทำงาน ซึ่งการทดสอบแบบเพิ่มทีละหน่วย (Incremental Testing) หรือโมดูลนั้นมี 2 วิธีให้เลือกใช้คือ การทดสอบแบบเพิ่มโมดูล จากบนลงล่าง (Top-down Approach) และการทดสอบแบบเพิ่มโมดูลจากล่างขึ้นบน (Bottom-up Approach)

1. การทดสอบแบบเพิ่มโมดูล จากบนลงล่าง (Top-down Approach)

เป็นการทดสอบโดยเพิ่มทีละโมดูลจากบนลงล่าง (Top-down Approach)ตามลำดับโครงสร้างซึ่งก็คือโมดูลระดับที่สูงกว่าจะเรียกใช้โมดูลระดับต่ำกว่าในรูปแบบของโมดูลหลักและโมดูลย่อย โดยมีหลักการดังนี้

1. กระบวนการทำงานใดๆที่จะทดสอบจะต้องมีโมดูลหลัก เพื่อรับข้อมูลแล้วส่งผ่านไปยังโมดูลทดสอบเรียกโมดูลตัวขับ (Driver) หากในขณะที่ยังไม่มีโมดูลหลักจะต้องเขียนโปรแกรมขึ้นมาเพื่อเป็นตัวแทนของโมดูลหลักเพื่อทำหน้าที่เป็นโมดูลตัวขับ
2. เช่นเดียวกับข้อที่ 1 หากโมดูลที่จะถูกทดสอบ ต้องการโมดูลย่อยจึงจะสมบูรณ์ แต่ในขณะนั้นโมดูลย่อยยังสร้างไม่เสร็จ ทีมงานจะต้องสร้าง โมดูลตัวต้นขั้ว (Stub Module) ขึ้นมาแทนเพื่อทดสอบก่อน เมื่อโมดูลจริงแล้วเสร็จ จึงนำมาประสานแทนโมดูลตัวต้นขั้ว
3. การทดสอบจัดทำขึ้นทุกครั้งที่มีการเพิ่มโมดูล จะสังเกตได้ว่าวิธีการแบบ บนลงล่างนั้นจะเริ่มจากระดับบนก่อนแล้วไปสิ้นสุดที่ระดับล่างสุด โดยเริ่มต้นจากโมดูลที่อยู่ระดับสูงซึ่งมันจะเป็นส่วนเชื่อมต่อต่างๆ

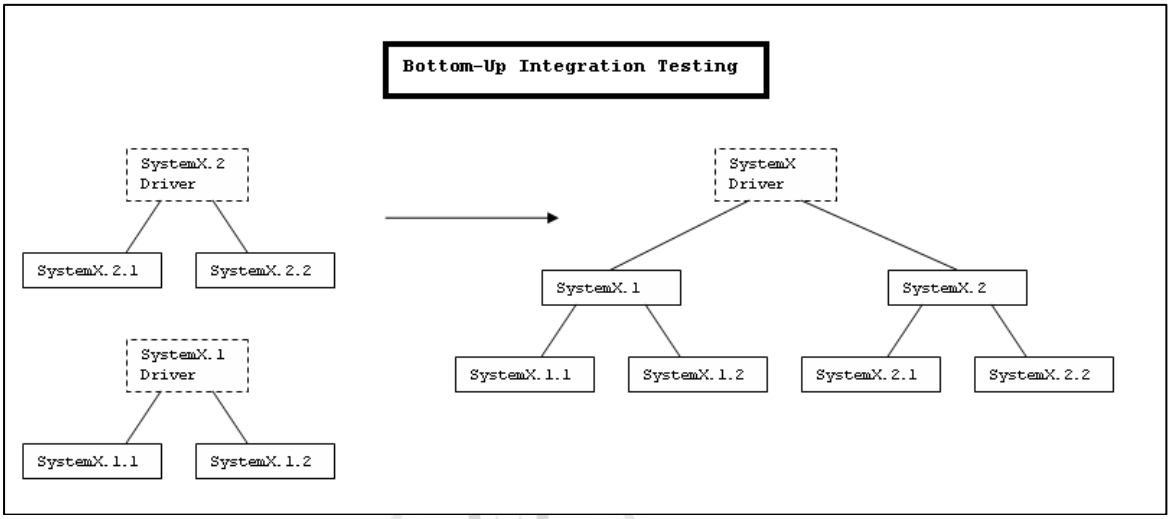


รูปที่ 2.6 Top-Down Integration Testing (http://sce.uhcl.edu,2556)

2.การทดสอบแบบเพิ่มโมดูลจากล่างขึ้นบน (Bottom-up Approach)

จะทดสอบโดยเริ่มจากโมดูลล่างสุดก่อน (จึงไม่จำเป็นต้องใช้ Stub Module) โดยการรวมโมดูลระดับล่างเข้าด้วยกันเป็น “คลัสเตอร์ (Cluster)” เพื่อทดสอบการทำงานรวม การแบ่งคลัสเตอร์จะจำแนกตามหน้าที่การทำงานให้กับฟังก์ชันเดียวกันหรือใกล้เคียง ดังนั้น ทีมงานจึงต้องเขียนโปรแกรมโมดูลตัวขับ ขึ้นมาเพื่อมาทดสอบการทำงานของ

โมดูลระดับล่าง เมื่อทดสอบเสร็จเรียบร้อยแล้ว จึงถอดโมดูลตัวขับออก แล้วแทนที่ด้วยคลัสเตอร์ใหม่ที่เพิ่มเข้ามา และเริ่มทำการทดสอบรวมกับคลัสเตอร์เก่าอีกครั้ง ทำเช่นนี้ไปเรื่อยๆ จนกว่าจะครบทุกคลัสเตอร์ หากลองสังเกตจะพบว่าเครื่องมือที่ช่วยแบ่งคลัสเตอร์ได้ดีที่สุดคือ “Structure Chart” ที่ได้จากระยะการออกแบบโปรแกรมนั่นเอง



รูปที่ 2.7 Bottom-Up Integration Testing (http://sce.uhcl.edu,2556)

การเลือกวิธีการทดสอบระดับรวมหน่วย ขึ้นอยู่กับโครงสร้างควบคุมการทำงานของระบบ หากเป็นระบบที่ควบคุมและตัดสินใจทำงานจากระดับบน ทีมงานสามารถใช้ การทดสอบแบบเพิ่มโมดูลจากบนลงล่างได้ แต่หากเป็นระบบที่การควบคุมขึ้นอยู่กับเงื่อนไขทางธุรกิจ เช่น ระบบประมวลผลรายการข้อมูล ควรเลือกใช้วิธี การทดสอบแบบเพิ่มโมดูลจากล่างขึ้นบนเป็นต้น

การทำงานร่วมกันของระบบเชิงวัตถุนั้นเป็นงานที่มีความซับซ้อน ซึ่งหนึ่งในจุดแข็งของการเขียนโปรแกรมเชิงวัตถุก็คือความสามารถในการสร้างบล็อกที่สามารถจัดการได้อย่างอิสระ ซึ่งสามารถนำมารวมกันเพื่อสร้างเป็นระบบทั้งระบบได้ โปรแกรมเชิงวัตถุที่ถูกออกแบบมาอย่างดีนั้นจะประกอบไปด้วยกลุ่มของบล็อกที่ประกอบกันอย่างง่ายๆ ดังนั้น ความซับซ้อนของระบบเชิงวัตถุก็จะย้ายจากตัวโมดูลไปอยู่ที่การเชื่อมโยงระหว่างโมดูล เช่น ถ้าเป็นภาษาแบบดังเดิมนั้น 40% ของข้อผิดพลาดจะเชื่อมโยงไปที่ปัญหาการรวมหน่วย ซึ่งเราสามารถคาดคะเนได้เลยว่าอัตราข้อผิดพลาดนั้นแนวโน้มที่จะเพิ่มขึ้นในโปรแกรมเชิงวัตถุ

การทดสอบระดับรวมหน่วยนั้นเป็นกิจกรรมที่จะทำการทดสอบ ปฏิสัมพันธ์ระหว่างแต่ละยูนิต ในขณะที่ซอฟแวร์แบบดั้งเดิมนั้นมักจะพิจารณาเพียงกระบวนการเดียวต่อหนึ่งหน่วย ซึ่งความหมายของส่วนประกอบที่เป็นยูนิตนั้นสำหรับระบบแบบเชิงวัตถุนั้นไม่ได้ตรงไปตรงมาแบบระบบดั้งเดิม

โดยที่คลาสนั้นถือได้ว่าเป็นยูนิตพื้นฐานของระบบแบบเชิงวัตถุ การทดสอบระดับรวมหน่วยแบบเชิงวัตถุนั้นมีจุดมุ่งหมายที่จะตรวจสอบว่าคลาสนั้นสามารถทำงานร่วมกันได้ตามที่คาดหมายไว้ ซึ่งกิจกรรมดังกล่าวนั้นเราเรียกว่า interclass level testing.

โดยทั่วไปแล้วเป็นไปได้ที่จะต้องพิจารณาความแตกต่างและความซับซ้อนที่มีอยู่มากสำหรับการทำงานร่วมกันของโปรแกรมเชิงวัตถุ เนื่องจากโครงสร้างที่เป็นแบบเฉพาะของระบบเชิงวัตถุ ความแตกต่างเล็กน้อยระหว่างการทดสอบ intraclass และ interclass อาจจะขาดบางสิ่งที่ที่จำเป็นสำหรับตัวแทนที่มีความแม่นยำที่จะระบุความแตกต่างของมุมมองในการรวมของระบบ

ความแตกต่างของการจำแนกระดับของการทดสอบได้มีการพูดถึงในงานวิจัยในช่วงที่ผ่านมา และเงื่อนไขเดียวกันนั้นได้ถูกนำมาใช้สำหรับการระบุแนวคิดที่แตกต่างกันในบริบทที่ต่างกันเพื่อหลีกเลี่ยงการเข้าใจผิดที่สามารถเกิดได้ง่ายนั้น ในส่วนที่เหลือจึงจะขอนิยามคำต่อไปนี้

กลยุทธ์การทดสอบแบบบนลงล่างและล่างขึ้นบนนั้นยังคงสามารถใช้ได้กับระบบเชิงวัตถุในกรณีที่เราสามารถอ้างอิงความพึ่งพากันในหมู่คลาสได้อย่างถูกต้อง ซึ่งจะมีความแตกต่างจากภาษาแบบดั้งเดิมที่สามารถระบุความสัมพันธ์ของการใช้งานระหว่างโมดูลได้แบบตรงไปตรงมา ระบบเชิงวัตถุนั้นมีลักษณะของความสัมพันธ์ที่แตกต่างกันซึ่งจะถูกเก็บไว้ระหว่างคลาส ความสัมพันธ์เหล่านี้จำเป็นที่จะต้องมีการพิจารณา และการมีปฏิสัมพันธ์ที่มีความซับซ้อนมากๆนั้นจำเป็นต้องมีการพิจารณาอย่างละเอียด โดยคำนึงถึงกรณีแบบดั้งเดิม คลาสหนึ่งๆนั้นจะขึ้นอยู่กับคลาสอื่นแม้ว่าคลาสนั้นจะไม่ถูกเรียกใช้ก็ตาม

ไม่ว่าจะเลือกวิธีการใดก็ตาม หลังจากการทดสอบระดับรวมหน่วยเสร็จสิ้นแล้ว ทีมงานอาจต้องใช้วิธีการทดสอบที่เรียกว่า “Regression Testing” เพื่อทดสอบซ้ำอีกครั้ง ทั้งนี้ก็เพื่อป้องกันปัญหาที่อาจเกิดขึ้นโดยไม่ได้คาดการณ์ไว้ล่วงหน้า เนื่องจากบางครั้งเมื่อถึงขั้นตอนนี้ ความต้องการบางอย่างของลูกค้าอาจเปลี่ยนแปลง ทำให้ต้องมีการเพิ่มเติมโมดูลหลายโมดูล ซึ่งอาจทำให้เกิดปัญหาขึ้นได้ จึงจำเป็นต้องใช้การทดสอบแบบ Regression เพื่อทดสอบการทำงานของโมดูลซ้ำอีกครั้งหนึ่ง โดยสามารถทดสอบด้วยกรณีทดสอบชุดเดิมทั้งหมด แต่เพื่อการประหยัดเวลา ทีมงานสามรถทดสอบเฉพาะส่วนที่เพิ่มเติมได้ อย่างไรก็ตาม หากต้องการทดสอบซ้ำทั้งหมดอีกครั้ง ทีมงานสามารถใช้เครื่องมือเรียกว่า “เครื่องมือบันทึกและเล่นซ้ำ (Capture/Playback Tools)” จะทำ

ให้การดักจับกรณีทดสอบ และเปรียบเทียบผลการทดสอบรวดเร็วขึ้นและสามารถทดสอบซ้ำได้ด้วย

2.3 การเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming)

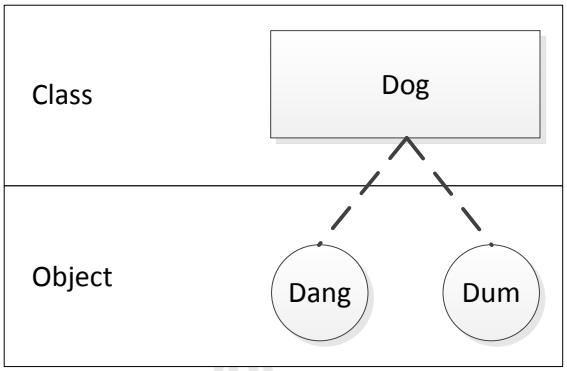
การเขียนโปรแกรมเชิงวัตถุ เป็นรูปแบบการเขียนโปรแกรมที่ใช้กันอย่างแพร่หลาย โดยอาศัยแนวคิดที่ว่า วัตถุชิ้นหนึ่งมีความสามารถในการปกป้องข้อมูลของตน และยังสามารถสืบทอดคุณสมบัติต่างๆของตนได้ด้วย การเขียนโปรแกรมชิงวัตถุนั้นจะให้ความสำคัญกับวัตถุ (Object) โดยจะมองส่วนต่างๆภายในโปรแกรมว่าเป็นวัตถุ ซึ่งวัตถุนั้นจะประกอบไปด้วยคุณสมบัติ (Property) ซึ่งจะสามารถอธิบายได้ว่าวัตถุนี้คืออะไรและวิธีการ (Method) ซึ่งจะอธิบายพฤติกรรมของวัตถุนั้นว่าสามารถทำอะไรได้บ้าง เป็นเอกลักษณ์เฉพาะตัวแฝงอยู่ด้วย อีกทั้งยังสามารถสืบทอดเพื่อสร้างเป็นวัตถุใหม่ได้อีกด้วย จึงมีความแตกต่างจากแนวคิดการเขียนโปรแกรมเชิงกระบวนการ (Procedural Programming) ซึ่งจะให้ความสำคัญกับขั้นตอนกระบวนการที่กระทำ

การเขียนโปรแกรมเชิงวัตถุนั้นเป็นการแบ่งโครงสร้างหรือส่วนประกอบออกเป็นส่วนๆซึ่งจะเรียกว่า คลาส (Class) และ วัตถุ (Object) เพื่อที่จะได้นำส่วนประกอบต่างๆเหล่านั้นกลับมาเรียกใช้งานอีกครั้งในการสร้างซอฟต์แวร์หรือโปรแกรมที่ต้องการ เพื่อลดความซ้ำซ้อนและเวลาในการพัฒนาโปรแกรมลงไป

การเขียนโปรแกรมเชิงวัตถุยังมีคุณสมบัติต่างๆ ดังนี้

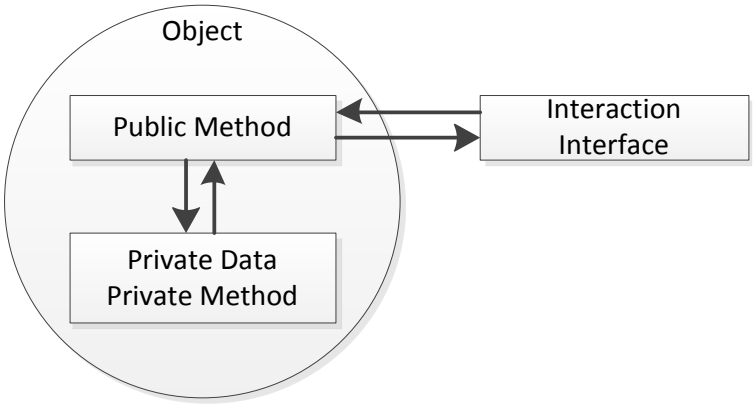
- คลาส (Class) ซึ่งถือว่าเป็นแม่แบบในการสร้างวัตถุ (Object) โดยที่คลาสๆหนึ่งนั้นสามารถนำไปสร้างวัตถุได้หลายตัว และวัตถุหลายๆตัวที่ถูกสร้างจากคลาสเดียวกันจะมีพฤติกรรมตามโครงสร้างของแม่แบบซึ่งก็คือคลาสแม่ของวัตถุเหล่านั้น แต่บางวัตถุก็อาจมีลักษณะเฉพาะบางอย่างที่เป็นของตนเอง แต่คุณสมบัติโดยรวมนั้นจะใกล้เคียงกันกับวัตถุที่ถูกสร้างมาจากคลาสเดียวกัน
- วัตถุ (Object) คือสิ่งต่างๆที่เป็นวัตถุเป้าหมายที่ต้องการและจะแสดงออกในลักษณะตัวตน (Instance) ของคลาส ซึ่งจะประกอบไปด้วย คุณสมบัติ (Properties) คือคุณสมบัติต่างๆที่วัตถุนั้นมี และพฤติกรรม (Behavior) หรือ (Method) ซึ่งก็คือพฤติกรรมที่มีไว้ตอบสนองต่อเหตุการณ์ (Event) ต่างๆ เช่น “Object: นักศึกษา” จะประกอบไปด้วย คุณสมบัติ (Properties) ชื่อ, เพศ, รหัสนักศึกษา, สาขาวิชา เป็นต้น และจะมีพฤติกรรม (Method) ที่จะตอบสนองต่อเหตุการณ์ต่างๆ เช่น การลงทะเบียนเรียน การเข้าเรียน การชำระค่าลงทะเบียนเรียน เป็นต้น จะแสดงตัวอย่างของความสัมพันธ์ระหว่างคลาสและวัตถุในรูปที่ 2.8

ในรูปที่ 2.8 นั้นจะแสดงความสัมพันธ์ระหว่างคลาสและวัตถุ ดังในภาพคลาสสุนัข ซึ่งมีคุณสมบัติทั่วไปของสุนัข เช่น การเห่า การกิน การนอน เป็นต้น และมีวัตถุของคลาสสุนัขคือ แดง และดำ โดยจะได้รับคุณสมบัติจากคลาส สุนัขมาเป็นของตัวเอง



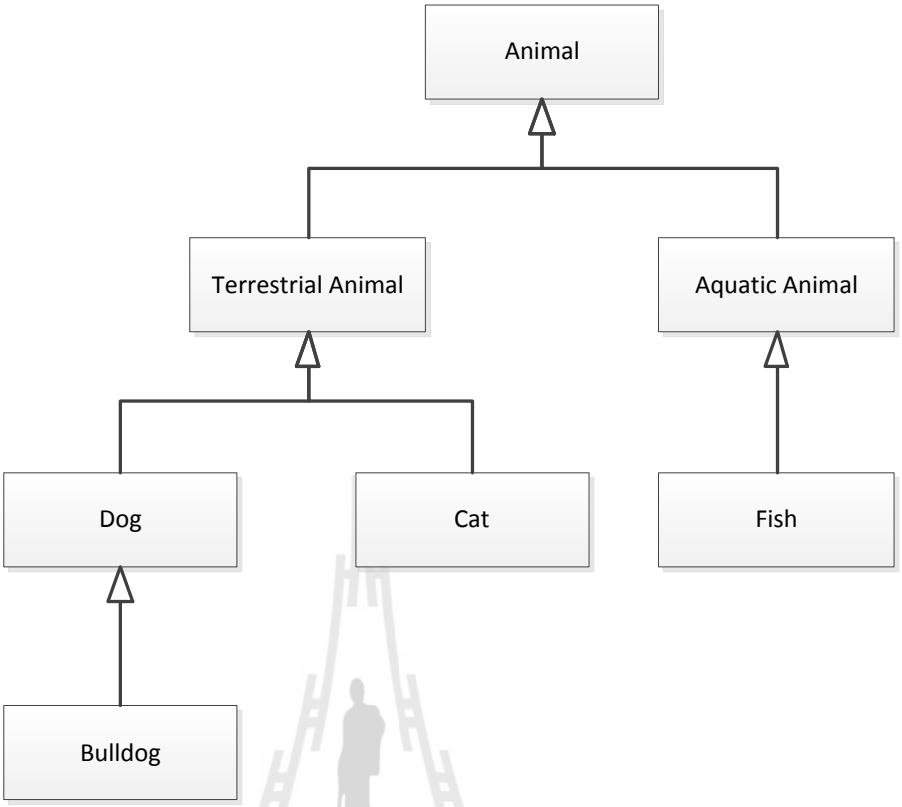
รูปที่ 2.8 รูปแสดงความสัมพันธ์ระหว่างคลาส (Class) และวัตถุ (Object)

- การห่อหุ้ม (Encapsulation) ในการสร้างคลาสหรือสร้าง คอนสตรัคเตอร์ (Constructors) ของคลาสนั้น ย่อมจะต้องมีการสร้างพฤติกรรมและคุณสมบัติเพื่อให้วัตถุที่สร้างมาจากคลาสนั้นทำงานได้ และบางกรณีมีการสร้างพฤติกรรมและคุณสมบัติเพื่อใช้ภายในคลาสเท่านั้นจึงไม่มีความจำเป็นให้คลาสอื่นๆเห็น ดังรูปที่2.9
- ไลบรารี่ (Library) คือชุดคำสั่งที่เก็บรวบรวมไว้ให้ผู้ใช้เรียกใช้งาน และมักจะมีการเรียกใช้งานการคำนวณและอัลกอริธึม (Algorithm) ในลักษณะต่างๆ ได้มีการแจกจ่ายและซื้อขายไลบรารี่ต่างๆเพื่อให้ผู้ใช้งานได้มีความสะดวกในการทำงาน เพื่อป้องกันความผิดพลาดจากการทำงานของการคำนวณและอัลกอริธึมต่างๆควรจะแยกให้มีการมองเห็นเฉพาะการคำนวณและอัลกอริธึมที่ต้องการซึ่งภาษาเชิงวัตถุนั้นก็มีคุณสมบัตินี้ อีกทั้งยังช่วยในเรื่องของการโดนแกะรอยไลบรารี่เพื่อนำไปพัฒนาต่อยอดและนำไปแสวงหาผลกำไรโดยผิดลิขสิทธิ์



รูปที่ 2.9 รูปแสดงถึงการเข้าถึงข้อมูลที่มีการห่อหุ้ม (Wikipedia, 2555)

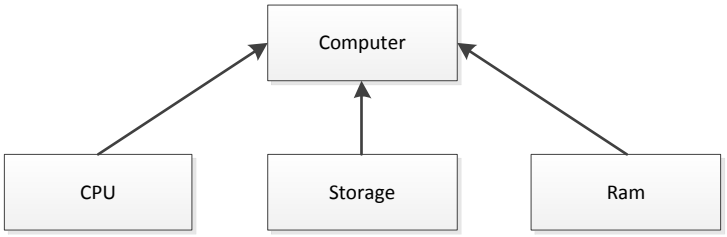
- การสืบทอดคุณสมบัติ (Inheritance) เป็นวิธีที่ทำให้สามารถนำคลาสกลับมาใช้ใหม่ได้อีกอย่างมีประสิทธิภาพ การถ่ายทอดข้อมูลคือ คุณสมบัติและพฤติกรรมจากคลาสลำดับที่สูงกว่า (Super Class) ไปคลาสลูก (Sub Class) โดยที่คลาสลูกนั้นสามารถเปลี่ยนแปลงหรือแทนที่ข้อมูล (Override) ที่ได้รับการถ่ายทอดมาได้ ดังรูปที่ 2.10 รูปแสดงความสัมพันธ์ระหว่างคลาสแม่ (Class) และคลาสลูก (Sub-Class) จะแสดงถึงความสำพันธ์ระหว่างคลาสในลักษณะของการสืบทอดคุณสมบัติ



รูปที่ 2.10 รูปแสดงความสัมพันธ์ระหว่างคลาสแม่ (Class) และคลาสลูก (Sub-Class)

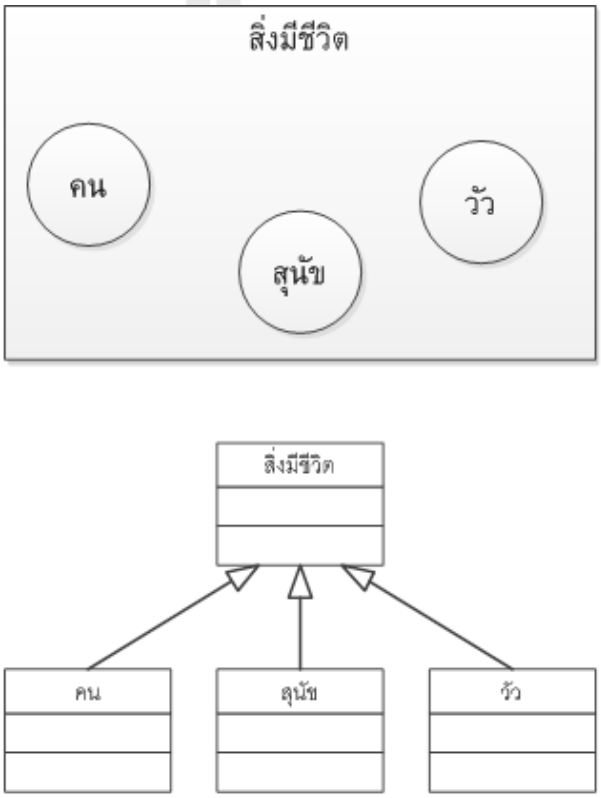
โดยปกตินั้นความสัมพันธ์ระหว่างคลาสมี 2 รูปแบบคือ การประกอบ (Composition) และการสืบทอดคุณสมบัติ (Inheritance)

การประกอบ (Composition) เป็นลักษณะของความสัมพันธ์ระหว่างคลาสที่ระบุว่าคลาสลูกนั้นเป็นส่วนประกอบของคลาสแม่ ดังรูปที่ 2.11 แสดงความสัมพันธ์แบบการประกอบ (Composition) จะเห็นว่า CPU นั้นไม่ใช่ Computer แต่เป็นส่วนประกอบของ Computer



รูปที่ 2.11 รูปแสดงความสัมพันธ์แบบการประกอบ (Composition)

การพ้องรูป (Polymorphism) เป็นคุณสมบัติที่สำคัญของการเขียนโปรแกรมเชิงวัตถุ การพ้องรูปคือวัตถุที่ถูกประกาศขึ้นจากคลาสที่ได้สืบทอดจากอีกคลาสนั้นสามารถถูกสร้างโดยกำหนดคุณสมบัติให้เป็นของคลาสที่ถูกสืบทอดมาได้ การพ้องรูปนั้นมีประโยชน์ในการนำคลาสกลับมาใช้ใหม่ได้อีกและเป็นกลไกสำคัญในการเขียนโปรแกรมเชิงวัตถุที่เป็นความสามารถของการสืบทอดคุณสมบัติ (Inheritance) ยกตัวอย่างเช่น ให้ระบุสิ่งมีชีวิตมา 1 ชนิด ไม่ว่าจะได้คำตอบว่า คน สุนัขหรือวัว ก็ล้วนเป็นสิ่งมีชีวิต ดังนั้นหากเราสร้างคลาส สิ่งมีชีวิต ขึ้นมาก็จะได้ตามรูปที่ 2.12 รูปแสดงความสัมพันธ์ของสิ่งมีชีวิต



รูปที่ 2.12 รูปแสดงความสัมพันธ์ของสิ่งมีชีวิต

นอกจากการสืบทอดคุณสมบัติ (Inheritance) และการพ้องรูป (Polymorphism) แล้ว คุณสมบัติของโปรแกรมเชิงวัตถุ ยังมี Overload, Override, Dynamic binding เป็นต้น

Overload คือการใช้ชื่อเมธอด (Method) ที่มีชื่อเหมือนกันภายในคลาสเดียวกันแต่มีพารามิเตอร์ต่างกัน ไม่ว่าจะต่างกันที่ชนิดของพารามิเตอร์ หรือจำนวนของพารามิเตอร์

Override คือคลาสที่สืบทอดมากนั้นได้ประกาศเมธอดที่มีชื่อและพารามิเตอร์เหมือนกันกับคลาสที่ถูกสืบทอดมา ทำให้เมธอดที่ถูกประกาศขึ้นมาใหม่นั้นทับเมธอดที่ถูกสืบทอดมา

Dynamic binding เป็นการสืบทอดคุณสมบัติจากคลาสที่ถูกสืบทอดมาทำให้คลาสที่สืบทอดนั้นมีคุณสมบัติเช่นเดียวกันกับคลาสที่ถูกสืบทอด เว้นแต่จะมีการ Override เพื่อแก้ไขเมธอดที่ถูกสืบทอดมาเพื่อให้เหมาะสมกับการทำงานของเมธอดนั้นๆ

ข้อดีข้อเสีย

ในการพัฒนาซอฟต์แวร์ประเภทเชิงวัตถุนั้น จะทำให้เข้าใจได้ไม่ยากสำหรับผู้ที่มีความรู้พื้นฐานด้านการเขียนโปรแกรมมาบ้างแล้วเพราะได้มีการจำลองให้เหมือนในโลกแห่งความเป็นจริง มองทุกๆอย่างเป็นวัตถุ เมื่อมีการมองทุกๆอย่างเป็นวัตถุแล้ว การบำรุงรักษาก็สามารถทำได้ง่ายขึ้นด้วย มีการนำกลับมาใช้ใหม่ได้ (Reusability) และใช้ได้หลายระบบปฏิบัติการ

ถ้าเป็นซอฟต์แวร์ที่มีความซับซ้อนมากๆ นั้นจำทำให้ยากต่อการจัดการ และมีความกำกวมเนื่องจากมีความสำพันธ์ระหว่างโมดูลแบบซับซ้อน และซอฟต์แวร์แบบเชิงวัตถุนั้นจะทำงานได้ช้ากว่าโปรแกรมแบบดั้งเดิม

2.4 งานวิจัยที่เกี่ยวข้อง

ในการศึกษาและพัฒนาการสร้างกรณีทดสอบสำหรับการทดสอบระดับรวมหน่วยแบบเพิ่มทีละหน่วยโดยอัตโนมัติจากกรณีทดสอบระดับหน่วย ผู้วิจัยได้ทำการศึกษาค้นคว้างานวิจัยในอดีตที่เกี่ยวข้องกับการสร้างกรณีทดสอบสำหรับการทดสอบระดับรวมหน่วยแบบเพิ่มทีละหน่วยโดยอัตโนมัติจากกรณีทดสอบระดับหน่วยดังสรุปในตารางที่ 2.1 และมีรายละเอียดดังต่อไปนี้

งานวิจัยของ W.K. Chan T.Y.Chen และ T.H. Tse (2002) ได้นำเสนอภาพรวมของเทคนิคต่างๆที่ใช้ในการทดสอบระดับรวมหน่วยของซอฟต์แวร์เชิงวัตถุซึ่งสรุปได้ดังนี้

- State-based approach ใช้การโต้ตอบของเครื่องสถานะจำกัด (Finite state machines) ในการจำลองระบบ แต่ความยากของเทคนิคนี้จะเพิ่มขึ้นเมื่อจำนวนของหน่วยนั้นเพิ่มขึ้น
- Event-based paradigm ใช้ความสัมพันธ์ของกิจกรรม (Events) ในการควบคุมระบบและตรวจสอบการฝ่าฝืนข้อกำหนด การแก้ไขที่ต่างกันของประเภทของ

เหตุการณ์ระหว่างวัตถุที่ต่างกันหรือสถานะที่แตกต่างกันซึ่งต้องการให้มีการวิจัยต่อไป

- Integrated formal methods เป็นการติดตามแนวโน้ม แต่ความซับซ้อนของการประกอบกันของโมเดลอาจทำให้เกิดข้อผิดพลาดได้
- Deterministic testing มีการบังคับให้ใช้วิธีการประสาน (synchronization) ในการดำเนินการ ซึ่งได้จัดลำดับตามความต้องการเพื่อให้การทำนายกรณีทดสอบนั้นเป็นไปตามที่กำหนด เนื่องจากการแปลงแอดเดรสแบบพลวัต (Dynamic Binding) ทำให้การประสานอาจส่งผลกระทบต่อจุดที่มีการผูกข้อมูลที่มีความแตกต่างกัน จึงอาจก่อให้เกิดความสับสนได้

วิทยานิพนธ์ของ Konstantin Rubinov (2010) แห่งมหาวิทยาลัย Lugano ได้นำเสนอภาพรวมของเทคนิคต่างๆที่ใช้ในการทดสอบระดับรวมหน่วยของซอฟต์แวร์เชิงวัตถุ นำเสนอแนวทางในการสร้างกรณีทดสอบระดับรวมหน่วยแบบอัตโนมัติ งานวิจัยชิ้นนี้ได้สังเกตว่าซอฟต์แวร์นั้นมักจะมีกรณีทดสอบระดับหน่วยจำนวนมาก และกรณีทดสอบระดับหน่วยที่มีจำนวนมากนั้นก็มีข้อมูลที่เป็นประโยชน์อยู่ภายในนั้นมากเช่นกัน ซึ่งสามารถนำมาใช้ประโยชน์ในการสร้างกรณีทดสอบระดับรวมหน่วยได้ จึงได้แสดงให้เห็นถึงวิธีการในการตรวจสอบข้อมูลในกรณีทดสอบระดับหน่วยด้วยเทคนิคการวิเคราะห์แบบคงที่เพื่อที่จะผสานกรณีทดสอบระดับหน่วยในการสร้างกรณีทดสอบระดับรวมหน่วยที่มีประโยชน์ ผลการศึกษาของงานวิจัยชิ้นนี้ได้แสดงความมีประสิทธิภาพของกรณีทดสอบระดับหน่วยในการสร้างกรณีทดอบระดับรวมหน่วย

สมคะเน บาลลา และ พิชโยทัย มหัทธนาภิวัฒน์ (2011) จากมหาวิทยาลัยเทคโนโลยีสุรนารี ได้เสนอเทคนิคในการเปรียบเทียบหาความแตกต่างของซอร์สโค้ดโดยการนำโครงสร้างไวยากรณ์ต้นไม้ (Abstract Syntax Tree) มาประยุกต์ใช้ในการเปรียบเทียบหาความแตกต่างระหว่างซอร์สโค้ดที่มีความต่างเวอร์ชั่นกัน ซึ่งงานวิจัยชิ้นนี้ได้นำเสนอเทคนิคต่างๆที่มีความสามารถและคุณสมบัติแตกต่างกันไปอีกทั้งยังมีการนำ Abstract Syntax tree มาประยุกต์ใช้ในการแสดงโครงสร้างของซอร์สโค้ด

วิทยานิพนธ์ของ Alessandro Orso (1998) จาก POLITECNICO DI MILANO ได้นำเสนอวิธีการทดสอบระดับรวมหน่วยของโปรแกรมแบบเชิงวัตถุโดยภาพรวม และได้นำเสนอเทคนิควิธีการในการทดสอบระดับรวมหน่วยในส่วนของ Polymorphism ซึ่งเป้าหมายคือการกำหนดกล

ยุทธ์ที่เหมาะสมกับระบบเชิงวัตถุ เพื่อระบุเส้นทางของการทำงานที่สำคัญและการเชื่อมโยงในระหว่างที่ทำการทดสอบ และเพื่อให้ได้หลักเกณฑ์ที่เพียงพอในการสร้างกรณีทดสอบระดับรวมหน่วย

งานวิจัยของ Hai Yuan และ Tao Xie (2006) จาก North Carolina State University ได้นำเสนอกรอบความคิด (Framework) สำหรับการทดสอบซอฟต์แวร์ระดับรวมหน่วยแบบอัตโนมัติโดยอยู่บนพื้นฐานของการอนุมานเงื่อนไขตามลำดับจากการดำเนินการแบบไดนามิคโดยมีชื่อว่า “Substra” และได้นำกรอบความคิดที่เสนอมานำไปใช้ร่วมกับเครื่องมือในการทดสอบระบบ ATM และได้ผลการศึกษาเบื้องต้นนั้นแสดงให้เห็นว่ามีประสิทธิภาพในการสร้างกรณีทดสอบสำหรับทดสอบพฤติกรรมของโปรแกรมที่พัฒนา

จากการศึกษางานวิจัยที่เกี่ยวข้องกับการพัฒนาระบบการสร้างกรณีทดสอบสำหรับการทดสอบระดับรวมหน่วยแบบเพิ่มทีละหน่วยโดยอัตโนมัติจากกรณีทดสอบระดับหน่วย ได้พบว่าแนวคิดในการสร้างกรณีทดสอบแบบอัตโนมัตินั้นได้รับความสนใจ เพราะว่าความต้องการในการลดความซับซ้อน เวลา และค่าใช้จ่ายในการสร้างกรณีทดสอบ โดยที่แต่ละงานวิจัยได้นำเสนอข้อมูลส่วนต่างๆที่มีประโยชน์ มีตั้งแต่การนำเสนอเทคนิคในการทดสอบระดับรวมหน่วยแบบต่างๆ การค้นพบข้อมูลที่จำเป็นในการสร้างกรณีทดสอบระดับรวมหน่วยจากกรณีทดสอบระดับหน่วย การนำโครงสร้างไวยากรณ์ต้นไม้มาประยุกต์ใช้ในการระบุโครงสร้างของโมดูลจากซอร์สโค้ด แต่อย่างไรก็ตามงานวิจัยที่ผู้วิจัยต้องการพัฒนานั้นจะเป็นการทดสอบระดับรวมหน่วยการมุ่งเน้นไปที่พิสูจน์ความสัมพันธ์ระหว่างโมดูล ไม่ได้พัฒนาออกมารองรับการสร้างและกำหนดเส้นทางการดำเนินงานของโมดูล

ตารางที่ 2.1 สรุปเปรียบเทียบงานวิจัยที่เกี่ยวข้องกับการการสร้างกรณีทดสอบสำหรับการทดสอบระดับรวมหน่วยแบบเพิ่มทีละหน่วยโดยอัตโนมัติจากกรณีทดสอบระดับหน่วย

บทความวิจัยที่เกี่ยวข้องประกอบด้วย “ก” แทนงานวิจัยของ W.K. Chan T.Y. Chen และ T.H. Tse (2002) “ข” วิทยานิพนธ์ของ Konstantin Rubinov (2002) “ค” สมคะเน บาลลา และ พิชโยทัย มหัทธนาภิวัฒน์ (2011) “ง” วิทยานิพนธ์ของ Alessandro Orso (1998) “จ” งานวิจัยของ Hai Yuan และ Tao Xie(2006) และ “ฉ” แทนงานวิจัยเรื่องการสร้างกรณีทดสอบสำหรับการทดสอบระดับรวมหน่วยแบบเพิ่มทีละหน่วยโดยอัตโนมัติจากกรณีทดสอบระดับหน่วย (งานวิจัยของวิทยานิพนธ์ฉบับนี้)

กระบวนการทำงาน	งานวิจัยที่เกี่ยวข้อง					
	ก	ข	ค	ง	จ	ฉ
การทดสอบซอฟต์แวร์						
ระดับของการทดสอบซอฟต์แวร์						
การทดสอบซอฟต์แวร์เชิงวัตถุ	✓			✓		✓
การทดสอบระดับหน่วย (Unit Testing)		✓		✓		
การทดสอบระดับรวมหน่วย (Integration Testing)	✓	✓		✓	✓	✓
เทคนิคการทดสอบระดับรวมหน่วย	✓	✓		✓	✓	✓
เทคนิคการทดสอบแบบเพิ่มทีละหน่วย				✓		✓
กระบวนการสร้างกรณีทดสอบอัตโนมัติ						
ประยุกต์โครงสร้างไวยากรณ์ต้นไม้(Abstract Syntax Tree)			✓			✓
การสร้างกรณีทดสอบอัตโนมัติ		✓			✓	✓
ขอบเขตของการวิจัย						
วิจัยเพื่อนำเสนอแนวคิดเท่านั้น	✓	✓		✓	✓	
วิจัยเพื่อนำเสนอเครื่องมือที่พัฒนา			✓			✓

บทที่ 3
วิธีการดำเนินการวิจัย

งานวิจัยนี้มีจุดประสงค์ที่จะพัฒนาเครื่องมือสำหรับสร้างกรณีทดสอบสำหรับการทดสอบระดับรวมหน่วยแบบเพิ่มทีละหน่วยโดยอัตโนมัติจากกรณีทดสอบระดับหน่วย โดยโปรแกรมที่พัฒนาขึ้นได้ใช้ภาษาจาวาและใช้ต้นไม้วากยสัมพันธ์แบบนามธรรม (Abstract Syntax Tree) ที่ได้ปรับปรุงโครงสร้างให้เหมาะสมในการแจกแจงความสัมพันธ์ระหว่างโมดูล ในบทนี้จะนำเสนอถึงวิธีการวิจัย เครื่องมือที่ใช้ในการวิจัย และกระบวนการต่างๆ ของการวิจัย โดยมีรายละเอียดดังนี้

3.1 กรอบแนวคิดของการวิจัย

แนวคิดหลักของงานวิจัยนี้คือการพัฒนาเครื่องมือเพื่อสร้างกรณีทดสอบระดับรวมหน่วย (Integration Testcase) โดยใช้กรณีทดสอบระดับหน่วย(Unit Testcase) มีการใช้ข้อมูลในการนำเข้า 2 ส่วนคือ ซอร์สโค้ดภาษาจาวาที่จะสร้างกรณีทดสอบ และกรณีทดสอบระดับหน่วยของซอร์สโค้ดที่นำเข้า กระบวนการทำงานนั้นจะแบ่งออกเป็น 2 ส่วน คือ กระบวนการแจกแจงโครงสร้างของซอร์สโค้ด และ กระบวนการสร้างกรณีทดสอบระดับรวมหน่วย

ในการพัฒนาเครื่องมือสำหรับสร้างกรณีทดสอบสำหรับการทดสอบระดับรวมหน่วยแบบเพิ่มทีละหน่วยโดยอัตโนมัติจากกรณีทดสอบระดับหน่วย มีขั้นตอนในการพัฒนาดังนี้

1. การออกแบบและพัฒนาโครงสร้างต้นไม้เพื่อใช้ในการแจกแจงโครงสร้างและความสัมพันธ์ระหว่างโมดูลจากซอร์สโค้ด
2. การออกแบบและพัฒนาการนำเข้ากรณีทดสอบระดับรวมหน่วย
3. การออกแบบและพัฒนากระบวนการสร้างกรณีทดสอบระดับรวมหน่วย
4. การออกแบบและพัฒนาเครื่องมือสำหรับสร้างกรณีทดสอบระดับรวมหน่วยแบบอัตโนมัติ

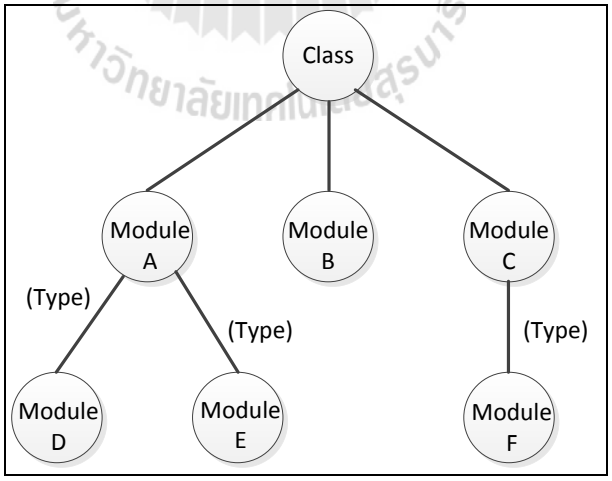
โดยมีรายละเอียดของแต่ละขั้นตอนอธิบายในหัวข้อต่อไปนี้

3.1.1 การออกแบบและพัฒนาโครงสร้างต้นไม้เพื่อใช้ในการแจกแจงโครงสร้างและความสัมพันธ์ระหว่างโมดูลจากซอร์สโค้ดภาษาจาวา

ในการพัฒนาโครงสร้างต้นไม้ภายในงานวิจัยนี้จะมุ่งเน้นไปที่การพัฒนาโครงสร้างต้นไม้วากยสัมพันธ์แบบนามธรรม (Abstract Syntax Tree) เพื่อแจกแจงโครงสร้างและความสัมพันธ์ระหว่างโมดูลจากซอร์สโค้ดภาษาจาวา โดยโครงสร้างต้นไม้ที่พัฒนาขึ้นมานั้นประกอบไปด้วยโครงสร้างหลัก 3 ส่วนคือ

1. Root เป็นตัวแทนของชื่อคลาส กำหนดค่าภายในคือ ชื่อของคลาส
2. Node เป็นตัวแทนของแต่ละโมดูลโดยจะเก็บชื่อของโมดูลที่อยู่ภายในคลาสและพารามิเตอร์ของโมดูล
3. Edge เป็นตัวแสดงความสัมพันธ์ระหว่าง Node จะเก็บค่าอาร์กิวเมนต์ของโมดูลที่ส่งให้กัน ในกรณีที่เกิดความสัมพันธ์ระหว่างโมดูลโดยไม่มีการส่งค่าอาร์กิวเมนต์นั้น Edge จะแสดงค่า null

โครงสร้างทั้ง 3 ส่วนนี้จะมี Attribute อยู่ภายในซึ่งใช้ในการระบุตัวตนและแสดงความสัมพันธ์ต่างๆ ดังรูปที่ 3.1

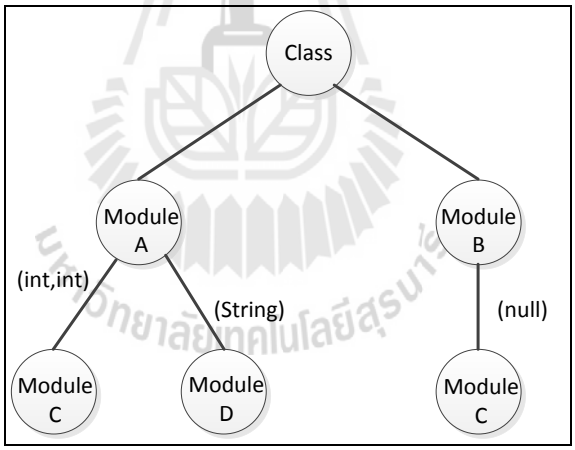


รูปที่ 3.1 โครงสร้าง AST ที่ได้ปรับปรุง

Node แต่ละ Node จะเรียงตามการเรียกใช้ Node อื่น โดยมี Edge แสดงความสัมพันธ์ระหว่าง Node โดยที่ Edge จะมีค่าอาร์กิวเมนต์ซึ่งจะสัมพันธ์กับพารามิเตอร์ ของโมดูลผู้รับ Edge ที่เชื่อมระหว่าง Root และ Child นั้นจะไม่มี Attribute ภายในเนื่องจากไม่ได้เกิดความสัมพันธ์ระหว่างระหว่างโมดูล เพียงแต่เป็นการแสดงโครงสร้างต้นไม้ให้สมบูรณ์

ระดับของ Node ภายในต้นไม้จะแสดงถึงลำดับการเรียกใช้โมดูล โดยที่ Node ที่ไม่ได้ถูกเรียกใช้จะอยู่ระดับบนสุด และ Node ที่ไม่ได้เรียกโมดูลใดๆจะอยู่ระดับล่างสุด

ภายในโครงสร้างต้นไม้นี้จะให้ความสำคัญกับโมดูลที่มีความสัมพันธ์กัน โดยซอร์สโค้ดที่นำเข้ามาในกระบวนการนี้จะต้องผ่านการ Compile จาก Compiler เรียบร้อยแล้ว ทำให้ไม่จำเป็นต้องคำนึงถึงการ Encapsulation เนื่องจาก Compiler ทำหน้าที่ตรวจสอบความถูกต้องให้แล้ว ในกรณีที่โมดูลมีการ Overloading คือการมีโมดูลที่มีชื่อเหมือนกันแต่มีพารามิเตอร์ต่างกัน จะมีการแสดงโมดูลแยกออกจากกันตามโครงสร้างของการเรียกใช้งานตามรูปที่ 3.2 ซึ่งโมดูล C จะถูกเรียกใช้ทั้ง2โมดูลแต่อาร์กิวเมนต์ที่ได้รับจะแตกต่างกันไปตามพารามิเตอร์ของโมดูลนั้นๆ

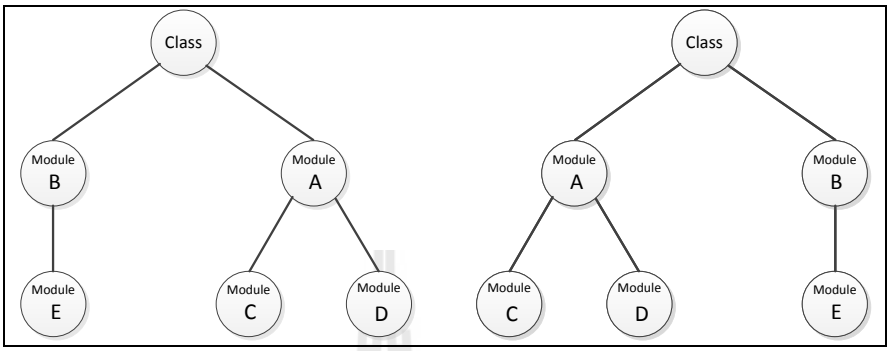


รูปที่ 3.2 ภาพแสดงโครงสร้าง AST ที่ปรับปรุงแล้วกับโมดูล Overloading

ในกรณีที่มีการสืบทอดระหว่างคลาส Node ของคลาสแม่ที่ถูกใช้งานจะปรากฏบนโครงสร้างต้นไม้ของคลาสลูก เปรียบเสมือนเป็น Node ของคลาสลูก

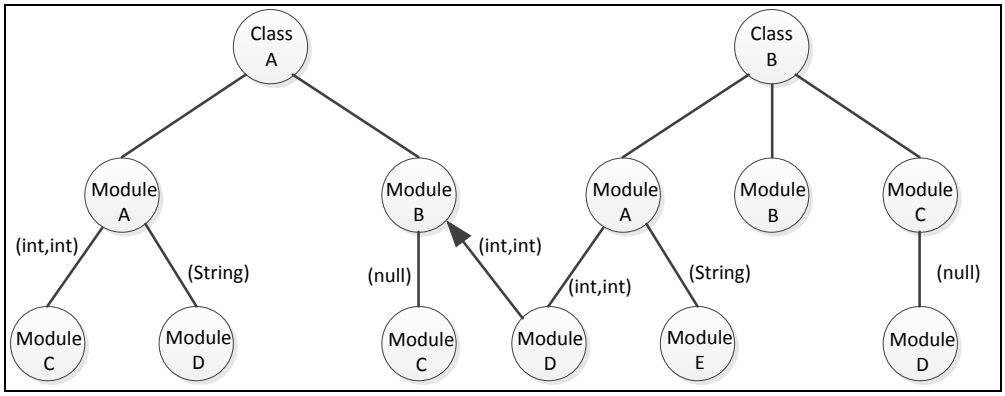
จำนวนของต้นไม้นั้นจะขึ้นอยู่กับจำนวนของคลาส โดยจะไม่สนใจคลาสที่เป็น Interface เนื่องจากโมดูลภายใน Interfaceไม่มีการ Implement ทำให้ไม่มีผลต่อโครงสร้างต้นไม้สำหรับแจกแจงความสัมพันธ์นี้ รูปร่างของต้นไม้แต่ละต้นนั้นจะแปรผันตามลำดับและความซับซ้อนของโมดูลที่มีความสัมพันธ์กันทำให้การจัดวางตำแหน่งของโมดูล การ

จัดลำดับของ Node ของโมดูลระดับแรกนั้น สามารถสลับตำแหน่งได้ขึ้นอยู่กับกระบวนการระบุโมดูล ดังนั้น ซอร์สโค้ดที่นำเข้าสู่กระบวนการสร้างสามารถสร้างต้นไม้ได้หลายแบบ แต่สามารถใช้อธิบายโครงสร้างซอร์สโค้ดได้เช่นเดียวกันดังรูปที่ 3.3



รูปที่ 3.3 ภาพแสดงความแตกต่างของโครงสร้างต้นไม้จากซอร์สโค้ดเดียวกัน

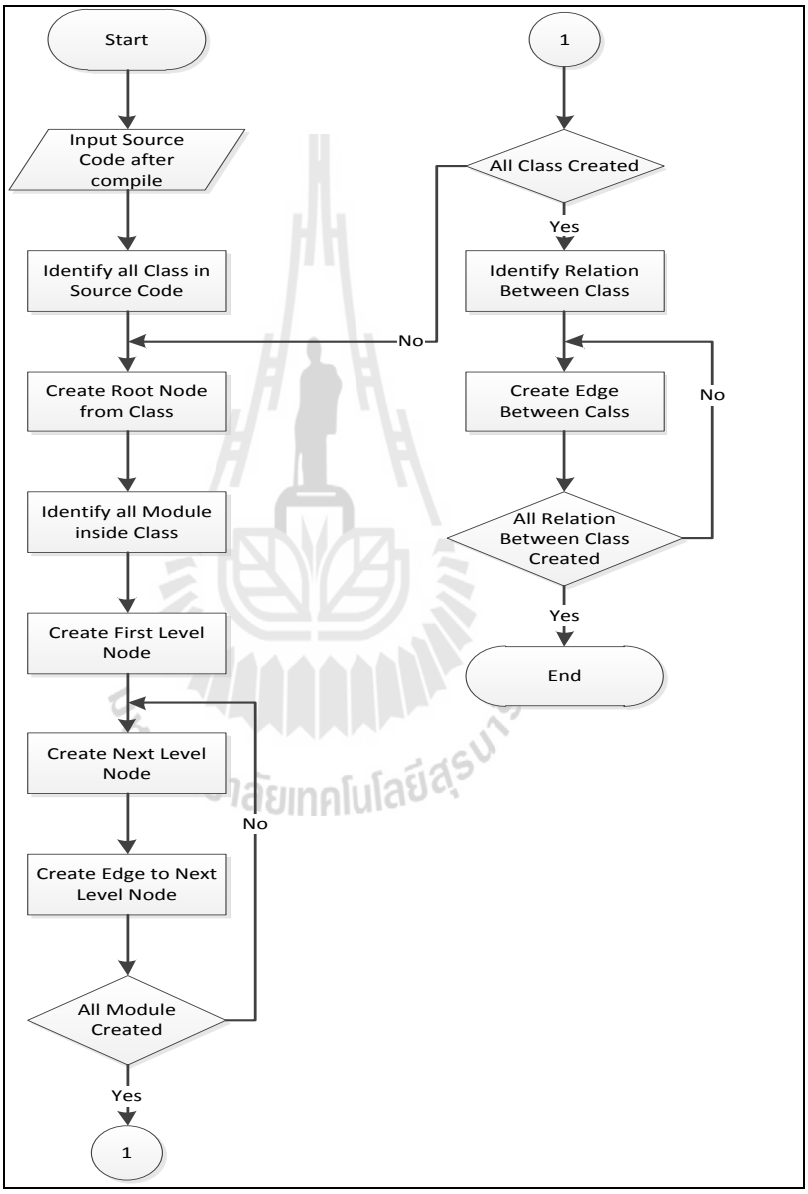
การเรียกใช้โมดูลจากคลาสไปยังโมดูลจากคลาสอื่นๆนั้นจะทำให้เกิดโครงสร้างความสัมพันธ์ระหว่างต้นไม้โดยมี Edge เชื่อมจาก Node ของต้นไม้ต้นหนึ่งไปยังอีกต้นหนึ่ง Edge ดังกล่าวนี้จำเป็นที่จะต้องมีการกำหนดทิศทางอย่างชัดเจน เนื่องจากการแสดงโมเดลโครงสร้างต้นไม้นี้ แต่ละต้นเป็นอิสระต่อกัน ถ้าไม่มีการกำหนดทิศทางของ Edge แล้วจะทำให้เกิดความสับสนระหว่างโมดูลผู้เรียกและผู้ถูกเรียกโดยมี Edge เชื่อมโยงดังรูปที่ 3.4 ซึ่งเป็นภาพที่แสดงความสัมพันธ์ระหว่างต้นไม้ โดยที่ความสัมพันธ์ข้ามคลาสนั้นเกิดขึ้นที่โมดูล D ภายในคลาส B และโมดูลB ภายในคลาส A



รูปที่ 3.4 ภาพแสดงความสัมพันธ์ระหว่างต้นไม้

ความซับซ้อนของต้นไม้นั้นจะขึ้นอยู่กับโครงสร้างความสัมพันธ์ระหว่างโมดูล ถ้าแต่ละคลาสมีความสัมพันธ์ระหว่างโมดูลจำนวนมากมาก ต้นไม้ก็จะมี Edge เชื่อมต่อกันระหว่างต้นไม้เป็นจำนวนมากตามไปด้วย

ในกระบวนการสร้างโครงสร้างตันไม้สำหรับแจกแจงความสัมพันธ์ระหว่างโมดูลนั้นมีขั้นตอนดังรูปที่ 3.5

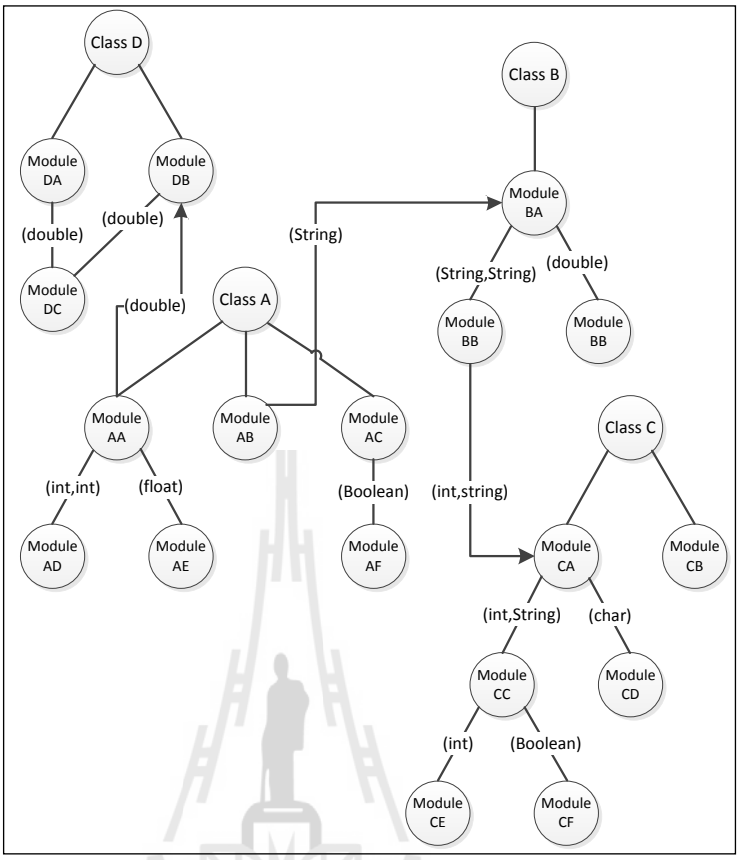


รูปที่ 3.5 ภาพแสดงผังงาน (Flow chart) ของกระบวนการสร้างโครงสร้างต้นไม้

จากรูปที่ 3.5 แต่ละขั้นตอนมีรายละเอียดดังนี้

1) Input Source Code after Compile : เป็นการนำเข้าซอร์สโค้ดที่ผ่านการ Compile แล้ว
2) Identify All Class in Source Code : เป็นการระบุคลาสทั้งหมดที่มีในซอร์สโค้ด รวมถึงการตรวจสอบว่าเป็น Interface หรือไม่
3) Create Root Node from Class : สร้าง Root โดยกำหนด Attribute เป็น ชื่อของคลาส
4) Identify all Module inside Class :นำข้อมูลของคลาสมาจำแนกโมดูลภายในโดยเริ่มจากระบุโมดูลที่มีทั้งหมดภายในคลาส และแยกระหว่างโมดูลที่เป็นผู้เรียก ผู้ถูกเรียก และเป็นทั้งสองอย่าง รวมถึงการแจกแจงพารามิเตอร์จองแต่ละโมดูลด้วย
5) Create First Level Node : กำหนด Node ระดับแรกจากโมดูลที่ไม่มีผู้เรียกก่อน แล้วใช้โมดูลระดับแรกที่กำหนดไว้แล้วมาระบุโมดูลที่ถูกเรียกจากโมดูลระดับแรก
6) Create Next Level Node : สร้าง Node ระดับถัดมาโดยอ้างอิงจาก Node ระดับก่อนหน้าจากนั้นตรวจสอบว่า Node นั้นมีการเรียกโมดูลหรือไม่ ถ้ามีการเรียกโมดูลอื่นจะนำโมดูลที่ถูกเรียกมาสร้างเป็นลำดับต่อไป ถ้าไม่มีการเรียกใช้โมดูลอื่น ก็จะข้ามโมดูลนี้ไป
7) Create Edge to Next Level Node : กำหนด Edge จาก พารามิเตอร์ ของโมดูลที่ถูกเรียก ถ้าโมดูลระดับแรกนี้ไม่มีการเรียกโมดูลอื่นก็ข้ามโมดูลนี้ไป
8) All Module Created : ทำขั้นตอนที่ 6 และ7 ซ้ำจนครบทุกโมดูล
9) All Class Created : ทำขั้นตอนที่ 3 ถึง 8 จนครับทุกคลาส
10) Identify Relation Between Class : ระบุความสัมพันธ์ของแต่ละคลาส เพื่อกำหนดจุดเชื่อมต่อระหว่างคลาส
11) Create Edge Between Class : สร้าง Edge เชื่อมโมดูลที่มีการเรียกข้ามคลาส
12) All Relation Between Class Created : ทำขั้นตอนที่ 11 ซ้ำจนครบทุกความสัมพันธ์ระหว่างโมดูลข้ามคลาสที่เกิดขึ้น

หลังจากทำตามขั้นตอนดังกล่าวเสร็จสิ้นแล้วจะได้กลุ่มของโครงสร้างต้นไม้ที่มีความสัมพันธ์กันดังรูปที่ 3.6 ซึ่งกลุ่มของต้นไม้ที่ได้มานั้นเป็นโครงสร้างต้นไม้ที่สามารถแสดงความสัมพันธ์ระหว่างโมดูลภายในคลาสเดียวกันและความสัมพันธ์ระหว่างโมดูลภายในคลาสที่มีความสัมพันธ์กับคลาสอื่นๆ



รูปที่ 3.6 ภาพแสดงกลุ่มของโครงสร้างต้นไม้ที่จำลองขึ้น

3.1.2 การออกแบบและพัฒนาการนำเข้ากรณีทดสอบระดับรวมหน่วย

กรณีทดสอบระดับหน่วยคือกรณีที่ใช้สำหรับทดสอบโมดูลทุกๆโมดูลเพื่อตรวจสอบว่าแต่ละโมดูลนั้นสามารถทำงานได้ถูกต้องตามที่ได้ออกแบบไว้ ซึ่งการออกแบบกรณีทดสอบระดับหน่วยนั้นจะออกแบบโดยการวิเคราะห์โครงสร้างและขอบเขตของตัวแปรต่างๆที่มีบทบาทภายในโมดูล

กรณีทดสอบระดับหน่วยนั้นจะต้องเป็นกรณีทดสอบระดับหน่วยที่ผ่านการทดสอบแล้วและอยู่ในรูปแบบไฟล์สกุล Comma Separated Values (CSV) ไฟล์ CSV คือไฟล์ที่เก็บข้อมูลโดยมี comma (,) เป็นตัวแบ่งคอลัมน์ของข้อมูลภายใน ใช้การขึ้นบรรทัดใหม่เป็นการแบ่งแถวของข้อมูล CSV นั้นเป็นข้อมูลที่มีความเรียบง่าย สามารถเปิดดูได้ด้วย Text Editor ทั่วๆไป มักใช้ในการนำเข้าและส่งออกข้อมูล มักพบในงานเกี่ยวกับฐานข้อมูล

ไฟล์ CSV ดังกล่าวจะมีโครงสร้างดังตารางที่ 3.1 โครงสร้างของข้อมูลจะมีลักษณะดังรูปที่ 3.7

TestID	Class	Module	TestCase	Expected Result	Result
T1	Class_A	ModuleAD	5,5	10	10
T2	Class_A	ModuleAE	5f	5f	5f
T3	Class_A	ModuleAF	false	false	false

ตารางที่ 3.1 ตารางแสดงโครงสร้างของไฟล์กรณีทดสอบ

จากตารางที่ 3.1 แต่ละคอลัมน์ ประกอบไปด้วย

TestID คือรหัสของกรณีทดสอบ

Class คือชื่อของคลาส

Module คือชื่อของโมดูล

TestCase คือกรณีที่ใช้ในการทดสอบซึ่งจะสัมพันธ์กับพารามิเตอร์ของโมดูลนั้นๆ

Expected Result คือ ผลลัพธ์ที่คาดหวังจากการทดสอบโดยใช้กรณีทดสอบ

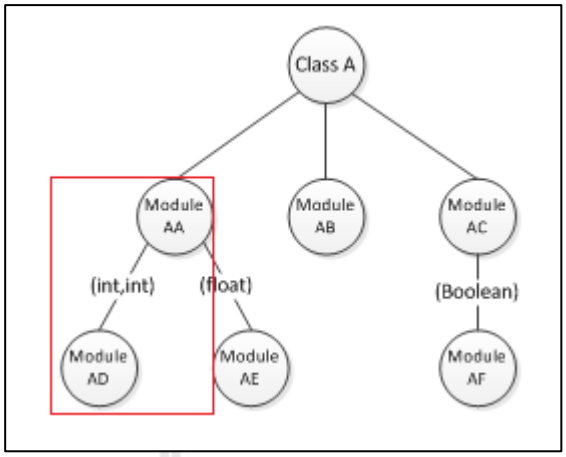
Result คือผลลัพธ์จริงที่ได้จากการทดสอบโดยใช้กรณีทดสอบ

```
TestID,Class,Module,TestCase,Expected Result,Result
T1,Class_A,ModuleAD,"5,5",10,10
T2,Class_A,ModuleAE,5f,5f,5f
T3,Class_A,ModuleAF,false,false,false
```

รูปที่ 3.7 ตัวอย่างไฟล์สกุล CSV

3.1.3 การออกแบบและพัฒนากระบวนการสร้างกรณีทดสอบระดับรวมหน่วยแบบอัตโนมัติ

กระบวนการสร้างกรณีทดสอบนั้นจะเริ่มจากการจำแนกโครงสร้างของซอร์สโค้ดโดยใช้โครงสร้างต้นไม้วากยสัมพันธ์แบบนามธรรมที่ปรับปรุงขึ้นมา ซึ่งในขั้นตอนสุดท้ายจะได้กลุ่มของโครงสร้างต้นไม้ หากพิจารณาแต่ละโมดูลที่มีความสัมพันธ์กัน อย่างเช่นรูปที่ 3.8 จะเห็นได้ว่า ModuleAA ได้มีความสัมพันธ์กับ ModuleAD โดยที่ ModuleAD ได้รับอาร์กิวเมนต์ที่เป็น (int,int) มาจาก ModuleAA



รูปที่ 3.8 ตัวอย่างโครงสร้างต้นไม้

จากนั้นพิจารณาข้อมูลจากตารางที่ ตารางที่ 3.2 ตารางแสดงตัวอย่างกรณีทดสอบระดับหน่วยจะเห็นได้ว่า กรณีทดสอบระกับหน่วยของ ModuleAD มีข้อมูลที่ใช้ในการทดสอบคือ 5,5 และผลลัพธ์ที่คาดหวัง(Expected Result) มีค่าเท่ากับผลลัพธ์(Result) ซึ่งก็หมายความว่า กรณีทดสอบระดับหน่วยของ ModuleAD นั้น ผลลัพธ์ถูกต้อง จึงสามารถอนุมานได้ว่า กรณีทดสอบระดับหน่วย TS4 นั้นสามารถทำให้ ModuleAD ทำงานได้

ซึ่งจากกรณีทดสอบระดับหน่วยนั้น แสดงว่าโมดูลทุกๆโมดูลนั้นสามารถทำงานได้ แต่ไม่ได้หมายความว่าถ้าทำงานร่วมกับโมดูลอื่นๆจะยังสามารถทำงานได้ดังเดิม

TestID	Class	Module	TestCase	Expected Result	Result
TS1	Class_A	ModuleAA	null	complete	complete
TS2	Class_A	ModuleAB	‘a’	‘a’	‘a’
TS3	Class_A	ModuleAC	“Hello”	“Hello”	“Hello”
TS4	Class_A	ModuleAD	5,5	10	10
TS5	Class_A	ModuleAE	5f	5f	5f
TS6	Class_A	ModuleAF	false	false	false

ตารางที่ 3.2 ตารางแสดงตัวอย่างกรณีทดสอบระดับหน่วย

ดังนั้นจึงมีความเป็นไปได้ในการใช้กรณีทดสอบระดับหน่วยมาใช้สร้างกรณีทดสอบระดับรวมหน่วยโดยใช้วิธีการจับคู่ระหว่างโมดูลที่ถูกเรียกใช้จากโครงสร้างต้นไม้วากยสัมพันธ์แบบนามธรรมที่ปรับปรุงขึ้นมา กับกรณีทดสอบระดับหน่วย จากนั้นนำข้อมูลที่ได้มาสร้างเป็นกรณีทดสอบระดับรวมหน่วยดังตารางที่ 3.3

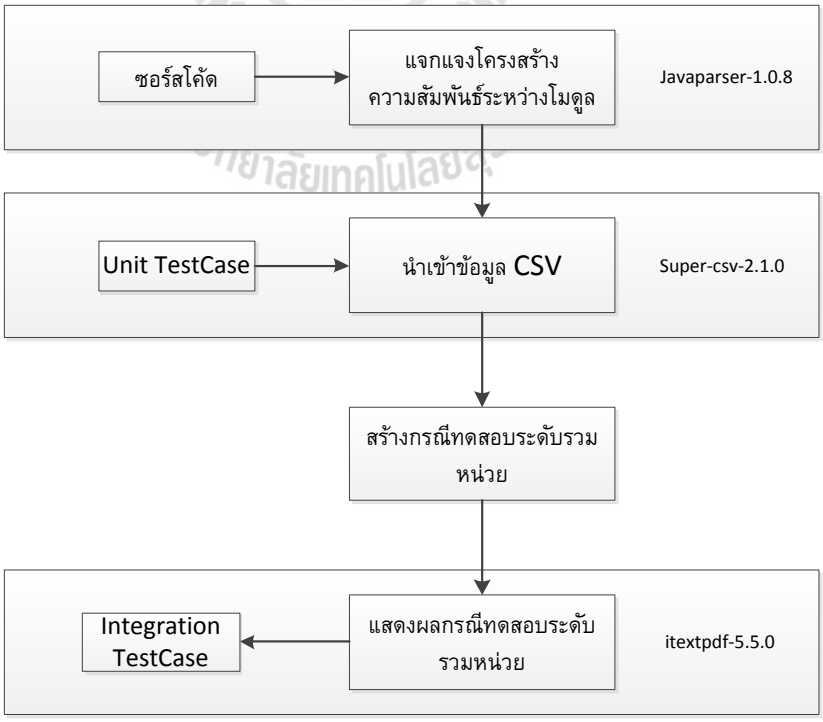
TestID	Class	Module Call	Module	TestCase	Expected Result	Result
TS1	Class_A	ModuleAA	ModuleAD	5,5	10	
TS2	Class_A	ModuleAA	ModuleAE	5f	5f	
TS3	Class_A	ModuleAC	ModuleAF	false	false	

ตารางที่ 3.3 ตารางแสดงตัวอย่างกรณีทดสอบระดับรวมหน่วย

จากตารางที่ 3.3 จะเป็นผลลัพธ์สุดท้ายจากจากกระบวนการทั้งหมด จะเป็นข้อมูลที่สามารถนำไปใช้ทดสอบกับการทดสอบระดับรวมหน่วยได้ เนื่องจากการทดสอบระดับรวมหน่วยนั้นจะเน้นไปที่การตรวจสอบโมดูลที่มีความสัมพันธ์กันทำงานร่วมกันได้หรือไม่ และมีข้อผิดพลาดใดๆอยู่หรือไม่

3.1.4 การออกแบบและพัฒนาเครื่องมือสำหรับสร้างกรณีทดสอบระดับรวมหน่วยแบบอัตโนมัติ

จากข้อมูลที่ได้กล่าวมาทั้งหมด ผู้วิจัยมีแนวคิดที่จะสร้างเครื่องมือที่จะช่วยสร้างกรณีทดสอบระดับรวมหน่วยแบบอัตโนมัติโดยมีการลำดับการศึกษาดังรูปที่ 3.9



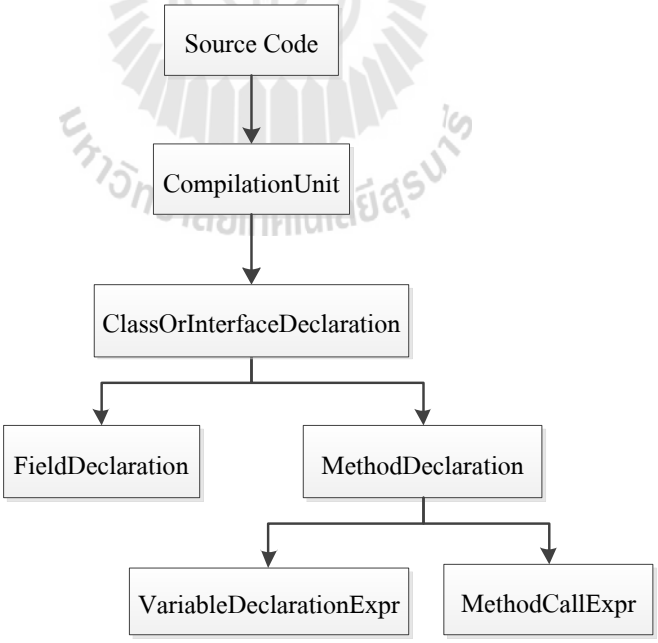
รูปที่ 3.9 รูปแสดงลำดับการพัฒนาเครื่องมือและไลบรารีที่ใช้

จากรูปที่ 3.9 ซึ่งได้แสดงถึงลำดับการพัฒนาเครื่องมือและไลบรารีที่ใช้ในการพัฒนา โดยแบ่งออกเป็น 4 ส่วนดังนี้

3.1.4.1 การพัฒนากระบวนการแจกแจงโครงสร้างความสัมพันธ์ระหว่างโมดูล

ผู้วิจัยได้เริ่มจากออกแบบกระบวนการแจกแจงโครงสร้างความสัมพันธ์ระหว่างโมดูล จากนั้นได้ทดลองใช้ไลบรารี javaparser-1.0.8.jar ในการประยุกต์ใช้แจกแจงโครงสร้างความสัมพันธ์ระหว่างโมดูล สามารถดาวน์โหลดได้ที่ https://code.google.com/p/javaparser/ ผลลัพธ์ออกมาเป็นที่น่าพอใจ javaparser-1.0.8 สามารถเข้าถึงและแจกแจงโครงสร้างของซอร์สโค้ดของภาษาจาวาได้ แต่ผู้วิจัยก็พบข้อบกพร่องของ ไลบรารีนี้คือ การเข้าถึงองค์ประกอบต่างๆภายในซอร์สโค้ดนั้นยังไม่สมบูรณ์ เช่น ไม่สามารถระบุประเภทตัวแปรในส่วนอาร์กิวเมนต์ของโมดูล แต่ก็สามารถแก้ไขได้โดยการสร้างฟังก์ชั่นเสริมสำหรับระบุประเภทของตัวแปร

ในการพัฒนากระบวนการแจกแจงโครงสร้างระหว่างโมดูลเป็นขั้นตอนที่มีความซับซ้อนและใช้เวลาในการพัฒนามากที่สุดในการพัฒนาเครื่องมือนี้ หลักจากผู้พัฒนาได้ศึกษาไลบรารี javaparser-1.0.8.jar จะมีคลาสที่จำเป็นต่อการแจกแจงความสัมพันธ์ระหว่างโมดูลดังนี้



รูปที่ 3.10 รูปแสดงโครงสร้างที่จำเป็นในการพัฒนาเครื่องมือจาก javaparser-1.0.8.jar

CompilationUnit จะเป็นออบเจ็กต์ที่ได้หลังการแจงซอร์สโค้ดแล้ว ออบเจ็กต์นี้จะรวบรวมข้อมูลต่างๆของซอร์สโค้ด 1 ไฟล์

ClassOrInterfaceDeclaration เป็นออบเจ็กต์ ที่ได้หลังจากใช้ ออบเจ็กต์ของ CompilationUnit มาผ่านเมท็อดชื่อ visit จากคลาส VoidVisitorAdapter ซึ่งเมท็อด visit นี้จะทำหน้าที่เป็นตัวดึงข้อมูลของคลาสภายในซอร์สโค้ดที่จำเป็นต้องใช้ในการทดสอบ ออกมาในรูปแบบออบเจ็กต์ของ ClassOrInterfaceDeclaration ออบเจ็กต์นี้จะเก็บข้อมูลภายในหลักๆคือ ชื่อของคลาส ชื่อของคลาสที่มีการสืบทอด (extends) และชื่อของคลาสที่อิมพลีเมนท์ (Implement)

FieldDeclaration เป็นออบเจ็กต์ทำหน้าที่ในการเข้าถึงการประกาศตัวแปรต่างๆที่ประกาศไว้ภายในคลาส ออบเจ็กต์นี้จำเป็นที่จะต้องใช้ เมท็อดชื่อ visit จากคลาส VoidVisitorAdapter โดยรับอาร์กิวเมนต์เป็นออบเจ็กต์ของ ClassOrInterfaceDeclaration

MethodDeclaration ออบเจ็กต์ของคลาสนี้จะทำหน้าที่เก็บข้อมูลเมท็อดภายในคลาสของซอร์สโค้ดที่ใช้ในการทดสอบ ประกอบไปด้วยข้อมูลหลักๆ เช่น ชื่อของเมท็อด พารามิเตอร์ของเมท็อด ประเภท (Type) ของเมท็อด รวมถึงซอร์ดโค้ดภายในเมท็อดด้วย การสร้างออบเจ็กต์นี้จะต้องใช้ ออบเจ็กต์ของ ClassOrInterfaceDeclaration ในการสร้างด้วย

VariableDeclarationExpr ออบเจ็กต์ของคลาสนี้จะเก็บข้อมูลของตัวแปรที่ประกาศอยู่ภายใน เมท็อด ใช้ออบเจ็กต์ของ MethodDeclaration ในการสร้าง

MethodCallExpr ออบเจ็กต์นี้จะเก็บข้อมูลของเมท็อดที่ถูกเรียกภายในเมท็อดที่เรากำลังแจงข้อมูล โดยใช้ออบเจ็กต์ของ MethodDeclaration ในการสร้าง

จากคลาสที่ใช้ในการพัฒนาเครื่องมือนี้ในรูปที่ 3.10 ซึ่งก่อนจะได้ออบเจ็กต์ที่ต้องการจะต้องมีการใช้คลาส VoidVisitorAdapter ในการเข้าถึงข้อมูลและสร้างออบเจ็กต์ต่างๆขึ้นมาทำให้ลำดับขั้นตอนนั้นเกิดความซับซ้อนขึ้น เมื่อนำมาประยุกต์ใช้ เพื่อให้การจัดการข้อมูลได้ง่ายขึ้นผู้วิจัยจึงได้สร้างคลาสมาเพื่อเก็บข้อมูลจากออบเจ็กต์ต่างๆเหล่านี้

ความท้าทายของการพัฒนากระบวนการแจกแจงโครงสร้างระหว่างโมดูลนี้คือการระบุว่าโมดูลใดเรียกโมดูลใด จากออบเจ็กต์ของ MethodCallExpr จะมีข้อมูลเพียง ชื่อของเมท็อด และค่าของอาร์กิวเมนต์ที่โมดูลนั้นได้รับ ทำให้ไม่สามารถแยกแยะได้ว่าโมดูลที่ถูกเรียกนั้นเป็นโมดูลใดภายในคลาสและโมดูลนั้นได้เรียกโมดูลใดๆอีกหรือไม่ ผู้วิจัยได้แก้ปัญหาโดยการสร้างโมดูลสำหรับตรวจเช็คโมดูลที่มีอยู่ในคลาสและโมดูลที่ถูกเรียก ทำให้สามารถระบุตัวตนของโมดู,ที่ถูกเรียก อีกทั้งยังช่วยให้แยกแยะโมดูลที่ถูกสร้างภายในคลาสและโมดูลที่มาจากไลบรารีพื้นฐานของจาวา

3.1.4.2 การพัฒนากระบวนการนำเข้าข้อมูล

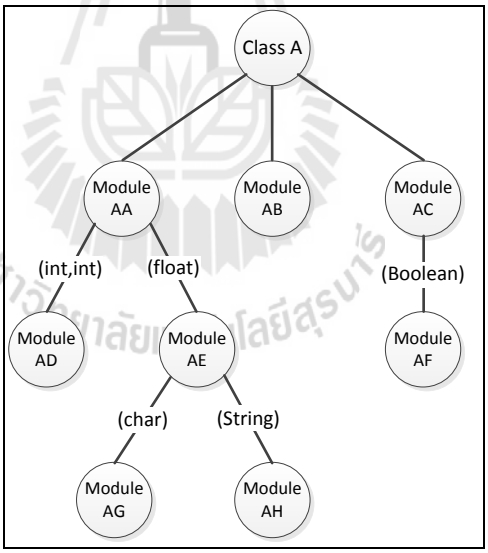
กระบวนการนำเข้าข้อมูลโดยข้อมูลที่นำเข้ามานั้นเป็นกรณีทดสอบระดับหน่วย อยู่ในรูปแบบของไฟล์ CSV การนำเข้าไฟล์สกุล CSV ของภาษาจาวานั้นสามารถทำได้หลายวิธี ผู้วิจัยได้เลือกใช้ไลบรารี Super-csv-2.1.0.jar

เนื่องจาก Super CSV เป็นไลบรารีที่ไม่ค่อยซับซ้อนและนำมาใช้ได้โดยไม่มีค่าใช้จ่าย ไลบรารีนี้สามารถเข้าไปดาวน์โหลดได้ที่ http://supercsv.sourceforge.net/index.html

เมื่อข้อมูลถูกนำเข้ามาในระบบแล้วจะสร้างออบเจ็กต์สำหรับรองรับข้อมูล โดยแต่ละ Attribute ของออบเจ็กต์ คือค่าแต่ละคอลัมน์ของข้อมูลนำเข้า ออบเจ็กต์เหล่านี้จะถูกเก็บอยู่ในรูปแบบของ ArrayList

3.1.4.3 การพัฒนากระบวนการสร้างกรณีทดสอบระดับรวมหน่วย

ในการพัฒนากระบวนการสร้างกรณีทดสอบระดับรวมหน่วยนั้น เริ่มจากการใช้โครงสร้างต้นไม้สำหรับแจกแจงความสัมพันธ์ระหว่างโมดูลเพื่อกำหนดโมดูลที่จะใช้ในการสร้างกรณีทดสอบ

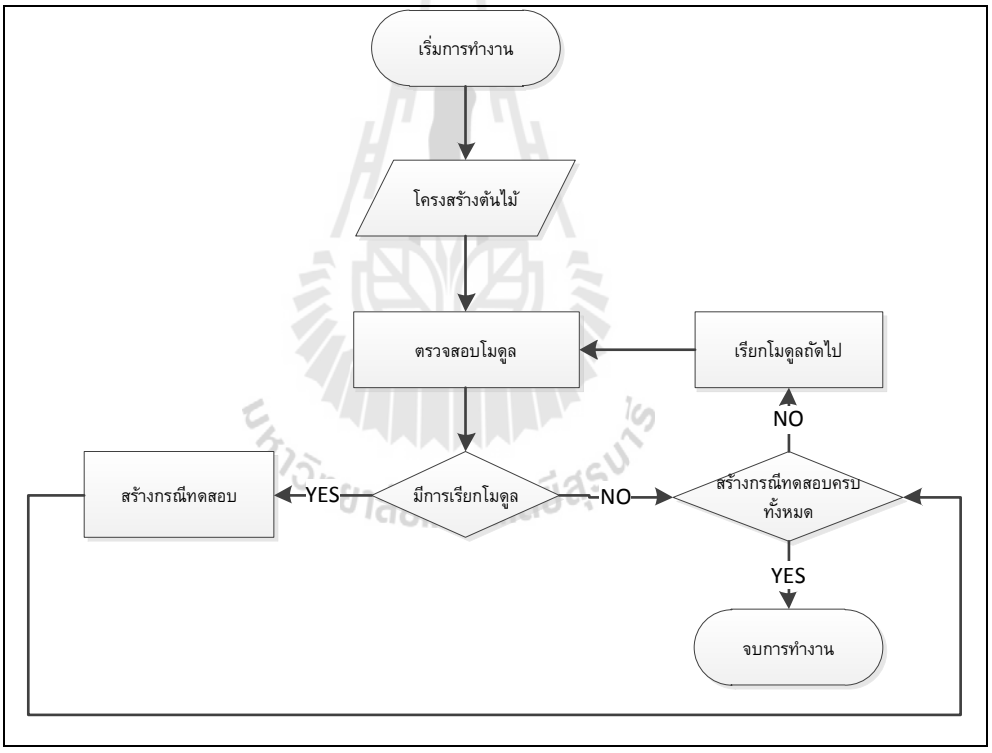


รูปที่ 3.11 รูปแสดงตัวอย่างโครงสร้างต้นไม้สำหรับแจกแจงความสัมพันธ์ระหว่างโมดูล

จากรูปที่ 3.11 เป็นตัวอย่างโครงสร้างต้นไม่สำหรับแจกแจงความสัมพันธ์ระหว่างโมดูล โดยการสร้างกรณีทดสอบนั้นจะใช้วิธีการแบบเพิ่มทีละหน่วย (Incremental Testing) วิธีการทดสอบนี้สามารถแบ่งออกได้เป็น 2 วิธีคือ บนลงล่าง(Top-Down)และล่างขึ้นบน (Bottom-Up) ผู้วิจัยได้เลือกใช้วิธีการทดสอบแบบบนลงล่างมาประยุกต์เพิ่มเติมเพื่อให้

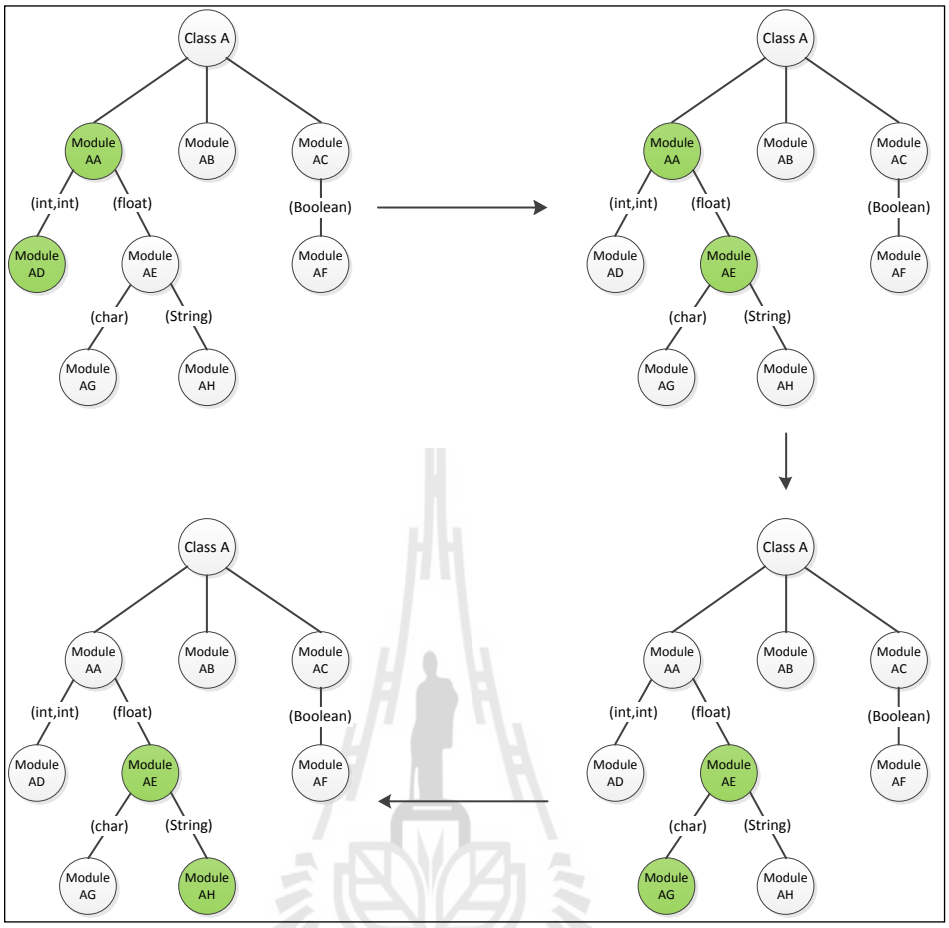
เหมาะสมกับกระบวนการทดสอบ โดยการทดสอบแบบบนลงล่างนี้ เนื่องจากงานวิจัยนี้ได้มุ่งเน้นไปที่การตรวจสอบว่าโมดูลที่มีความสัมพันธ์กันนั้นเมื่อมาทำงานร่วมกันจะยังคงสามารถทำงานได้หรือไม่

เริ่มจากโหนดของโมดูลที่อยู่ระดับสูงสุด จากนั้นตรวจสอบว่าโหนดของโมดูลดังกล่าวได้มีการเรียกโมดูลอื่นหรือไม่ ถ้ามีจะเริ่มกระบวนการสร้างกรณีทดสอบ หลังจากสร้างกรณีทดสอบสำหรับโมดูลที่ถูกกเรียกเสร็จแล้ว จะทำการตรวจสอบโมดูลที่ถูกเรียกนั้นว่ามีการเรียนโมดูลอื่นอีกหรือไม่ ถ้ามี จะสร้างกรณีทดสอบต่อ ถ้าไม่มีจะเลื่อนไปยังโมดูลต่อไปเรียงลำดับจากบนลงล่าง ในกรณีที่โมดูลที่ถูกเรียกนั้นไม่มีการเรียกใครต่อแล้วจะกลับขึ้นไปตรวจสอบว่าโมดูลผู้เรียกนั้นเรียกโมดูลอื่นหรือไม่ ถ้าเรียกก็จะเข้าสู่กระบวนการสร้างกรณีทดสอบต่อถ้าไม่ก็จะกลับขึ้นไป ทำเช่นนี้จนครบทุกโมดูล



รูปที่ 3.12 รูปแสดงกระบวนการจับคู่โมดูลเพื่อลำดับการสร้างกรณีทดสอบ

การสร้างกรณีทดสอบนั้นจะใช้วิธีดังที่ได้กล่าวไว้ในหัวข้อที่ 3.1.3 แล้วโดยการจับคู่โมดูล ตรวจสอบโมดูลทีถูกเรียก เปรียบเทียบโมดูลที่ถูกเรียกจากกรณีทดสอบระดับหน่วย จากนั้นนำกรณีทดสอบที่พบมาสร้างกรณีทดสอบระดับรวมหน่วย



รูปที่ 3.13 รูปแสดงตัวอย่างลำดับการสร้างกรณีทดสอบระดับรวมหน่วย

3.1.4.3 การพัฒนากระบวนการสร้างรายงาน

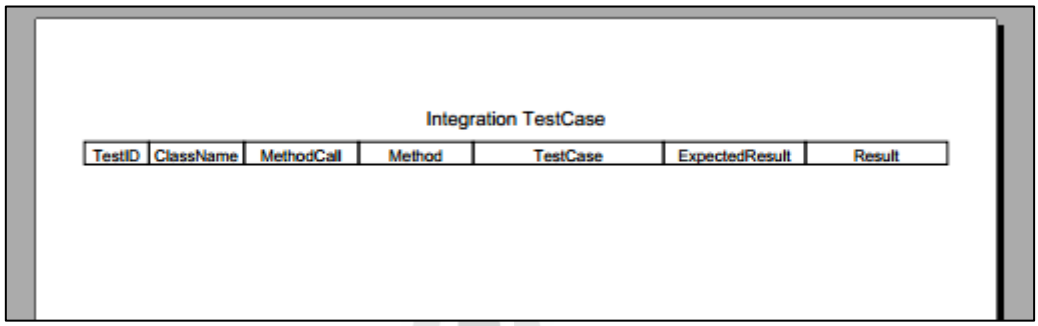
การสร้างรายงานนั้นผู้วิจัยได้เลือกรายงานที่เป็นไฟล์สกุล PDF (Portable Document Format) เนื่องจากไฟล์ PDF นั้นได้รับความนิยมใช้กันอย่างแพร่หลาย จุดเด่นหลักๆคือสามารถรักษารูปแบบการจัดวางข้อมูลได้แม้ว่าโปรแกรมที่เปิดดูข้อมูลจะต่างกัน ซึ่งรายงานที่ผู้วิจัยต้องการสร้างนั้นเป็นรูปแบบตาราง เพื่อป้องกันไม่ให้ข้อมูลผิดเพี้ยนจากรูปแบบที่ได้ออกแบบไว้จึงได้เลือกใช้ไฟล์ PDF

ภาษาจาวานั้นมีไลบรารีมากมายที่สามารถสร้างไฟล์ PDF ได้ ผู้วิจัยได้เลือก itextpdf-5.5.0.jar ไลบรารีนี้เป็นที่นิยมสำหรับนักพัฒนาที่ต้องการจะออกรายงานเป็นไฟล์ PDF เนื่องจากเป็นไลบรารีที่มีประสิทธิภาพ การนำไปประยุกต์ใช้งานไม่ซับซ้อน เป็นต้น สามารถดาวน์โหลดได้ที่ http://itextpdf.com/

การนำไลบรารี itextpdf-5.5.0.jar มาใช้งานนั้นสามารถทำได้ง่าย แต่ต้องมีการกำหนดตำแหน่งของข้อมูลต่างๆที่ต้องการให้ชัดเจน ยกตัวอย่างเช่น

Document document = new Document(PageSize.A4, 50, 50, 50, 70);

เป็นการกำหนดขนาดกระดาษ และระยะห่างจากขอบทั้ง 4 ด้าน มีหน่วยเป็น พิกเซล (pixel)table.setWidths(new float[]{40, 59, 71, 71, 118, 88, 88}); เป็นการกำหนดขนาดของแต่ละคอลัมน์ มีหน่วยเป็น พิกเซล เป็นต้น



รูปที่ 3.14 รูปแสดงตัวอย่างผลการสร้างไฟล์ PDF ที่ยังไม่ได้ใส่ข้อมูล

3.2 เครื่องมือที่ใช้ในงานวิจัย

ในส่วนนี้จะกล่าวถึงเครื่องมือที่ใช้ในการพัฒนาและทดสอบ โดยมีรายละเอียดดังนี้

3.2.1 เครื่องมือที่ใช้ในการพัฒนาและทดสอบ

1) เครื่องคอมพิวเตอร์สำหรับการพัฒนา มีรายละเอียดดังนี้

- หน่วยประมวลผลกลาง : Intel Core i7
- หน่วยความจำสำรอง : HDD 500 GB SSD 128 GB
- หน่วยความจำหลัก : 8 GB
- อุปกรณ์เสริมอื่นๆ เช่น เมาส์ แป้นพิมพ์ เป็นต้น

2) ระบบปฏิบัติการและโปรแกรมประยุกต์สำหรับการพัฒนา ประกอบไปด้วย

- ระบบปฏิบัติการ : Windows 7 64 bit
- เครื่องมือที่ใช้ในการพัฒนา : NetBeans 7.3.1
- JDK : 1.7.0_40

บทที่ 4
การทดสอบและอภิปรายผล

การสร้างกรณีทดสอบสำหรับการทดสอบระดับรวมหน่วยแบบเพิ่มทีละหน่วยโดยอัตโนมัติจากกรณีทดสอบระดับหน่วยนั้นสามารถสร้างกรณีทดสอบระดับรวมหน่วยแบบอัตโนมัติได้ การทดสอบของเครื่องมือนี้มีรายละเอียดของหัวข้อต่างๆดังนี้ หัวข้อที่ 4.1 การทดสอบการแจกแจงโครงสร้างความสัมพันธ์ระหว่างโมดูล หัวข้อนี้เป็นการทดสอบประสิทธิภาพของการแจกแจงโครงสร้างความสัมพันธ์ระหว่างโมดูลในโมดูลที่มีรูปแบบต่างๆ เพื่อพิสูจน์ว่าเครื่องมือนี้สามารถแจกแจงโครงสร้างความสัมพันธ์ระหว่างโมดูลในรูปแบบต่างๆได้ หัวข้อที่ 4.2 การทดสอบการสร้างกรณีทดสอบระดับรวมหน่วย หัวข้อนี้จะทดสอบเกี่ยวกับการสร้างกรณีทดสอบระดับรวมหน่วย เพื่อวัดประสิทธิภาพทางด้านเวลาและพิสูจน์ว่าเครื่องมือนี้สามารถสร้างกรณีทดสอบระดับรวมหน่วยและสามารถนำไปใช้ประโยชน์ได้ หัวข้อที่ 4.3 อภิปรายผล จะสรุปผลการทดสอบต่างๆของเครื่องมือนี้

4.1 การทดสอบการแจกแจงโครงสร้างความสัมพันธ์ระหว่างโมดูล

การแจกแจงโครงสร้างความสัมพันธ์ระหว่างโมดูลนั้นเป็นขั้นตอนที่มีความสำคัญมากต่อเครื่องมือนี้ เนื่องจากการทดสอบระดับรวมหน่วยนั้นจำเป็นจะต้องมีโครงสร้างของโมดูลที่ถูกต้องทำให้การลำดับโมดูลเพื่อสร้างกรณีทดสอบถูกต้องไปด้วย การทดสอบการแจงแจงโครงสร้างความสัมพันธ์นั้นจะเน้นที่การทดสอบความถูกต้องการแจงแจงโครงสร้างความสัมพันธ์ระหว่างโมดูลในรูปแบบต่างๆ

4.1.1 การระบุโมดูลที่ถูกประกาศในรูปแบบพื้นฐาน

การแจกแจงโครงสร้างความสัมพันธ์ระหว่างโมดูลนั้นจะต้องแจกแจงโมดูลได้หลายรูปแบบ โดยโมดูลที่ใช้ในการทดสอบการแจกแจงจะใช้โมดูลที่ถูกประกาศในรูปแบบพื้นฐานประกอบไปด้วย

1. โมดูลที่ไม่มีพารามิเตอร์และไม่มีการคืนค่า
2. โมดูลที่มีพารามิเตอร์และไม่มีการคืนค่า
3. โมดูลที่ไม่มีพารามิเตอร์และมีการคืนค่า

4. โมดูลที่มีพารามิเตอร์และมีการคืนค่า

ซอร์สโค้ดดังกล่าวแสดงอยู่ในรูปที่ 4.1

```
public class Class {
    public void Module_A() {
    }

    public int Module_B() {
        return 6;
    }

    public void Module_C(int a) {
    }

    public String Module_D(char c,double d) {
        return "complete";
    }
}
```

รูปที่ 4.1 ซอร์สโค้ดของโมดูลในรูปแบบพื้นฐานที่ใช้ทดสอบ

เมื่อนำซอร์สโค้ดดังกล่าวมาทดสอบการแจกแจงความสัมพันธ์ระหว่างโมดูลของเครื่องมือนี้เพื่อตรวจสอบว่าเครื่องมือนี้สามารถเข้าถึงและระบุข้อมูลของโมดูลดังกล่าวได้ ผลลัพธ์ที่ได้ดังรูปที่ 4.2

```
ClassName: Class
 void Module_A null from Class
 int Module_B null from Class
 void Module_C [int] from Class
 String Module_D [char, double] from Class
```

รูปที่ 4.2 ผลลัพธ์การแจกแจงความสัมพันธ์ระหว่างโมดูลของโมดูลในรูปแบบพื้นฐานจากรูปที่ 4.1

จากรูปที่ 4.2 ผลลัพธ์ที่ได้นั้นให้ข้อมูลที่ตรงกันกับซอร์สโค้ดที่ใช้ในการทดสอบ ทำให้เครื่องมือนี้สามารถระบุโมดูลที่ถูกประกาศในรูปแบบพื้นฐานได้

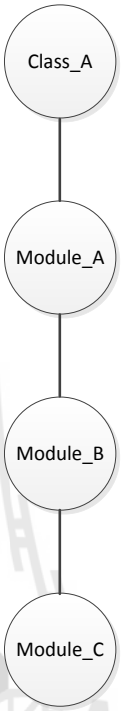
4.1.2 การระบุโมดูลที่มีความซับซ้อนและมีความสัมพันธ์ภายในคลาสเดียวกัน

โมดูลที่มีความซับซ้อนจำเป็นที่จะต้องนำมาทดสอบเพื่อตรวจสอบว่าเครื่องมือนี้สามารถเข้าถึงซอร์สโค้ดและสกัดข้อมูลที่จำเป็นได้หรือไม่ โดยใช้ซอร์สโค้ดในรูปที่ 4.3

```
public class Class_A {

    public void Module_A() {
        Random r = new Random();
        int a = r.nextInt(100) + 1;
        int b = r.nextInt(100) + 1;
        System.out.println("This is Module_A");
        for (int i = a; i <= b; i++) {
            int randomInt = r.nextInt(100) + 1;
            if (randomInt % 2 == 0) {
                System.out.println(randomInt + "is even number");
            } else {
                System.out.println(randomInt + "is ood number");
            }
            int GCD = Module_B(a, b);
            System.out.println("GCD of " + a + " " + b + " is " + GCD);
        }
        int GCD = Module_B(a, b);
        System.out.println("GCD of " + a + " " + b + " is " + GCD);
    }

    public int Module_B(int a, int b) {
        if (a == 0) {
            return b;
        }
        while (b != 0) {
            if (a > b) {
                a = a - b;
            } else {
                b = b - a;
            }
        }
        int LCM = Module_C(a, b);
        System.out.println("LCM of " + a + " " + b + " is " + LCM);
        return a;
    }

    public int Module_C(int a, int b) {
        int j = 1;
        int val = a * b;
        for (int i = 2; i < val; i++) {
            if ((a % i == 0) && (b % i == 0)) {
                j = i;
            }
        }
        int lcm = val / j;
        System.out.println("LCM=" + lcm);
        return lcm;
    }
}
```

รูปที่ 4.3 รูปซอร์สโค้ดที่ใช้ในการทดสอบการแจกแจงความสัมพันธ์ของโมดูลที่มีความซับซ้อน

หากพิจารณาความสัมพันธ์ระหว่างโมดูลจากซอร์สโค้ดในรูปที่ 4.3 จะเห็นว่าความสัมพันธ์ระหว่างโมดูลนั้นมีความสัมพันธ์ต่อกันเป็นทอดๆดังรูปที่ 4.4



รูปที่ 4.4 รูปโมเดลแสดงความสัมพันธ์ระหว่างโมดูลของรูปที่ 4.3

เมื่อทดสอบซอร์สโค้ดในรูปที่ 4.3 กับเครื่องมือเพื่อทดสอบการแจกแจงความสัมพันธ์ระหว่างโมดูลเพื่อตรวจสอบการเข้าถึงและแจกแจงความสัมพันธ์จากซอร์สโค้ดที่มีความซับซ้อนได้ผลลัพธ์ดังภาพที่ 4.5

```
ClassName: Class_A
 void Module_A null from Class_A
  int Module_B [int, int] from Class_A
   int Module_C [int, int] from Class_A
```

รูปที่ 4.5 ผลลัพธ์การแจกแจงความสัมพันธ์ระหว่างโมดูลของซอร์สโค้ดในรูปที่ 4.3

จากรูปที่ 4.5 ผลลัพธ์ที่ได้นั้นเป็นไปตามโครงสร้างของซอร์สโค้ดในรูปที่ 4.4 สรุปได้ว่าเครื่องมือนี้มีความสามารถในการแจกแจงความสัมพันธ์ระหว่างโมดูลที่มีความสัมพันธ์ภายในคลาสเดียวกันได้

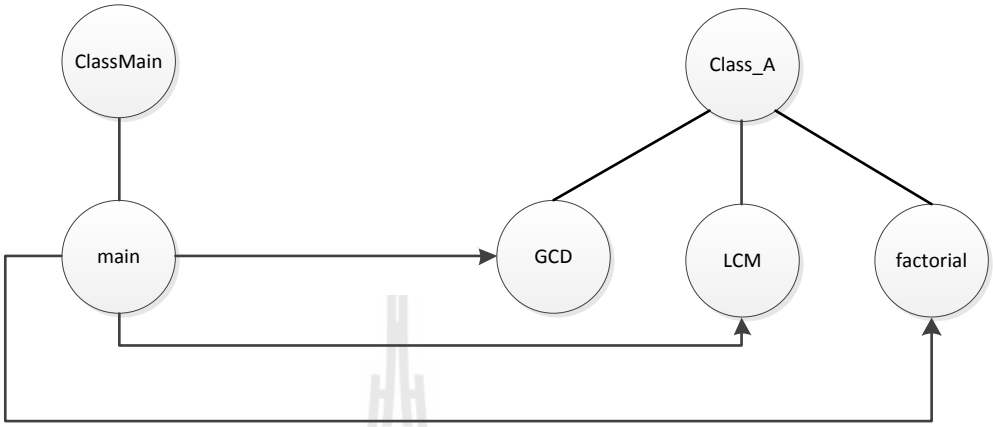
4.1.3 การระบุโมดูลที่มีความซับซ้อนและมีความสัมพันธ์ระหว่างคลาส

จากการทดสอบก่อนหน้านี้เป็นการทดสอบการแจกแจงความสัมพันธ์ระหว่างโมดูลที่มีความซับซ้อนแต่อย่างไรก็ตามความสัมพันธ์ดังกล่าวนั้นยังคงเป็นความสัมพันธ์ที่เกิดขึ้นภายในคลาสเดียวกัน ดังนั้นผู้วิจัยจึงได้ทดสอบกับซอร์สโค้ดที่มีความซับซ้อนและมีความสัมพันธ์ระหว่างคลาสดังภาพที่ 4.6

```
public class ClassMain {
    public static void main(String[] args) {
       Class_A a = new Class_A();
       a.GCD(10, 7);
       a.LCM(5, 9);
       a.factorial(10);
    }
}
public class Class_A {
    public int GCD(int a, int b) {
        if (a == 0) {
            return b;
        }
        while (b != 0) {
            if (a > b) {
                a = a - b;
            } else {
                b = b - a;
            }
        }
        return a;
    }

    public int LCM(int a, int b) {
        int j = 1;
        int val = a * b;
        for (int i = 2; i < val; i++) {
            if ((a % i == 0) && (b % i == 0)) {
                j = i;
            }
        }
        int lcm = val / j;
        return lcm;
    }

    public int factorial(int n) {
        int result = 1;
        for (int i = 1; i <= n; i++) {
            result = result * i;
        }
        return result;
    }
}
```

รูปที่ 4.6 ซอร์สโค้ดที่มีความซับซ้อนแบบมีความสัมพันธ์ระหว่างคลาส

เมื่อพิจารณาซอร์สโค้ดในรูปที่ 4.6 จะเห็นว่าความสัมพันธ์ที่เกิดขึ้นนั้นเป็นความสัมพันธ์ระหว่าง 2 คลาส เมื่อสร้างโมดูลความสัมพันธ์ด้วยมือจะได้ดังรูปที่ 4.7



รูปที่ 4.7 รูปโมเดลแสดงความสัมพันธ์ระหว่างโมดูลของรูปที่ 4.6

หลักจากนำซอร์สโค้ดที่มีความซับซ้อนแบบมีความสัมพันธ์ระหว่างคลาสจากรูปที่ 4.6 มาผ่านกระบวนการแจกแจงความสัมพันธ์ระหว่างโมดูลได้ผลลัพธ์ดังรูปที่ 4.8

```
ClassName: ClassMain
 void main [String[]] from ClassMain
  int GCD [int, int] from Class_A
  int LCM [int, int] from Class_A
  int factorial [int] from Class_A
ClassName: Class_A
 int GCD [int, int] from Class_A
 int LCM [int, int] from Class_A
 int factorial [int] from Class_A
```

รูปที่ 4.8 ผลลัพธ์การแจกแจงความสัมพันธ์ระหว่างโมดูลของซอร์สโค้ดในรูปที่ 4.6

จากรูปที่ 4.8 ผลลัพธ์ที่ได้นั้นถูกต้องตามโครงสร้างของซอร์สโค้ดในรูปที่ 4.6 โดยที่ภายในซอร์สโค้ดนั้นประกอบไปด้วยคลาส 2 คลาส และมีการเรียกใช้โมดูลข้ามคลาส ซึ่งโมดูลที่มีการเรียกนั้นมีความซอบซ้อนระดับหนึ่ง และเครื่องมือนี้ก็สามารถแจกแจงได้ถูกต้องตามโครงสร้าง รวมถึงสามารถเข้าถึงองค์ประกอบต่างๆของโมดูลได้

4.1.4 การแจกแจงความสัมพันธ์ระหว่างโมดูลที่มีความซับซ้อนระดับหนึ่ง

การทดสอบการแจกแจงความสัมพันธ์ระหว่างโมดูลที่มีความซับซ้อนนั้นเป็นส่วนที่สำคัญ เนื่องจากการพัฒนาโปรแกรมในปัจจุบันมักจะมีความซับซ้อนมาก ผู้วิจัยได้ออกแบบซอร์สโค้ดที่ใช้ในการทดสอบเพื่อให้ครอบคลุมความสัมพันธ์ระหว่างโมดูลในหลายๆ กรณี เพื่อพิสูจน์ความถูกต้องของเครื่องมือนี้

ซอร์สโค้ดที่ใช้ทดสอบในส่วนนี้นั้นประกอบไปด้วย คลาสจำนวน 5 คลาส โมดูลรวมกัน 21 โมดูล ดังรูปที่ 4.9 ถึงรูปที่ 4.13

```
public class Class_A {

    public String Module_AA(String a) {
        return a += a;
    }

    public void Module_AB() {
                System.out.println("complete");
    }

    public void Module_AC(float a) {
        Module_AD(5, 'a');
        Module_AE(a);
    }

    public char Module_AD(int a, char b) {
        b += a;
        Module_AF();
        return b;
    }

    public float Module_AE(float a) {
        Module_AG("aa");
        return a;
    }

    public void Module_AF() {
        System.out.println("complete");
    }

    public void Module_AG(String a) {
        System.out.println(a);
    }
}
```

รูปที่ 4.9 รูปซอร์สโค้ดที่มีความสัมพันธ์ซับซ้อน (1)

```
public class Class_B extends Class_A {
    public long Module_BA(long a){
        Module_AE(50f);
        Module_AA("a");
        Module_AA("a");
        return a/2;
    }
    public void Module_BB(){
         System.out.println("complete");
    }
    public String Module_BC(int a){

        return String.valueOf(a);
    }
    @Override
    public float Module_AE(float a){
        Module_BC(5);
        return a+1;
    }
}
```

รูปที่ 4.10 รูปซอร์สโค้ดที่มีความสัมพันธ์ซับซ้อน (2)

```
public class Class_C {
    public void Module_CA(double a,String b){
        Module_CC(true);
         System.out.println(a + "," + b);
    }
    public void Module_CB(int a,int b){
        Module_CC(5,5);
        Module_CD();
         System.out.println(a + "," + b);
    }
    public boolean Module_CC(boolean a){
        return a;
    }
    public int Module_CC(int a,int b){
        return a+b;
    }
    public void Module_CD(){
        System.err.println("complete");
    }
}
```

รูปที่ 4.11 รูปซอร์สโค้ดที่มีความสัมพันธ์ซับซ้อน (3)

```
public class Class_D {
    public void Module_DA(int a,String b){
        Module_DC(a,b);
        System.out.println(a+","+b);
    }
    public void Module_DB(){
        Module_DC(11,"x");
        new Class_C().Module_CC(true);
         new Class_C().Module_CC(true);
        System.out.println("complete");
    }
    public int Module_DC(int a,String b){
        return a;
    }
}
```

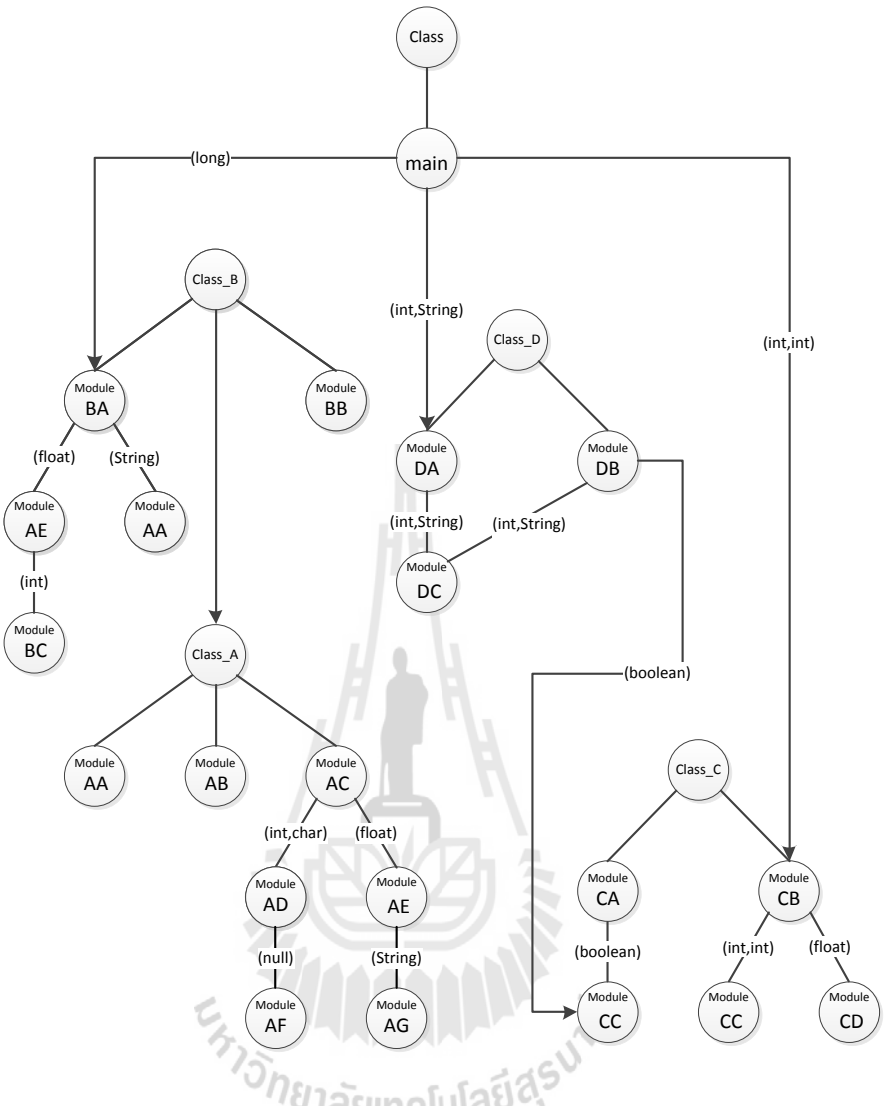
รูปที่ 4.12 รูปซอร์สโค้ดที่มีความสัมพันธ์ซับซ้อน (4)

```
public class Class {
    public static void main(String[] args) {
        new Class_D().Module_DA(20, "aaa");
        new Class_B().Module_BA(50);
        new Class_C().Module_CB(20, 50);
    }

}
```

รูปที่ 4.13 รูปซอร์สโค้ดที่มีความสัมพันธ์ซับซ้อน (5)

จากซอร์สโค้ดดังกล่าว ความสัมพันธ์ที่เกิดขึ้นนั้นมีความซับซ้อน ไม่ว่าจะเป็นความสัมพันธ์ที่เกิดขึ้นภายในคลาสเดียวกัน ความสัมพันธ์ที่เกิดขึ้นระหว่างคลาส และความสัมพันธ์ระหว่างคลาสแม่และคลาสลูก

เมื่อวิเคราะห์โครงสร้างของซอร์สโค้ดดังกล่าวสามารถสร้างโมเดลด้วยมือได้ดังภาพที่ 4.14



รูปที่ 4.14 รูปแสดงโมเดลความสัมพันธ์ระหว่างโมดูลของซอร์สโค้ดใน รูปที่ 4.9 ถึง รูปที่ 4.13

หลังจากใช้เครื่องมือที่พัฒนาขึ้นมานี้แจกแจงโครงสร้างระหว่างโมดูลจากซอร์สโค้ดในรูปที่ 4.9 ถึงรูปที่ 4.13 ได้ผลลัพธ์ดังรูปที่ 4.15

```
ClassName: Class
 void main [String[]] from Class
  void Module_DA [int, String] from Class_D
  long Module_BA [long] from Class_B
  void Module_CB [int, int] from Class_C
ClassName: Class_A
 String Module_AA [String] from Class_A
 void Module_AB null from Class_A
 void Module_AC [float] from Class_A
  char Module_AD [int, char] from Class_A
   void Module_AF null from Class_A
  float Module_AE [float] from Class_A
   void Module_AG [String] from Class_A
ClassName: Class_B extends Class_A
 long Module_BA [long] from Class_B
  float Module_AE [float] from Class_B
   String Module_BC [int] from Class_B
  String Module_AA [String] from Class_A
 void Module_BB null from Class_B
ClassName: Class_C
 void Module_CA [double, String] from Class_C
  boolean Module_CC [boolean] from Class_C
 void Module_CB [int, int] from Class_C
  int Module_CC [int, int] from Class_C
  void Module_CD null from Class_C
ClassName: Class_D
 void Module_DA [int, String] from Class_D
  int Module_DC [int, String] from Class_D
 void Module_DB null from Class_D
  int Module_DC [int, String] from Class_D
  boolean Module_CC [boolean] from Class_C
```

รูปที่ 4.15 ผลลัพธ์การแจกแจงความสัมพันธ์ระหว่างโมดูลจากซอร์สโค้ดในรูปที่ 4.9 ถึงรูปที่ 4.13

จากรูปที่ 4.14 ผลลัพธ์ที่ได้นั้นตรงกับโมเดลที่ได้สร้างไว้ก่อนหน้า ดังนั้น จากการทดสอบการแจกแจงความสัมพันธ์ระหว่างโมดูลของเครื่องมือนี้สามารถแจกแจงความสัมพันธ์ระหว่างโมดูลได้ถูกต้องตามโครงสร้างของซอร์สโค้ด ไม่ว่าจะเป็นโครงสร้างพื้นฐานของโมดูลในภาษาจาวา ซอร์สโค้ดที่มีความซับซ้อน มีความสัมพันธ์ภายในคลาสเดียวกัน และซอร์สโค้ดที่มีความซับซ้อนและมีความสัมพันธ์ระหว่างคลาส รวมถึงซอร์สโค้ดที่มีความสัมพันธ์ระหว่างคลาสที่มี

ความซับซ้อนได้ ทำให้มีประสิทธิภาพในการแจงแจงโครงสร้างความสัมพันธ์ระหว่างโมดูลเพื่อใช้ในกระบวนการสร้างกรณีทดสอบระดับรวมหน่วยต่อไป

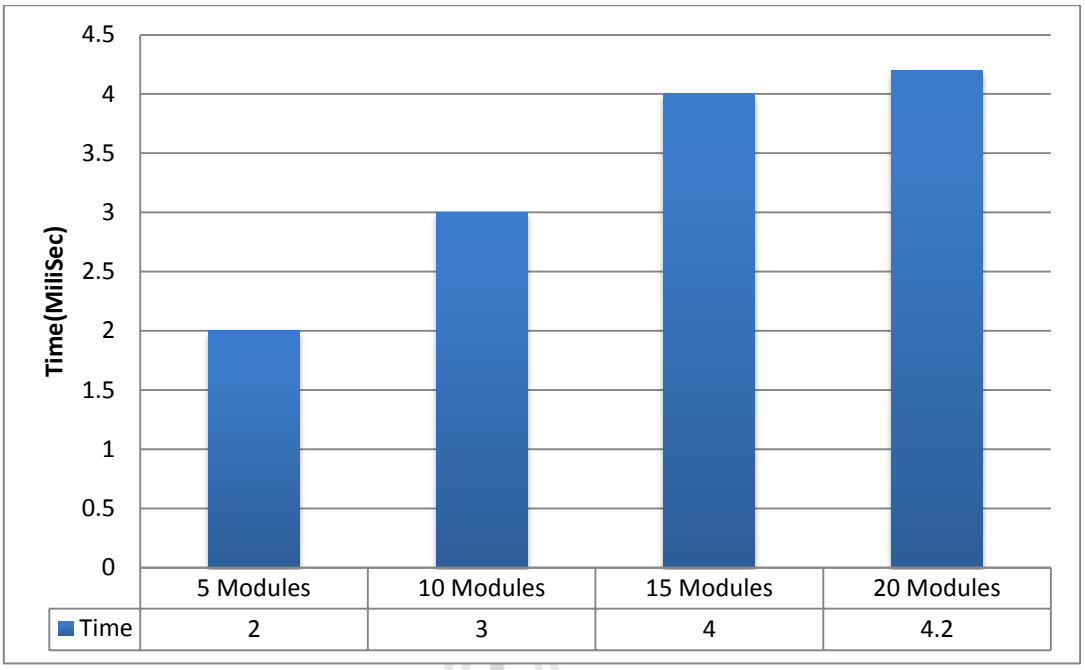
4.2 การทดสอบการสร้างกรณีทดสอบระดับรวมหน่วย

หลังจากที่ได้ทดสอบการแจกแจงความสัมพันธ์ระหว่างโมดูลไปแล้ว ผลลัพธ์ที่ได้เป็นที่น่าพอใจ แต่กระบวนการที่สำคัญอีกส่วนหนึ่งของงานวิจัยนี้คือกระบวนการสร้างกรณีทดสอบระดับรวมหน่วย การทดสอบการสร้างกรณีทดสอบระดับรวมหน่วยนี้จะมุ่งเน้นไปที่ประสิทธิภาพทางด้านเวลาการทำงาน และผลลัพธ์ที่ได้ โดยแบ่งการทดสอบออกเป็นการทดสอบประสิทธิภาพทางด้านเวลาโดยใช้ การทดสอบโมดูลที่มีความสัมพันธ์ระหว่างโมดูลจำนวนมาก และการทดสอบคลาสที่มีการเรียกคลาสอื่นจำนวนมาก โดยจำลองการสร้างกรณีทดสอบโดยใช้กรณีทดสอบระดับหน่วย ส่วนการทดสอบผผลลัพธ์จากเครื่องมือนี้ ซึ่งผลลัพธ์นั้นก็คือ กรณีทดสอบระดับรวมหน่วยจะใช้ซอร์สโค้ดดังรูปที่ 4.9 ถึง รูปที่ 4.13 ในการทดสอบ

4.2.1 การทดสอบโปรเจคที่มีความสัมพันธ์ระหว่างโมดูลจำนวนมาก

การทดสอบโมดูลที่มีความสัมพันธ์ระหว่างโมดูลจำนวนมาก โดยจะแบ่งโมดูลในการทดสอบออกเป็น 5,10,15,20 โมดูล โดยใช้กรณีทดสอบที่แสดงสถานะการทำงานของโมดูล เนื่องจากโมดูลที่ใช้ในการทดสอบนั้นเป็นโมดูลแบบไม่มีการคืนค่า แต่อย่างไรก็ตาม เครื่องมือนี้ยังคงต้องทำการแจกแจงซอร์สโค้ดและนำโครงสร้างที่ได้มาจับคู่กับกรณีทดสอบระดับหน่วยเช่นเดียวกัน

ในการทดสอบนี้ โครงสร้างซอร์ดโค้ดนั้นจะมีเพียงคลาสเดียว และมีโมดูลลักษณะเดียวกันแต่จะเพิ่มที่จำนวนของโมดูลที่มีความสัมพันธ์กัน กรณีทดสอบที่ใช้ก็จะมีจำนวนเพิ่มตามจำนวนของซอร์สโค้ดที่ใช้ในการทดสอบ เพื่อให้ทราบถึงผลกระทบระหว่างความซับซ้อนของความสัมพันธ์ระหว่างโมดูลและเวลาในการกระมวลผล



รูปที่ 4.16 แผนภูมิแสดงเวลาในการทดสอบโปรเจคที่มีความสัมพันธ์ระหว่างโมดูลจำนวนมาก

จากรูปที่ 4.16 แผนภูมิที่ได้นั้นแสดงถึงเวลาที่เพิ่มขึ้นตามจำนวนโมดูลที่มีในโปรเจค เวลาที่ใช้นั้นถือว่าน้อยมาก แม้จะมีโมดูลถึง 20 โมดูล ใช้เวลาเพียง 4.2 มิลลิวินาที

4.2.2 การทดสอบโปรเจคที่มีความสัมพันธ์ระหว่างคลาสจำนวนมาก

การทดสอบคลาสที่มีการเรียกคลาสอื่นจำนวนมาก จะเป็นการทดสอบเวลาในการประมวลผลโดยเปรียบเทียบกับจำนวนของคลาสที่มีความสัมพันธ์กัน ความสัมพันธ์ระหว่างคลาสของแต่ละโปรเจคนั้นจะแบ่งออกเป็น 5,10,15,20 คลาส แต่ละคลาสมี 1 โมดูล และเรียกต่อๆกันไป โดยแต่ละโปรเจคจะมีกรณีทดสอบ 1 ชุด กรณีทดสอบที่ใช้นั้นก็จะมีตามจำนวนโมดูลในแต่ละคลาส

รูปที่ 4.17 แผนภูมิแสดงเวลาในการทดสอบโปรเจคที่มีความสัมพันธ์ระหว่างคลาสจำนวนมาก

จากรูปที่ 4.17 แผนภูมิได้แสดงถึงเวลา เช่นเดียวกับแผนภูมิของรูปที่ 4.16 ซึ่งเวลาในการประมวลผลเพิ่มขึ้นตามจำนวนของคลาส เวลาที่ใช้ในการประมวลผลก็ถือว่าน้อยเช่นเดียวกัน แต่หากเปรียบเทียบกันแล้ว จะเห็นว่า จำนวนของคลาสนั้นมีผลต่อเวลามากกว่าจำนวนของโมดูล

4.2.3 การทดสอบโปรเจคที่มีความซับซ้อนและจำลองกรณีทดสอบ

การทดสอบนี้จะเน้นไปที่การจำลองความซับซ้อนของซอร์สโค้ดและกรณีทดสอบให้มีความซับซ้อนใกล้เคียงกับซอฟต์แวร์ที่พัฒนาขึ้นมาจริง โดยใช้การจำลองซอร์สโค้ดเพื่อให้ครอบคลุมกรณีต่างๆที่จะเกิดขึ้นให้มากที่สุด และจำลองกรณีทดสอบจากซอร์สโค้ดดังกล่าวเพื่อใช้ในการทดสอบการทำงานโดยซอร์สโค้ดที่ใช้คือรูปที่ 4.8 ถึงรูปที่ 4.12

กรณีทดสอบที่ใช้นั้นเป็นกรณีทดสอบของคลาสจำนวน 5 คลาส 21 โมดูล โดยกรณีทดสอบที่ใช้นั้นมีทั้งหมด 72 กรณีทดสอบอยู่ในรูปแบบของไฟล์ CSV รายละเอียดของกรณีทดสอบแสดงอยู่ในตารางที่ 4.1

ตารางที่ 4.1 ตารางแสดงกรณีทดสอบที่ใช้ในการทดสอบกับซอร์สโค้ดที่มีความสัมพันธ์ซับซ้อน

TestID	Class	Module	TestCase	Expected Result	Result
TS1	Class_A	Module_AA	"Hello"	"HelloHello"	"HelloHello"
TS2	Class_A	Module_AA	"123"	"123123"	"123123"
TS3	Class_A	Module_AA	"+-*/"	"+-*/+-*/"	"+-*/+-*/"

TS4	Class_A	Module_AA	"???"	"??????"	"??????"
TS5	Class_A	Module_AA	"WORLD"	"WORLDWORLD"	"WORLDWORLD"
TS6	Class_A	Module_AB	null	complete	complete
TS7	Class_A	Module_AC	3.4028235E38f	3.4028235E38f	3.4028235E38f
TS8	Class_A	Module_AC	1.4E-45f	1.4E-45f	1.4E-45f
TS9	Class_A	Module_AC	0.58313185f	0.58313185f	0.58313185f
TS10	Class_A	Module_AC	1.0768449f	1.0768449f	1.0768449f
TS11	Class_A	Module_AC	10.078546f	10.078546f	10.078546f
TS12	Class_A	Module_AD	2147483647,'H'	'H'	'H'
TS13	Class_A	Module_AD	-2147483648,'Z'	'Z'	'Z'
TS14	Class_A	Module_AD	145863479,'a'	a'	a'
TS15	Class_A	Module_AD	0,'i'	'i'	'i'
TS16	Class_A	Module_AD	-5748,'O'	'O'	'O'
TS17	Class_A	Module_AE	3.4028235E38f	3.4028235E38f	3.4028235E38f
TS18	Class_A	Module_AE	1.4E-45f	1.4E-45f	1.4E-45f
TS19	Class_A	Module_AE	0.58313185f	0.58313185f	0.58313185f
TS20	Class_A	Module_AE	1.0768449f	1.0768449f	1.0768449f
TS21	Class_A	Module_AE	10.078546f	10.078546f	10.078546f
TS22	Class_A	Module_AF	null	complete	complete
TS23	Class_A	Module_AG	"s"	"s"	"s"
TS24	Class_A	Module_AG	"A"	"A"	"A"
TS25	Class_A	Module_AG	" "	" "	" "
TS26	Class_A	Module_AG	"+-*/"	"+-*/"	"+-*/"
TS27	Class_A	Module_AG	"123"	"123"	"123"
TS28	Class_B	Module_BA	203	101	101
TS29	Class_B	Module_BA	999	499	499
TS30	Class_B	Module_BA	673477	336738	336738
TS31	Class_B	Module_BA	848452	424226	424226
TS32	Class_B	Module_BA	155152	77576	77576
TS33	Class_B	Module_BB	null	complete	complete
TS34	Class_B	Module_BC	5	"5"	"5"
TS35	Class_B	Module_BC	1547	"1547"	"1547"
TS36	Class_B	Module_BC	2081966562	"2081966562"	"2081966562"
TS37	Class_B	Module_BC	0	"0"	"0"
TS38	Class_B	Module_BC	-1966562	"-1966562"	"-1966562"
TS39	Class_B	Module_AE	5978	5979	5979
TS40	Class_B	Module_AE	9999	10000	10000
TS41	Class_B	Module_AE	-2066249954	-2066249953	-2066249953
TS42	Class_B	Module_AE	421224030	421224031	421224031
TS43	Class_B	Module_AE	-1324296098	-1324296097	-1324296097
TS44	Class_C	Module_CA	1.33205522," "	1.33205522," "	1.33205522," "
TS45	Class_C	Module_CA	1.281,"ZZZ"	1.281,"ZZZ"	1.281,"ZZZ"

TS46	Class_C	Module_CA	-1966562,"hello"	-1966562,"hello"	-1966562,"hello"
TS47	Class_C	Module_CA	1286296612,"AAA"	1286296612,"AAA"	1286296612,"AAA"
TS48	Class_C	Module_CA	9898,"..."	9898,"..."	9898,"..."
TS49	Class_C	Module_CB	5,5	5,5	5,5
TS50	Class_C	Module_CB	-147483,-147483	-147483,-147483	-147483,-147483
TS51	Class_C	Module_CB	-214748,-214748	-214748,-214748	-214748,-214748
TS52	Class_C	Module_CB	0,0	0,0	0,0
TS53	Class_C	Module_CB	145,789,857	145,789,857	145,789,857
TS54	Class_C	Module_CC	TRUE	TRUE	TRUE
TS55	Class_C	Module_CC	FALSE	FALSE	FALSE
TS56	Class_C	Module_CC	145,789,857	24435	24435
TS57	Class_C	Module_CC	0,0	0	0
TS58	Class_C	Module_CC	99,1	100	100
TS59	Class_C	Module_CC	-2,2	0	0
TS60	Class_C	Module_CC	123,456	579	579
TS61	Class_C	Module_CD	null	complete	complete
TS62	Class_D	Module_DA	5578,"iii"	5578,"iii"	5578,"iii"
TS63	Class_D	Module_DA	2147483647," "	2147483647," "	2147483647," "
TS64	Class_D	Module_DA	-2147483648," "	-2147483648," "	-2147483648," "
TS65	Class_D	Module_DA	0,"z"	0,"z"	0,"z"
TS66	Class_D	Module_DA	597823,"1"	597823,"1"	597823,"1"
TS67	Class_D	Module_DB	null	complete	complete
TS68	Class_D	Module_DC	2147483647," "	2147483647," "	2147483647," "
TS69	Class_D	Module_DC	-2147483648," "	-2147483648," "	-2147483648," "
TS70	Class_D	Module_DC	0,"z"	0,"z"	0,"z"
TS71	Class_D	Module_DC	597823,"1"	597823,"1"	597823,"1"
TS72	Class_D	Module_DC	5,"--"	5,"--"	5,"--"

หลังจากทดสอบการสร้างกรณีทดสอบจากซอร์สโค้ดและกรณีทดสอบดังกล่าว เครื่องมือนี้ใช้เวลาในการประมวลผล 14 มิลลิวินาที โดยไม่รวมเวลาในการสร้างไฟล์ PDF ใช้วิธีเฉลี่ยจากการประมวลผลทั้งหมด 20 รอบ ผลลัพธ์ที่ได้เครื่องมือนี้จะเป็นกรณีทดสอบระดับรวมหน่วย โดยมีกรณีทดสอบทั้งหมด 60 กรณีทดสอบสำหรับใช้ทดสอบระดับรวมหน่วยดังตารางที่ 4.4 ถึงตารางที่ 4.5

ตารางที่ 4.2 ตารางแสดงกรณีทดสอบระดับรวมหน่วยที่ได้จากการประมวลผลของเครื่องมือ

TestID	ClassName	ModuleCall	Module	TestCase	ExpectedResult	Result
TS1	Class	main	Module_DA	5578, "iii"	5578,"iii"	
TS2	Class	main	Module_DA	2147483647, " "	2147483647," "	
TS3	Class	main	Module_DA	-2147483648, " "	-2147483648," "	
TS4	Class	main	Module_DA	0, "z"	0,"z"	
TS5	Class	main	Module_DA	597823, "1"	597823,"1"	
TS6	Class	main	Module_BA	203	101	
TS7	Class	main	Module_BA	999	499	
TS8	Class	main	Module_BA	673477	336738	
TS9	Class	main	Module_BA	848452	424226	
TS10	Class	main	Module_BA	155152	77576	
TS11	Class	main	Module_CB	5, 5	5,5	
TS12	Class	main	Module_CB	-147483, -147483	-147483,-147483	
TS13	Class	main	Module_CB	-214748, -214748	-214748,-214748	
TS14	Class	main	Module_CB	0, 0	0,0	
TS15	Class_A	Module_AC	Module_AD	2147483647, 'H'	'H'	
TS16	Class_A	Module_AC	Module_AD	-2147483648, 'Z'	'Z'	
TS17	Class_A	Module_AC	Module_AD	145863479, 'a'	a'	
TS18	Class_A	Module_AC	Module_AD	0, 'i'	'i'	
TS19	Class_A	Module_AC	Module_AD	-5748, 'O'	'O'	
TS20	Class_A	Module_AC	Module_AE	3.4028235E38f	3.4028235E38f	
TS21	Class_A	Module_AC	Module_AE	1.4E-45f	1.4E-45f	
TS22	Class_A	Module_AC	Module_AE	0.58313185f	0.58313185f	
TS23	Class_A	Module_AC	Module_AE	1.0768449f	1.0768449f	
TS24	Class_A	Module_AC	Module_AE	10.078546f	10.078546f	
TS25	Class_A	Module_AD	Module_AF	null	complete	
TS26	Class_A	Module_AE	Module_AG	"s"	"s"	
TS27	Class_A	Module_AE	Module_AG	"A"	"A"	
TS28	Class_A	Module_AE	Module_AG	" "	" "	
TS29	Class_A	Module_AE	Module_AG	"+-*/"	"+-*/"	
TS30	Class_A	Module_AE	Module_AG	"123"	"123"	
TS31	Class_B	Module_BA	Module_AE	5978	5979	
TS32	Class_B	Module_BA	Module_AE	9999	10000	
TS33	Class_B	Module_BA	Module_AE	-2066249954	-2066249953	
TS34	Class_B	Module_BA	Module_AE	421224030	421224031	
TS35	Class_B	Module_BA	Module_AE	-1324296098	-1324296097	
TS36	Class_B	Module_BA	Module_AA	"Hello"	"HelloHello"	
TS37	Class_B	Module_BA	Module_AA	"123"	"123123"	
TS38	Class_B	Module_BA	Module_AA	"+-*/"	"+-*/+-*/"	
TS39	Class_B	Module_BA	Module_AA	"???"	"??????"	
TS40	Class_B	Module_BA	Module_AA	"WORLD"	"WORLDWORLD"	
TS41	Class_B	Module_AE	Module_BC	5	"5"	
TS42	Class_B	Module_AE	Module_BC	1547	"1547"	
TS43	Class_B	Module_AE	Module_BC	2081966562	"2081966562"	
TS44	Class_B	Module_AE	Module_BC	0	"0"	

TS45	Class_B	Module_AE	Module_BC	-1966562	"-1966562"	
TS46	Class_C	Module_CB	Module_CC	0, 0	0	
TS47	Class_C	Module_CB	Module_CC	99, 1	100	
TS48	Class_C	Module_CB	Module_CC	-2, 2	0	
TS49	Class_C	Module_CB	Module_CC	123, 456	579	
TS50	Class_C	Module_CB	Module_CD	null	complete	
TS51	Class_D	Module_DA	Module_DC	2147483647, " "	2147483647," "	
TS52	Class_D	Module_DA	Module_DC	-2147483648, " "	-2147483648," "	
TS53	Class_D	Module_DA	Module_DC	0, "z"	0,"z"	
TS54	Class_D	Module_DA	Module_DC	597823, "1"	597823,"1"	
TS55	Class_D	Module_DA	Module_DC	5, "--"	5,"--"	
TS56	Class_D	Module_DB	Module_DC	2147483647, " "	2147483647," "	
TS57	Class_D	Module_DB	Module_DC	-2147483648, " "	-2147483648," "	
TS58	Class_D	Module_DB	Module_DC	0, "z"	0,"z"	
TS59	Class_D	Module_DB	Module_DC	597823, "1"	597823,"1"	
TS60	Class_D	Module_DB	Module_DC	5, "--"	5,"--"	

4.3 อภิปรายผล

ในการทดสอบประสิทธิภาพของเครื่องมือที่ผู้วิจัยได้พัฒนาขึ้นมานี้ โดยแบ่งการทดสอบออกเป็นสองส่วนคือ การทดสอบการแจกแจงโครงสร้างความสัมพันธ์ระหว่างโมดูล และ การทดสอบการสร้างกรณีทดสอบระดับรวมหน่วย

การทดสอบการแจกแจงโครงสร้างความสัมพันธ์ระหว่างโมดูล ผลลัพธ์ที่ได้นั้นแสดงถึงประสิทธิภาพในการแจกแจงโครงสร้างซอร์สโค้ดได้หลายรูปแบบ และระบุความสัมพันธ์ระหว่างโมดูลได้ถูกต้อง แสดงให้เห็นว่ากระบวนการแจกแจงความสัมพันธ์ในเครื่องมือนี้สามารถแจกแจงความสัมพันธ์ได้ถูกต้องตามโครงสร้างของซอร์สโค้ด

การทดสอบการสร้างกรณีทดสอบระดับรวมหน่วย ได้แสดงถึงประสิทธิภาพทางด้านเวลาโดยการจำลองซอร์สโค้ดในรูปแบบต่างๆกัน จะสังเกตได้ว่าเวลาในการทดสอบโปรเจคที่มีความสัมพันธ์ระหว่างคลาสจำนวนมาก นั้นใช้เวลามากกว่าโปรเจคที่มีความสัมพันธ์ระหว่างโมดูลจำนวนมาก แต่เวลาที่ใช้นั้นน้อยมาก และการทดสอบกับซอร์สโค้ดที่มีความสัมพันธ์ซับซ้อนและจำลองกรณีทดสอบจำนวนมากเพื่อจำลองการนำไปใช้กับซอร์สโค้ดจริงนั้น ได้ผลลัพธ์ที่สามารถนำไปใช้งานได้จริง

บทที่ 5
สรุปผลการวิจัย

5.1 สรุปผลการวิจัย

จากการศึกษาวิจัยและออกแบบวิธีการสร้างกรณีทดสอบสำหรับการทดสอบระดับรวมหน่วยแบบเพิ่มทีละหน่วยโดยอัตโนมัติจากกรณีทดสอบระดับหน่วย สามารถสรุปได้ว่าวิธีการสร้างกรณีทดสอบสำหรับการทดสอบระดับรวมหน่วยแบบเพิ่มทีละหน่วยโดยอัตโนมัติจากกรณีทดสอบระดับหน่วยสามารถช่วยลดความซับซ้อนและเวลาในการสร้างกรณีทดสอบ เนื่องจากเครื่องมือที่พัฒนาขึ้นมานี้สามารถสร้างกรณีทดสอบได้แบบอัตโนมัติ และการสร้างกรณีทดสอบแบบดั้งเดิมนั้นจำเป็นต้องใช้ข้อมูลจำนวนมาก อีกทั้งยังต้องใช้มนุษย์ในการสร้างกรณีทดสอบ

ผลลัพธ์ที่ได้จากการวัดประสิทธิภาพโดนการทดสอง ทำให้เห็นว่าเครื่องมือนี้สามารถแจกแจงโครงสร้างความสัมพันธ์ระหว่างโมดูลและสร้างกรณีทดสอบได้ถูกต้อง เวลาที่ใช้นั้นก็น้อยมากเมื่อเทียบกับการสร้างกรณีทดสอบแบบดั้งเดิม

เนื่องจากงานวิจัยนี้มุ่งหวังให้วิธีวิธีการสร้างกรณีทดสอบสำหรับการทดสอบระดับรวมหน่วยแบบเพิ่มทีละหน่วยโดยอัตโนมัติเป็นแนวทางให้กับนักพัฒนาโปรแกรมในการพัฒนาโปรแกรมต่างๆ ในการสร้างเครื่องมือเพื่อช่วยในการสร้างกรณีทดสอบระดับรวมหน่วยสำหรับทดสอบซอฟต์แวร์ที่พัฒนาขึ้น

5.2 ประโยชน์ของการสร้างกรณีทดสอบสำหรับการทดสอบระดับรวมหน่วยแบบเพิ่มทีละหน่วยโดยอัตโนมัติจากกรณีทดสอบระดับหน่วย

1) ช่วยลดความซับซ้อนในการสร้างกรณีทดสอบระดับรวมหน่วย
2) ช่วยลดระยะเวลาในการสร้างกรณีทดสอบระดับรวมหน่วย
3) เป็นการนำกรณีทดสอบที่มีอยู่มาใช้ใหม่ ทำให้ช่วยลดทรัพยากรในการสร้างกรณีทดสอบระดับรวมหน่วย

5.3 ข้อจำกัดของการสร้างกรณีทดสอบสำหรับการทดสอบระดับรวมหน่วยแบบเพิ่มทีละหน่วยโดยอัตโนมัติจากกรณีทดสอบระดับหน่วย

เนื่องจากการสร้างกรณีทดสอบสำหรับการทดสอบระดับรวมหน่วยแบบเพิ่มทีละหน่วยโดยอัตโนมัติจากกรณีทดสอบระดับหน่วย ผู้วิจัยได้พัฒนาขึ้นมานี้เป็นเพียงเครื่องมือต้นแบบสามารถแจกแจงโครงสร้างและรอบรับการสร้างกรณีทดสอบเฉพาะตัวแปรพื้นฐานของภาษาจาวาและกรณีทดสอบระดับหน่วยที่ใช้ในการสร้างกรณีทดสอบระดับรวมหน่วยนั้นจะต้องอยู่ในรูปแบบที่กำหนดไว้

5.4 แนวทางในการพัฒนาต่อ

การนำเสนองานวิจัยนี้เพื่อเป็นแนวทางในการสร้างเครื่องมือสำหรับสร้างกรณีทดสอบระดับรวมหน่วยแบบอัตโนมัติ แนวทางในการพัฒนาต่อนั้นสามารถพัฒนาต่อได้ในหลากหลายแนวทาง ไม่ว่าจะเป็น การพัฒนาเพื่อนำไปประยุกต์ใช้กับภาษาในการเขียนโปรแกรมอื่นๆที่เป็นภาษาเชิงวัตถุ การพัฒนาเพื่อให้รองรับกรณีทดสอบที่มีความซับซ้อนยิ่งขึ้น รวมถึงการพัฒนากรณีทดสอบให้รองรับกับตัวแปรที่ไม่ใช้ตัวแปรพื้นฐานในภาษาจาวา

รายการอ้างอิง

สมคะเน บาลลา, พิชโยทัย มหัทธนาภิวัฒน์.(2011).การหาความแตกต่างของซอร์สโค้ดโดยใช้โครงสร้างไวยากรณ์ต้นไม้.**การประชุมวิชาการเสนอผลงานวิจัยระดับบัณฑิตศึกษาแห่งชาติ ครั้งที่ 23,** นครราชสีมา.:135-138

วิกิพีเดีย สารานุกรมเสรี (2556). การทดสอบซอฟต์แวร์ [ออนไลน์]. ที่มา : th.wikipedia.org/wiki/การทดสอบซอฟต์แวร์

วิกิพีเดีย สารานุกรมเสรี (2556). การเขียนโปรแกรมเชิงวัตถุ [ออนไลน์]. ที่มา : th.wikipedia.org/wiki/การเขียนโปรแกรมเชิงวัตถุ

วิกิพีเดีย สารานุกรมเสรี (2556). Unit testing [ออนไลน์]. ที่มา : en.wikipedia.org/wiki/Unit_testing

W.K. Chan, T.Y.Chen, T.H. Tse . (2002). An Overview of Integration Testing Techniques for Object-Oriented Programs. In **Proceedings of the 2nd ACIS Annual International Conference on Computer and Information Science (ICIS 2002),** Michigan.**:**1-6

Konstantin Rubinov.(2010). Generating Integration Test Cases Automatically. In **Proceeding FSE '10 Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering,** New York.:357-360

Alessandro Orso. (1998). Integration Testing of Object-Oriented Software [Online]**. www.cc.gatech.edu/~orso/papers/orso.thesis.pdf**

Hai Yuan, Tao Xie .(2006). Substra: a framework for automatic generation of integration tests. In **Proceedings of the 2006 international workshop on Automation of software test,** New York.:64-70

G. Fischer, J. Lusiardi, J.Wolff von Gudenberg.(2007). **Abstract Syntax Trees – and their Role in Model Driven Software Development**. In Proceedings of the International Conference on Software Engineering Advances(ICSEA 2007), Cap Esterel.: 38

Junfei Huang , Yunzhan Gong.(2010) .An EMF Activity Tree Based BPEL Defect Pattern Testing Method. **2nd International Conference on Computer Engineering and Technology**. Chengdu.: 468-471

Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Anna, Lorraine Bier.(1998). **Clone Detection Using Abstract Syntax Trees**. International Conference on Software Maintenance. Bethesda.: 368-377

Tahira Khatoon, Priyansha Singh, Shikha Shukla. (2012). **Abstract Syntax Tree Based Clone Detection for Java Projects**. IOSR Journal of Engineering. Vol. 2.: 45-47

Guo Tao , Dong Guowei, Qin Hu, Cui Baojiang.(2013). **Improved Plagiarism Detection Algorithm Based on Abstract Syntax Tree**. 2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies. Xi'an.: 714-719

Iulian Neamtiu, Jeffrey S. Foster, Michael Hicks. (2005**). Understanding Source Code Evolution Using Abstract Syntax Tree Matching**. MSR '05 Proceedings of the 2005 international workshop on Mining software repositories. New York.: 1-5

H.Reza (2012). Integration and system Testing[ออนไลน์]. ที่มา : http://people.aero.und.edu/~reza/Csci565-IntegrationTesting.ppt

e-Books Ramkhamhaeng University(2012).**วิศวกรรมซอฟต์แวร์ (Software engineering The Production quality software) [ออนไลน์]**. ที่มา : http://ebook.ram.edu/ebook/inside/html/dlbook.asp?code=CT484(48)

ภาคผนวก ก

บทความวิชาการที่ได้รับการตีพิมพ์เผยแพร่ ในระหว่างศึกษา

รายชื่อบทความที่ได้รับการตีพิมพ์เผยแพร่ในระหว่างการศึกษา

Zagon Budsabong, Pichayotai Mahathanapawat. 2557. **Tree Structure for Association Identification between Module**. The 10th National Conference on Computing and Information Technology (NCCIT 2014), Angsana Laguna Phuket, Thailand, 8-9 May 2014.

โครงสร้างต้นไม้สำหรับแจกแจงความสัมพันธ์ระหว่างโมดูล
Tree Structure for Association Identification between Module

สกรณ์ บุษบง และพิชโยทัย มหัทธนาภิวัฒน์
สาขาวิศวกรรมคอมพิวเตอร์ มหาวิทยาลัยเทคโนโลยีสุรนารี จนครราชสีมา.
zbzagon@gmail.com, pmh@sut.ac.th

บทคัดย่อ

ในปัจจุบันการเขียนโปรแกรมเชิงวัตถุเป็นที่นิยมอย่างมากในหมู่นักพัฒนาโปรแกรมทุกระดับ จากลักษณะเฉพาะของการเขียนโปรแกรมเชิงวัตถุนั้นทำให้การเปลี่ยนแปลง แก้ไขและการนำกลับมาใช้ใหม่ได้ง่าย ทำให้การเขียนโปรแกรมมีความสะดวกรวดเร็วมากขึ้น แต่ถ้าโปรแกรมที่ทำการเขียนนั้นมีความซับซ้อนมาก ทำให้มีคลาสเป็นจำนวนมาก ความสัมพันธ์ระหว่างคลาสและโมดูลก็มากตามไปด้วย เมื่อมีความจำเป็นที่จะต้องตรวจสอบโครงสร้างความสัมพันธ์ระหว่างโมดูลซึ่งอยู่ในคลาสหรือทดสอบการทำงาน ก็จะเกิดความยุ่งยากอันเนื่องมาจากความซับซ้อนของตัวโปรแกรมเอง

จากปัญหานี้ผู้วิจัยจึงได้ออกแบบโครงสร้างต้นไม้เพื่อใช้ในการระบุความสัมพันธ์ระหว่างโมดูล ในแต่ละคลาส โดยการปรับปรุง Abstract Syntax Tree *ให้เหมาะสมในการแจกแจงโครงสร้างความสัมพันธ์เพื่อใช้แก้ปัญหาข้างต้นและเป็นต้นแบบในการพัฒนาต่อไป*

คำสำคัญ: *การเขียนโปรแกรมเชิงวัตถุ, การทดสอบ, โครงสร้างไวยากรณ์ต้นไม้, ความสัมพันธ์ของโมดูล*

Abstract

Nowadays, object-oriented programming is very popular among developers of all levels.

Characteristic of object-oriented programming are to make the transition editing and reusability easier, make programming a lot more convenient. But if that program is very complex, containing a lot of classes and a lot of relationships between classes and modules. It is necessary to examine the structure of relationships between these. In order to verify accuracy, or functional testing, it would be difficult because of the complexity of the program.

By this problem researchers have designed a tree structure to identify relationships between modules in each class by improving the Abstract Syntax Tree for proper identification of the structural relationship to the above solution and a prototype for further development.

Keyword: Object-oriented programming, Testing, Abstract Syntax Trees, Module Relation

1. บทนำ

การเขียนโปรแกรมเชิงวัตถุนั้นเป็นที่นิยมอย่างมากในหมู่นักพัฒนาทุกระดับ ตั้งแต่ผู้เริ่มต้นไปจนถึงผู้เชี่ยวชาญ ในการพัฒนาโปรแกรมนั้นมักจะมีความซับซ้อนมาก เนื่องจากความก้าวหน้าทางเทคโนโลยีทำให้การออกแบบโครงสร้างของโปรแกรมมีความซับซ้อนตามไปด้วย ดังนั้นการตรวจสอบว่าโปรแกรมที่พัฒนาขึ้นนั้นมีโครงสร้างความสัมพันธ์ระหว่างโมดูลถูกต้องตามการออกแบบจึงทำได้ยากมากขึ้น

ในการเขียนโปรแกรมเชิงวัตถุนั้นมักจะมีคลาสจำนวนมาก ภายในคลาสนั้นก็จะประกอบไปด้วยโมดูลต่างๆมากมายซึ่งแต่ละโมดูลจะมีความสัมพันธ์ระหว่างโมดูลในคลาสเดียวกัน หรือโมดูลในคลาสอื่นๆทำให้เกิดความสัมพันธ์ทับซ้อนกันหลายชั้น

จุดนี้มักจะเป็นปัญหาในการตรวจสอบความถูกต้องของโปรแกรมเพื่อให้ตรงกับการออกแบบ

ในเอกสารงานวิจัยนี้ จะนำเสนอเทคนิคและวิธีการที่ใช้ในการตรวจสอบความสัมพันธ์ระหว่างโมดูลโดยใช้ การปรับปรุง Abstract Syntax Tree ในการแจกแจงโครงสร้างของซอร์สโค้ดเพื่อระบุความสัมพันธ์ระหว่างโมดูลออกมาในรูปแบบของโครงสร้างต้นไม้

2. วรรณกรรมที่เกี่ยวข้อง

Abstract Syntax Tree(AST) เป็นโครงสร้างพื้นฐานที่ทำหน้าที่แสดงโครงสร้างของข้อมูลและซอร์สโค้ด ถูกนำไปประยุกต์ใช้กันอย่างแพร่หลาย เช่น Compiler, Interpreter รวมถึงการอธิบายเงื่อนไขต่างๆ เป็นต้น

Somka-ne Balla และ P. Mahatthanapiwat[1] ได้นำเสนองานวิจัยเกี่ยวกับความเป็นไปได้ของการนำ AST ใช้ในการแจกแจงโครงสร้างของซอร์สโค้ด เพื่อใช้ในการตรวจจับความแตกต่างของซอร์สโค้ด ซึ่งให้ผู้วิจัยได้แนวคิดในการใช้ AST แจกแจงโครงสร้างในระดับโมดูล งานวิจัยของ Junfei Huang และ Yunzhan Gong[2] ได้นำเสนอการทดสอบ BPEL(Web Services Business Process Execution Language) โดยใช้ต้นไม้ EMF ซึ่งมีความคล้ายคลึงกับ AST เพื่อใช้ในการอธิบายโครงสร้างของ BPEL ซึ่งมีแนวคิดเดียวกันกับงานวิจัยนี้คือมีการใช้ AST แจกแจงโครงสร้างเพื่อใช้ในการทดสอบ

Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant'Annaและ Lorraine Bier[3]ได้นำเสนองานวิจัยที่เกี่ยวกับวิธีการตรวจจับการทำซ้ำของซอร์สโค้ดโดยใช้ AST มาประยุกต์ใช้ในการแจกแจงโครงสร้างของซอร์สโค้ด

ยังมีงานวิจัยของ Tahira Khatoon Priyansha Singh และ Shikha Shukla[4]ที่ได้นำเสนองานวิจัยเกี่ยวกับการตรวจจับการซ้ำกันของซอร์สโค้ดเช่นเดียวกันแต่มีการนำไปใช้กับภาษาจาวา ทำให้ผู้วิจัยเกิดแนวคิดในการนำ AST มาประยุกต์ร่วมกับภาษาเชิงวัตถุ

งานวิจัยของ Guo Tao, Dong Guowei, Qin Hu, Cui Baojiang[5] เป็นการพัฒนาอัลกอริทึมเพื่อตรวจจับการถูกปลอมแปลงซอร์สโค้ดโดยเฉพาะ ภายในคำสั่ง if และ while โดยใช้พื้นฐานของ AST ซึ่งมีการคำนวณ Hash Value ของทุก Node และนำค่า Hash Value ของแต่ละ Node มาเปรียบเทียบกัน

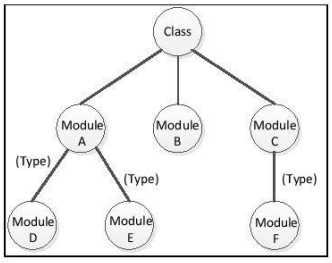
งานวิจัยของ Iulian Neamtiu, Jeffrey S. Foster และ Michael Hicks[6] ได้นำเสนองานวิจัยเกี่ยวกับการตรวจสอบการพัฒนาซอฟต์แวร์ภาษาซี โดยใช้เทคนิคการจับคู่ของ AST เน้นไปที่ชนิดและตัวแปรเพื่อตรวจสอบความแตกต่างของเวอร์ชันของซอฟต์แวร์ งานวิจัยของ G. Fischer, J. Lusiardi และ J.Wolff von Gudenberg[7] ได้นำเสนอ เกี่ยวกับบทบาทของ AST ต่อการพัฒนาซอฟต์แวร์

จากงานวิจัยเหล่านี้ทำให้เกิดแนวคิดที่จะนำ AST มาประยุกต์ใช้ในการแจกแจงโครงสร้างของซอร์สโค้ด เพื่อสร้างต้นไม้สำหรับแจกแจงความสัมพันธ์ระหว่างโมดูล เพื่อใช้ในการตรวจสอบความสัมพันธ์ระหว่างโมดูลต่อไป

3. วิธีการดำเนินการวิจัย

งานวิจัยนี้ได้มีการปรับปรุง AST ขึ้นมาเพื่อมุ่งเน้นไปที่การแจกแจงความสัมพันธ์ระหว่างโมดูล

AST นั้นมีโครงสร้างเหมือนต้นไม้ทั่วๆไปเดิมที AST นั้นได้ถูกนำไปประยุกต์ได้หลากหลาย หนึ่งในนั้นคือทำหน้าที่แสดงโครงสร้างของซอร์สโค้ด และแจกแจงในรูปแบบของต้นไม้



ภาพที่ 1:โครงสร้าง AST ที่ได้ปรับปรุง

โครงสร้างต้นไม้ ที่ได้ปรับปรุงขึ้นมานั้น แต่ละต้นไม้จะเป็นตัวแทนของแต่ละคลาส ดังภาพที่ 1 ต้นไม้เหล่านี้จะอยู่รวมกันเป็นกลุ่ม โดยมีการเชื่อมโยงกันระหว่างต้นไม้

โครงสร้างต้นไม้ที่ผู้วิจัยได้ปรับปรุงขึ้นมานั้นประกอบไปด้วย โครงสร้างหลัก 3 ส่วน ดังต่อไปนี้

Root เป็นตัวแทนของชื่อคลาส กำหนดค่าภายในคือ ชื่อของคลาส

Node เป็นตัวแทนของแต่ละโมดูลโดยจะเก็บชื่อของโมดูลที่อยู่ภายในคลาสและพารามิเตอร์ของโมดูล

Edge เป็นตัวแสดงความสัมพันธ์ระหว่าง Node จะเก็บค่าอาร์กิวเมนต์ของโมดูลที่ส่งให้กันรวมถึงโมดูลที่ไม่มีการส่งอาร์กิวเมนต์ให้กันด้วย

โครงสร้างทั้ง 3 ส่วนนี้จะมี Attribute อยู่ภายในซึ่งใช้ในการระบุตัวตนและแสดงความสัมพันธ์ต่างๆ

Node แต่ละ Node จะเรียงตามการเรียกใช้ Node อื่นๆ โดยมี Edge แสดงความสัมพันธ์ระหว่าง Node โดยที่ Edge จะมีค่าอาร์กิวเมนต์ของโมดูลผู้เรียกซึ่งจะสัมพันธ์กับพารามิเตอร์ ของโมดูลผู้ถูกเรียก Edge ที่เชื่อมระหว่าง Root และ Child นั้นจะไม่มี Attribute ภายใน เนื่องจากไม่ได้เกิดความสัมพันธ์ระหว่างโมดูล เพียงแต่เป็นการแสดงโครงสร้างต้นไม้ให้สมบูรณ์

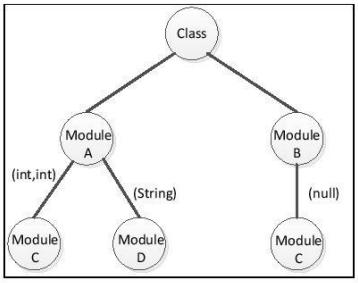
ระดับของ Node ภายในต้นไม้จะแสดงถึงลำดับการเรียกใช้โมดูล โดยที่ Node ที่มีการเรียกโมดูลอื่นๆ ก็จะมี Node มาเชื่อมต่อจาก Node นี้ เป็นลำดับชั้นลงไปเรื่อยๆซึ่งสุดท้ายแล้ว Node ที่ไม่ได้ถูกเรียกใช้จะอยู่ระดับบนสุด และ Node ที่ไม่ได้เรียกโมดูลใดๆจะอยู่ระดับล่างสุด

เนื่องจากกระบวนการนี้อยู่ภายหลังกระบวนการพัฒนาซอฟต์แวร์ ซอร์สโค้ดที่นำเข้ามาในกระบวนการนี้จึงผ่านการ Compile จาก Compiler เรียบร้อยแล้ว ทำให้ไม่จำเป็นต้องคำนึงถึงการ Encapsulation เนื่องจาก Compiler ทำหน้าที่ตรวจสอบความถูกต้องทั่วไปให้แล้ว

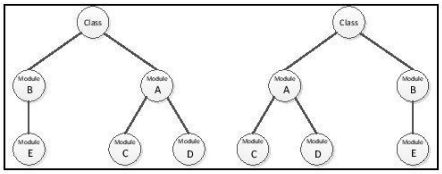
ในกรณีที่โมดูลมีการ Overloading คือการมีโมดูลที่มีชื่อเหมือนกันแต่มีพารามิเตอร์ต่างกัน จะมีการแสดงโมดูลแยกออกจากกันตามโครงสร้างของการเรียกใช้งานตามภาพที่ 2 ซึ่งโมดูล C จะถูกเรียกใช้ทั้ง2โมดูลแต่อาร์กิวเมนต์ที่ได้รับจะแตกต่างกันไปตามพารามิเตอร์ของโมดูลนั้นๆ

จำนวนของต้นไม้นั้นจะขึ้นอยู่กับจำนวนของคลาส โดยจะไม่สนใจคลาสที่เป็น Interface เนื่องจากโมดูลภายใน Interface นั้นไม่มีการ Implement ทำให้ไม่มีผลต่อโครงสร้างต้นไม้สำหรับแจกแจงความสัมพันธ์นี้

รูปร่างของต้นไม้แต่ละต้นนั้นจะแปรผันตามลำดับ และความซับซ้อนของโมดูลที่มีความสัมพันธ์กันทำให้การจัดวางตำแหน่งของโมดูล การจัดลำดับของ Node ของโมดูลระดับแรกนั้น สามารถสลับตำแหน่งได้ขึ้นอยู่กับกระบวนการระบุโมดูล ดังนั้น ซอร์สโค้ดที่นำเข้าสู่กระบวนการสร้างสามารถสร้างต้นไม้ได้หลายแบบ แต่สามารถใช้อธิบายโครงสร้างซอร์สโค้ดได้เช่นเดียวกันดังภาพที่ 3



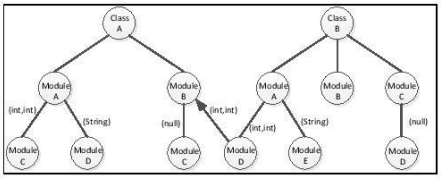
ภาพที่ 2:ภาพแสดงโครงสร้าง AST ที่ปรับปรุงแล้วกับโมดูล Overloading



ภาพที่ 3:ภาพแสดงความแตกต่างของโครงสร้างต้นไม้จากซอร์สโค้ดเดียวกัน

การเรียกใช้โมดูลจากคลาสไปยังโมดูลจากคลาสอื่นๆนั้นจะทำให้เกิดโครงสร้างความสัมพันธ์ระหว่างต้นไม้โดยมี Edge เชื่อมจาก Node ของต้นไม้ต้นหนึ่งไปยังอีกต้นหนึ่ง โดยที่ความซับซ้อนของต้นไม้จะขึ้นอยู่กับความสัมพันธ์ระหว่างโมดูล ถ้าแต่ละคลาสเกิดความสัมพันธ์ระหว่างโมดูลจำนวนมากต้นไม้ก็จะมี Edge เชื่อมต่อกันระหว่างต้นไม้เป็นจำนวนมากตามไปด้วย

Edge ที่เชื่อมระหว่างโมดูลที่อยู่ภายในคลาสที่ต่างกันนั้นจำเป็นที่จะต้องมีการกำหนดทิศทางอย่างชัดเจน เนื่องจากการแสดงโมเดลโครงสร้างต้นไม้นี้ แต่ละต้นไม้นั้นเป็นอิสระต่อกัน ถ้าไม่มีการกำหนดทิศทางของ Edge อย่างชัดเจนจะทำให้เกิดความสับสนระหว่างโมดูลผู้เรียกและผู้ถูกเรียกโดยมี Edge เชื่อมโยงดังภาพที่ 4 ซึ่งเป็นภาพที่แสดงความสัมพันธ์ระหว่างต้นไม้ โดยที่ความสัมพันธ์ข้ามคลาสนั้นเกิดขึ้นที่ โมดูล D ภายในคลาส B และโมดูลB ภายในคลาส A



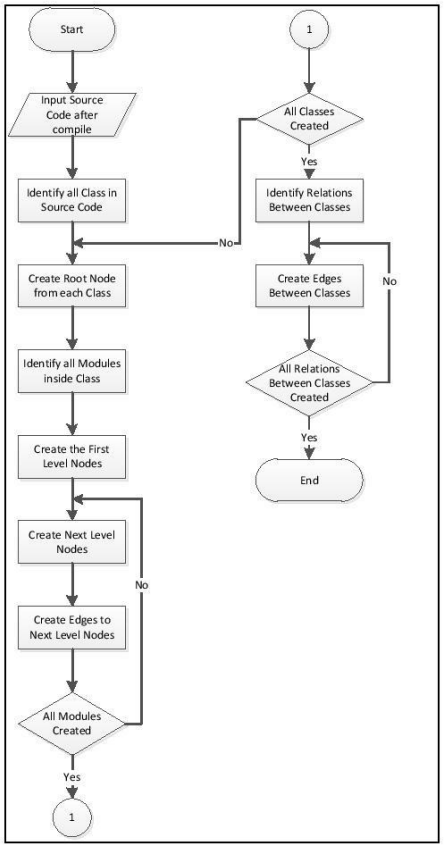
ภาพที่ 4:ภาพแสดงความสัมพันธ์ระหว่างต้นไม้

ในกระบวนการสร้างโครงสร้างต้นไม้สำหรับแจกแจงความสัมพันธ์ระหว่างโมดูลนั้นมีขั้นตอนดัง ภาพที่ 5 ซึ่งมีรายละเอียดดังนี้

1. Input Source Code after Compile : เป็นการนำเข้าซอร์สโค้ดที่ผ่านการ Compile แล้ว
2. Identify All Class in Source Code : เป็นการระบุคลาสทั้งหมดที่มีในซอร์สโค้ด รวมถึงการตรวจสอบว่าเป็น Interface หรือไม่
3. Create Root Node from each Class : สร้าง Root โดยกำหนด Attribute เป็น ชื่อของคลาส
4. Identify all Modules inside Class :นำข้อมูลของคลาสมาจำแนกโมดูลภายในโดยเริ่มจากระบุโมดูลที่มีทั้งหมดภายในคลาส และแยกระหว่างโมดูลที่เป็นผู้เรียก ผู้ถูกเรียก และเป็นทั้งสองอย่าง รวมถึงการแจกแจงพารามิเตอร์จองแต่ละโมดูลด้วย
5. Create the First Level Nodes : กำหนด Node ระดับแรกจากโมดูลที่ไม่มีผู้เรียกก่อน แล้วใช้โมดูลระดับแรกที่กำหนดไว้แล้วมาระบุโมดูลที่ถูกเรียกจากโมดูลระดับแรก
6. Create Next Level Nodes : สร้าง Node ระดับถัดมาโดยอ้างอิงจาก Node ระดับก่อนหน้า จากนั้นตรวจสอบว่า Node นั้นมีการเรียกโมดูลหรือไม่ ถ้ามีการเรียกโมดูลอื่นจะนำโมดูลที่ถูกเรียกมาสร้างเป็นลำดับต่อไป ถ้าไม่มีการเรียกใช้โมดูลอื่น ก็จะข้ามโมดูลนี้ไป
7. Create Edges to Next Level Nodes : กำหนด Edge จาก พารามิเตอร์ ของโมดูลที่ถูกเรียก ถ้าโมดูลระดับแรกนี้ไม่มีการเรียกโมดูลอื่นก็ข้ามโมดูลนี้ไป
8. All Modules Created : ทำขั้นตอนที่ 6 และ7 ซ้ำจนครบทุกโมดูล
9. All Classes Created : ทำขั้นตอนที่ 3 ถึง 8 จนครบทุกคลาส
10. Identify Relations Between Classes : เป็นการระบุความสัมพันธ์ของแต่ละคลาส เพื่อกำหนดโมดูดที่จะเป็นจุดเชื่อมต่อระหว่างคลาส
11. Create Edge Between Classes : สร้าง Edge เชื่อมโมดูลที่มีการเรียกข้ามคลาส พร้อมทั้งมีการกำหนดพารามิเตอร์ ของโมดูลที่ถูกเรียก
12. All Relation Between Classes Created : ทำขั้นตอนที่ 11 ซ้ำจนครบทุกความสัมพันธ์ระหว่างโมดูลข้ามคลาสที่เกิดขึ้น

หลังจากทำตามขั้นตอนดังกล่าวเสร็จสิ้นแล้วจะได้กลุ่มของโครงสร้างต้นไม้ที่มีความสัมพันธ์กัน ซึ่งกลุ่มของต้นไม้ที่ได้มานั้นเป็นโครงสร้างต้นไม้ที่สามารถแสดงความสัมพันธ์ระหว่าง

โมดูลภายในคลาสเดียวกันและความสัมพันธ์ระหว่างโมดูลภายในคลาสที่มีความสัมพันธ์กับคลาสอื่นๆ



ภาพที่ 5:ภาพแสดงผังงาน(Flow chart) ของกระบวนการสร้างโครงสร้างต้นไม้

4. ผลการดำเนินงาน

จากขั้นตอนที่กล่าวมาผู้วิจัยได้ดำเนินการทดลองกระบวนการสร้างโครงสร้างต้นไม้นี้โดยการใช้ซอร์สโค้ดภาษาจาวาดังภาพที่ 6 ซอร์สโค้ดดังกล่าวประกอบไปด้วยคลาสทั้งหมด 4 คลาส มีโมดูลรวมกันทั้งหมด 18 โมดูล และมีความสัมพันธ์ระหว่างโมดูลข้ามคลาสทั้งหมด 3 แห่ง นำซอร์สโค้ดเหล่านี้มาเข้าสู่กระบวนการสร้างโครงสร้างต้นไม้

```
1   package sample;
2   public class Class_A {
3       public void ModuleAA() {
4           ModuleAD(5, 7);
5           ModuleAE(1f);
6           new Class_D().ModuleDB();
7       }
8       public void ModuleAB() {
9           new Class_B().ModuleBA("a");
10      }
11      public void ModuleAC() {
12          ModuleAF(Boolean.TRUE);
13      }
14      public int ModuleAD(int a, int b) {
15          return a + b;
16      }
17      public void ModuleAE(float a) {
18      }
19      public void ModuleAF(Boolean a) {
20      }
21  }
```

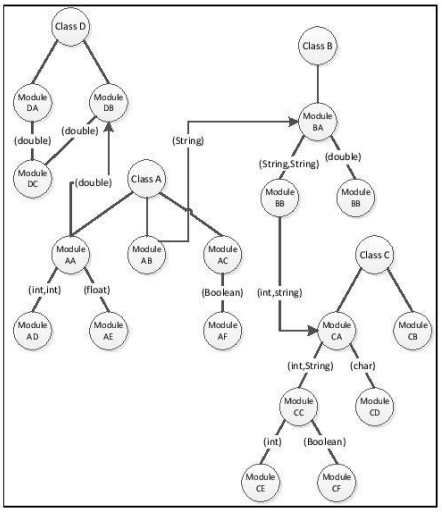
```
1   package sample;
2   public class Class_B {
3       public void ModuleBA(String a) {
4           ModuleBB(a, a);
5           ModuleBB(5.142);
6       }
7       public void ModuleBB(String a, String b) {
8           new Class_C().ModuleCA(5, b);
9       }
10      public void ModuleBB(double a) {
11      }
12  }
```

```
1   package sample;
2   public class Class_C {
3       public void ModuleCA(int a, String b) {
4           ModuleCC(a, b);
5           ModuleCD('a');
6       }
7       public void ModuleCB() {
8       }
9       public void ModuleCC(int a, String b) {
10          ModuleCE(a);
11          ModuleCF(true);
12      }
13      public void ModuleCD(char a) {
14      }
15      public void ModuleCE(int a) {
16      }
17      public void ModuleCF(Boolean a) {
18      }
19  }
```

```
1   package sample;
2   public class Class_D {
3       public void ModuleDA() {
4           ModuleDC(5.55);
5       }
6       public void ModuleDB() {
7           ModuleDC(5.55);
8       }
9       public void ModuleDC(double a) {
10      }
11  }
```

ภาพที่ 6:ภาพแสดงซอร์สโค้ดที่ใช้ในการทดสอบ

หลักจากที่กระบวนการสร้างต้นไม้สำหรับแจกแจงความสัมพันธ์ระหว่างโมดูลเสร็จสิ้นแล้ว จะได้กลุ่มของต้นไม้ดังภาพที่ 7 ซึ่งจะเห็นได้ว่าโครงสร้างต้นไม้นี้สามารถแจกแจงโครงสร้างของซอร์สโค้ด และสามารถนำไปใช้ตรวจสอบความสัมพันธ์ระหว่างโมดูลได้สะดวกเนื่องจากโครงสร้างมีความเรียบง่ายและง่ายต่อการทำความเข้าใจ



ภาพที่ 7:ภาพแสดงกลุ่มของโครงสร้างต้นไม้ที่จำลองขึ้นมาจากซอร์สโค้ดในภาพที่ 6

5.สรุป

โครงสร้างต้นไม้สำหรับแจกแจงความสัมพันธ์ระหว่างโมดูลเป็นโครงสร้างต้นไม้ที่จะช่วยให้การตรวจสอบความสัมพันธ์ระหว่างโมดูลนั้นง่ายขึ้น สามารถนำไปใช้ร่วมกับกระบวนการทดสอบโปรแกรมได้และเป็นแนวคิดที่สามารถนำไปพัฒนาเครื่องมือสร้างโครงสร้างต้นไม้สำหรับแจกแจงความสัมพันธ์ระหว่างโมดูลแบบอัตโนมัติได้ อย่างไรก็ตามแนวคิดนี้ยังคงต้องพึ่งมนุษย์ในการจำลองโครงสร้าง อีกทั้งยังไม่ได้ถูกนำไปใช้จริงและในขั้นการทดสอบนั้นได้ใช้ซอร์สโค้ดที่มีความซับซ้อนไม่มากในการทดสอบ

ปัจจุบันผู้เขียนกำลังพัฒนาโครงสร้างต้นไม้สำหรับแจกแจงความสัมพันธ์ระหว่างโมดูลในภาษาจาวาให้เป็นเครื่องมือเพื่อนำไปใช้ร่วมกับการสร้าง Integration Test Case ต่อไป

เอกสารอ้างอิง

[1] Somka-ne Balla, P. Mahatthanapiwat, “Difference Detection Using Abstract Syntax Tree”, The *23*rd National Graduate Research Conference (GRAD*23)*, Rajamangala University of Technology Isan, Nakorn Ratchasima, Thailand, Oct 23-24, 2011.

[2] Junfei Huang , Yunzhan Gong, “An EMF Activity Tree Based BPEL Defect Pattern Testing Method”,2nd International Conference on Computer Engineering and Technology, vol. 7, pp. 468-471, 2010.

[3] Ira D. Baxter, Andrew Yahin, Leonardo Moura, Marcelo Sant’Anna, Lorraine Bier, “Clone Detection Using Abstract Syntax Trees”, International Conference on Software Maintenance, pp. 368-377, 1998.

[4] Tahira Khatoon, Priyansha Singh, Shikha Shukla, “Abstract Syntax Tree Based Clone Detection for Java Projects”, IOSR Journal of Engineering, Vol 2, pp. 45-47, 2012.

[5] Guo Tao , Dong Guowei, Qin Hu, Cui Baojiang, “Improved Plagiarism Detection Algorithm Based on Abstract Syntax Tree”, 2013 Fourth International Conference on Emerging Intelligent Data and Web Technologies, pp. 714-719, 2013.

[6] Iulian Neamtiu, Jeffrey S. Foster, Michael Hicks, “Understanding Source Code Evolution Using Abstract Syntax Tree Matching”, MSR '05 Proceedings of the 2005 international workshop on Mining software repositories, pp. 1-5, 2005.

[7] G. Fischer, J. Lusiardi, J.Wolff von Gudenberg, “Abstract Syntax Trees – and their Role in Model Driven Software Development”, International Conference on Software Engineering Advances(ICSEA 2007), pp. 38, 2007.

ภาคผนวก ข

รายละเอียดของคลาสต่างๆในการพัฒนาเครื่องมือ

รายละเอียดของคลาสต่างๆในการพัฒนาเครื่องมือ

ในการพัฒนาเครื่องมือในวิทยานิพนธ์ฉบับนี้ผู้วิจัยได้พัฒนาคลาสต่างๆเพื่อนำมาประยุกต์ใช้กับวิธีการต่างๆที่ได้กล่าวไว้ในวิทยานิพนธ์ฉบับนี้ โดยผู้วิจัยได้แบ่งลักษณะของคลาสออกเป็น 3 กลุ่มดังนี้

1) กลุ่ม Visitor เป็นกลุ่มที่พัฒนาโดยใช้ไลบรารีที่มีชื่อว่า javaparser1.0.8.jar ทำหน้าที่ในการแจงแจงซอร์สโค้ดเพื่อให้ได้ข้อมูลส่วนประกอบต่างๆของซอร์สโค้ด คลาสกลุ่ม Visitor มี 4 คลาส ดังนี้
    - AST_ClassVisitor.java
    - AST_FieldDeclarationVisitor.java
    - AST_MethodCallExprVisitor.java
    - AST_MethodVisitor.java
2) กลุ่ม Generate เป็นกลุ่มที่มุ่งเน้นไปที่กระบวนการทำงานในส่วนต่างๆของเครื่องมือ คลาสกลุ่ม Generate มีดังนี้
    - AST_GenerateClass.java
    - AST_GenerateMethod.java
    - AST_GenerateTestCase.java
    - AST_Generate_MultiClass.java
    - IT_GeneratePDF.java
    - AST_AllRun.java
3) กลุ่ม Attribute เป็นกลุ่มของคลาสที่ทำหน้าที่เก็บข้อมูลต่างๆในรูปของ Object ประกอบไปด้วยคลาสดังนี้
    - AST_Class.java
    - AST_ClassMethod.java
    - AST.java
    - AST_Child.java
    - AST_Method.java
    - AST_MethodCall.java
    - AST_MethodCallClass.java
    - AST_MethodCallN.java

- AST_TestCase.java
- AST_Var.java
- TypePromo.java

ประวัติผู้เขียน

นายสกรณ์ บุษบง เกิดเมื่อวันที่ 15 พฤจิกายน พ.ศ. 2531 ที่โรงพยาบาลสรรพสิทธิประสงค์ อำเภอเมืองอุบลราชธานี จังหวัดอุบลราชธานี เริ่มเข้าศึกษาระดับอนุบาล 1ถึงระดับอนุบาล 2 ที่โรงเรียนมารีย์อนุสรณ์ จังหวัดบุรีรัมย์ จากนั้นย้ายไปเรียนระดับชั้นประถมศึกษาตอนต้น และชั้นประถมศึกษาตอนปลายที่โรงเรียนอนุบาลบุรีรัมย์ หลังจากสำเร็จการศึกษาในระดับประถมศึกษา ได้เข้าศึกษาต่อในระดับมัธยมศึกษาตอนต้นที่โรงเรียนบุรีรัมย์พิทยาคม จนจบการศึกษาในระดับมัธยมศึกษาตอนปลายในปีการศึกษา 2550 หลังจากนั้นได้เข้าศึกษาต่อระดับปริญญาตรีที่มหาวิทยาลัยเทคโนโลยีราชมงคลตะวันออก วิทยาเขตจักรพงษภูวนารถ คณะบริหารธุรกิจและเทคโนโลยีสารสนเทศ ในสาขาวิชาวิทยาการคอมพิวเตอร์ และสำเร็จการศึกษาเมื่อปี พ.ศ. 2553 หลังจากจบการศึกษาระดับปริญญาตรี ได้เข้ารับการศึกษาในระดับปริญญาโท สาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี ในปี 2554

ระหว่างที่ได้รับการศึกษาในระดับปริญญาโท ได้รับความอนุเคราะห์จากอาจารย์ ผู้ช่วยศาสตราจารย์ พิชโยทัย มหัทธนาภิวัฒน์ และได้รับความไว้วางใจให้เป็นผู้ช่วยสอนปฏิบัติการในรายวิชา Object-Oriented Technology

-