วิธีการเรียงลำดับคลาสสำหรับการทดสอบแบบบูรณาการโดยใช้เทคนิคการตัดส่วนเชิงวัตถุ

นางสาวจุฑารัตน์ เจริญไพบูลย์กิจ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2551

AN APPROACH OF ORDERING CLASS FOR INTEGRATION TESTING USING OBJECT-ORIENTED SLICING TECHNIQUE

Miss Jutarat Jaroenpiboonkit

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering Program in Computer Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2008

หัวข้อวิทยานิพนธ์	วิธีการเรียงลำดับคลาสสำหรับการทดสอบแบบบูรณาการโดยใช้เทคนิคการตัดส่วนเชิงวัตถุ
โดย	นางสาวจุฑารัตน์ เจริญไพบูลย์กิจ
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร.ธาราทิพย์ สุวรรณศาสตร์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(รองศาสตราจารย์ ดร. บุญสม เลิศหิรัญวงศ์)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(รองศาสตราจารย์ ดร. วิวัฒน์ วัฒนาวุฒิ)

..... อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(รองศาสตราจารย์ ดร. ธาราทิพย์ สุวรรณศาสตร์)

..... กรรมการ
(รองศาสตราจารย์ ดร. พรศิริ หมื่นไชยศรี)

..... กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร. อาทิตย์ ทองทักษ์)

..... กรรมการภายนอกมหาวิทยาลัย
(ผู้ช่วยศาสตราจารย์ ดร. ภัทรชัย ลลิดโรจน์วงศ์)

จุฑารัตน์ เจริญไพบูลย์กิจ : วิธีการเรียงลำดับคลาสสำหรับการทดสอบแบบบูรณาการโดยใช้เทคนิคการตัดส่วนเชิงวัตถุ. (AN APPROACH OF ORDERING CLASS FOR INTEGRATION TESTING USING OBJECT-ORIENTED SLICING TECHNIQUE) อ. ที่ปรึกษาวิทยานิพนธ์หลัก: รศ.ดร. ธาราทิพย์ สุวรรณศาสตร์, 90 หน้า.

การทดสอบแบบบูรณาการเป็นการหาข้อผิดพลาดในการทำงานร่วมกันระหว่างส่วนประกอบของซอฟต์แวร์ ในการทดสอบแบบบูรณาการสำหรับซอฟต์แวร์เชิงวัตถุ จำเป็นต้องกำหนดลำดับคลาสที่นำมาทดสอบ แต่ปัญหาของการเรียงลำดับคลาสคือ การเรียงลำดับคลาสโดยที่ความสัมพันธ์ระหว่างคลาสมีการเรียกที่ขึ้นต่อกันแบบมีวง สำหรับวิธีการเรียงลำดับคลาสวิธีอื่นๆ จะทำการลบความสัมพันธ์ระหว่างคลาสเพื่อขจัดการเรียกที่ขึ้นต่อกันแบบมีวง โดยจะต้องสร้างสตับขึ้นเพื่อใช้ทดสอบคลาสที่ถูกลบความสัมพันธ์ งานวิจัยนี้นำเสนอวิธีการเรียงลำดับคลาสสำหรับการทดสอบแบบบูรณาการ วิธีการนี้ใช้การตัดส่วนเชิงวัตถุในการขจัดการเรียกที่ขึ้นต่อกันแบบมีวง เพื่อลดจำนวนสตับในการเรียงลำดับคลาสสำหรับทำการทดสอบ

วิธีการเรียงลำดับคลาสสำหรับการทดสอบแบบบูรณาการที่เสนอในวิทยานิพนธ์นี้ประกอบด้วยขั้นตอนหลัก 3 ขั้นตอนได้แก่ ขั้นตอนการวิเคราะห์ความสัมพันธ์ระหว่างคลาส ขั้นตอนการขจัดการเรียกที่ขึ้นต่อกันแบบมีวง และขั้นตอนการเรียงลำดับคลาส โดยในขั้นตอนการวิเคราะห์ความสัมพันธ์ระหว่างคลาส วิทยานิพนธ์นี้ใช้ทาร์จันอัลกอริทึมในการหากลุ่มคลาสที่ความสัมพันธ์ระหว่างคลาสมีการเรียกที่ขึ้นต่อกันแบบมีวง จากนั้นจึงทำการแบ่งประเภทกลุ่มคลาสที่มีการเรียกที่ขึ้นต่อกันแบบมีวง สำหรับการขจัดการเรียกที่ขึ้นต่อกันแบบมีวงซึ่งเป็นขั้นตอนถัดไปนั้น ใช้การตัดส่วนเชิงวัตถุในการขจัดการเรียกที่ขึ้นต่อกันตามประเภทของกลุ่มคลาสที่ได้แบ่งประเภทไว้แล้ว เมื่อการเรียกที่ขึ้นต่อกันแบบมีวงถูกขจัดจนหมด จึงเรียงลำดับคลาสเป็นขั้นตอนสุดท้าย

ในงานวิทยานิพนธ์นี้ได้พัฒนาเครื่องมือเพื่อทดสอบวิธีการที่นำเสนอ และทดลองเรียงลำดับคลาสสำหรับการทดสอบแบบบูรณาการจากโปรแกรมเชิงวัตถุ ผลที่ได้จากการทดลองกับกรณีศึกษา 3 ระบบ แสดงให้เห็นว่า วิธีการเรียงลำดับการบูรณาการคลาสโดยใช้เทคนิคการตัดส่วนเชิงวัตถุ มีการใช้สตับน้อยกว่าวิธีการเรียงลำดับคลาสวิธีอื่นๆ

ภาควิชา ....วิศวกรรมคอมพิวเตอร์... ลายมือชื่อนิสิต.....จุฑารัตน์ เจริญไพบูลย์กิจ.....

สาขาวิชา... วิศวกรรมคอมพิวเตอร์... ลายมือชื่ออ.ที่ปรึกษาวิทยานิพนธ์หลัก.....

ปีการศึกษา....2551.....

4970759121 : MAJOR COMPUTER ENGINEERING
KEYWORDS : INTEGRATION TESTING / TEST ORDERS / OBJECT-ORIENTED PROGRAM SLICING / SOFTWARE TESTING

JUTARAT JAROENPIBOONKIT : AN APPROACH OF ORDERING CLASS FOR INTEGRATION TESTING USING OBJECT-ORIENTED SLICING TECHNIQUE. ADVISOR : ASSOC.PROF. TARATIP SUWANNASART, PH.D., 90 pp.

Integration testing is to find component faults that cause inter-component failures. Integration testing for object-oriented software needs ordering classes to be tested. However, a problem of ordering classes is the determination of cyclic dependency calls. Most approaches proposed ordering classes by removing relationships to break cycles. Removing a relationship would require a stub to be created for testing classes at the end of the relationship. This thesis presents an approach of ordering class for integration testing using object-oriented slicing technique to break cyclic dependencies. The objective is to minimize a number of test stubs in ordering classes.

The proposed approach consists of three steps: analyzing relationship step, breaking cycle step, and sorting class step. In the analyzing relationship step, this research uses tarjan algorithm for detecting cycle, then classifies classes with cyclic dependencies to cycle type. Breaking cycles, which is the next step, uses object-oriented slicing to break cycles by cycle type. After no cycle remains, the final step is ordering classes.

This thesis develops a tool to test this approach and, to demonstrate ordering classes for integration testing from object-oriented software. The result of our experiments with three case studies indicates that an approach for ordering class integration using object-oriented slicing technique use less test stubs than other approaches.

Department:.....Computer Engineering.. Student's Signature.....
Field of Study:..Computer Engineering.. Advisor's Signature.....
Academic Year:.....2008.....

กิตติกรรมประกาศ

ขอขอบพระคุณอาจารย์ที่ปรึกษา รองศาสตราจารย์ ดร.ธาราทิพย์ สุวรรณศาสตร์ ผู้เสียสละเวลาช่วยเหลือและให้คำปรึกษา คำแนะนำที่มีประโยชน์ต่องานวิจัย และความรู้ทางวิชาการอื่นๆ ทำให้งานวิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดี

ขอขอบพระคุณคณะกรรมการสอบวิทยานิพนธ์ทุกท่าน รองศาสตราจารย์ ดร.วิวัฒน์ วัฒนาวุฒิ รองศาสตราจารย์ ดร.พรศิริ หมื่นไชยศรี ผู้ช่วยศาสตราจารย์ ดร.ภัทรชัย ลลิตโรจน์วงศ์ และผู้ช่วยศาสตราจารย์ ดร.อาทิตย์ ทองทักษ์ ที่กรุณาสละเวลาในการให้คำแนะนำเกี่ยวกับงานวิจัย และตรวจสอบความถูกต้องสมบูรณ์ของวิทยานิพนธ์ฉบับนี้

ขอขอบพระคุณคณาจารย์ทุกท่านในภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ที่ให้ความรู้ คำแนะนำในการเรียน และการทำวิจัย

ขอขอบคุณสมาชิกในห้องปฏิบัติการวิศวกรรมซอฟต์แวร์ และเพื่อนๆ ทุกคน ที่มอบกำลังใจ ความช่วยเหลือ และคำแนะนำต่างๆ อย่างสม่ำเสมอ

สุดท้ายนี้ ขอกราบขอบพระคุณบิดา มารดา และสมาชิกในครอบครัวทุกท่าน ผู้สนับสนุน ในการทำงานที่ต้องใช้ความมานะและอดทน แก่ผู้วิจัยตลอดมาจนสำเร็จการศึกษา

สารบัญ

หน้า

หน้า

หน้า

สารบัญตาราง

สารบัญภาพ

ภาพประกอบ หน้า

บทที่ 1
บทนำ

ในบทนี้ จะกล่าวถึงแนวคิดหลักของงานวิจัย อันประกอบไปด้วย ที่มาและความสำคัญ วัตถุประสงค์ ขอบเขต ประโยชน์ ขั้นตอนและโครงสร้างของเนื้อหางานวิจัย ซึ่งมีเนื้อหาดังต่อไปนี้

1.1 ที่มาและความสำคัญของงานวิจัย

การทดสอบซอฟต์แวร์ (Software Testing) ประกอบด้วยระดับของการทดสอบหลักๆ 3 ระดับ คือ การทดสอบแบบหน่วย (Unit Testing) การทดสอบแบบบูรณาการ (Integration Testing) และการทดสอบระบบ (System Testing) ซึ่งในการทดสอบแบบบูรณาการนั้น เป็นการหาข้อผิดพลาดของส่วนประกอบ (Component) ที่มีสาเหตุมาจากความขัดข้อง (Failure) ในการทำงานร่วมกันระหว่างส่วนโปรแกรม [1] ซึ่งหน้าที่หลักอย่างหนึ่งในการบูรณาการของส่วนโปรแกรมเพื่อทำการทดสอบโปรแกรมเชิงวัตถุคือการกำหนดลำดับการทดสอบของคลาส (Class) ที่ถูกนำมาทดสอบ [2] นอกจากนั้น ปัญหาของการหาลำดับการทดสอบคือการกำหนดลำดับการทดสอบของโปรแกรมที่มีการเรียกที่ขึ้นต่อกันแบบมีวง (Cyclic Dependency Call) หรือกล่าวอีกนัยหนึ่งคือ การที่คลาสใดคลาสหนึ่งที่เป็นคลาสต้นทาง (Source Class) มีการเรียกไปยังคลาสอื่นๆ แล้วคลาสที่ถูกเรียกในเส้นทางการเรียกนั้นๆ มีการเรียกกลับไปยังคลาสต้นทาง ซึ่งลักษณะของคลาสที่มีการเรียกกันเป็นวงกลมนี้จะถูกเรียกว่า ไซเคิล (Cycle)

การกำหนดลำดับการทดสอบคลาสที่อยู่ในไซเคิลทำได้ยากลำบาก เนื่องจากความสัมพันธ์ของกลุ่มคลาสมีลักษณะเรียกกันเป็นวง และถ้าจำนวนคลาสมีจำนวนมากความสัมพันธ์ระหว่างคลาสจะเรียกกันเป็นวงซ้อนกันอยู่หลายวง ทำให้ไม่สามารถกำหนดได้ว่าคลาสแรกที่ถูกทดสอบก่อนจะเป็นคลาสไหน

เพื่อที่จะแก้ปัญหาดังกล่าว นักวิจัยหลายท่านได้นำเสนอเทคนิคในการนำความสัมพันธ์ระหว่างคลาสออก โดยทำให้อยู่ในรูปแบบของกราฟระบุทิศทางแบบไม่มีวงหรือดีเอจี (Directed Acyclic Graph - DAG) และใช้การเรียงลำดับแบบทอพอโลยี (Topological Sorting) กับกราฟชนิดนี้เพื่อหาลำดับการทดสอบ ซึ่งการนำความสัมพันธ์ออกนั้นหมายถึงการที่จะต้องสร้างสตับ (Stub) เพื่อที่จะทดสอบคลาสต้นทาง ดังนั้น วัตถุประสงค์หลักของการทดสอบแบบบูรณาการคือ การลดจำนวนของสตับที่ใช้สำหรับคลาสที่ยังไม่ถูกทดสอบหรือคลาสที่ยังไม่ได้พัฒนา เพราะจำนวนของสตับก็คือ ปัจจัยหนึ่งของค่าใช้จ่ายในการทดสอบแบบบูรณาการนั่นเอง

ปัจจุบันมีการเสนอวิธีการต่างๆ เพื่อช่วยในการหาลำดับการทดสอบ อาทิเช่น Tai และ Daniels [2] หาลำดับการทดสอบจากการแบ่งคลาสเป็นระดับและเลือกลบความสัมพันธ์ที่มีค่าถ่วงน้ำหนักสูงที่สุด ซึ่งได้จากผลบวกของจำนวนความสัมพันธ์ที่เข้ามายังคลาสต้นทางกับจำนวนความสัมพันธ์ที่ออกจากคลาสปลายทาง ซึ่งจำนวนสตับคือ จำนวนของคลาสปลายทางที่

ความสัมพันธ์ที่ถูกลบนั้นชี้ไป ขณะที่ Traon et al.[3] ใช้ค่าถ่วงน้ำหนักอยู่ที่คลาสแทน แล้วเลือกลบความสัมพันธ์ที่ชี้มาที่คลาสที่มีค่าถ่วงน้ำหนักมากที่สุด สุดท้าย Briand et al. [4] หาลำดับการทดสอบคล้ายกับวิธีของ Traon et al. แต่การเลือกลบความสัมพันธ์เพื่อสร้าง สตับนั้นคล้ายกับวิธีของ Tai และ Daniels แต่ใช้ค่าถ่วงน้ำหนักเป็นผลคูณแทน รายละเอียดของวิธีการทั้งสามนั้นอยู่ในบทที่ 2 หัวข้อ 2.2.3, 2.2.4, 2.2.5 ตามลำดับ

การหาลำดับการทดสอบด้วยวิธีการที่เสนอมาข้างต้น เป็นการขจัดการเรียกที่ขึ้นต่อกันแบบมีวง โดยลบความสัมพันธ์แล้วใช้สตับในการทดสอบ ซึ่งการทดสอบคลาสโดยใช้สตับนั้น ไม่สามารถแสดงข้อผิดพลาดของการทำงานระหว่างคลาสได้ เพราะผู้พัฒนาโปรแกรมหรือผู้ทดสอบ มักเป็นผู้สร้างสตับเอง เพื่อให้การเรียกนั้นสมบูรณ์และถูกต้อง และการทดสอบคลาสโดยใช้สตับนี้เอง เมื่อเชื่อมต่อคลาสจริงแล้วทำการทดสอบระบบ ข้อผิดพลาดที่ควรจะถูกพบตั้งแต่ในระดับการทดสอบแบบบูรณาการกลับปรากฏในระดับของการทดสอบระบบ และในบางกรณีการสร้างสตับมีความยุ่งยากและใช้เวลานานเนื่องจากแต่ละค่าของข้อมูลส่งออกของสตับมีความหมาย (Semantic) ในตัวมันเอง ซึ่งมีผลต่อการทำงานของคลาส งานวิจัยนี้จึงได้นำเสนอวิธีการหาลำดับการทดสอบ เพื่อที่จะใช้คลาสจริงในการทดสอบแทนการใช้สตับและลดจำนวนสตับที่ต้องใช้จริงให้น้อยที่สุดโดยใช้เทคนิคการตัดส่วนเชิงวัตถุ และงานวิจัยนี้ใช้กราฟแสดงความสัมพันธ์ของการทดสอบในการแสดงความสัมพันธ์ระหว่างคลาส เพราะสามารถแสดงรายละเอียดในระดับของการเรียกเมท็อด ซึ่งใช้สำหรับการทดสอบบางส่วน (Partial Testing) ได้ ในขณะเดียวกันเทคนิคการตัดส่วนเชิงวัตถุ (Object-Oriented Slicing) ถูกใช้ในการขจัดการเรียกที่ขึ้นต่อกันแบบมีวง และใช้สำหรับหาลำดับการทดสอบด้วย

1.2 วัตถุประสงค์ของงานวิจัย

1) เพื่อลดจำนวนสตับในการทดสอบแบบบูรณาการ โดยใช้วิธีการเรียงลำดับคลาสสำหรับการทดสอบแบบบูรณาการโดยใช้เทคนิคการตัดส่วนเชิงวัตถุ

2) เพื่อสร้างเครื่องมือที่สามารถเรียงลำดับคลาสสำหรับการทดสอบแบบบูรณาการจากการตัดส่วนเชิงวัตถุ

1.3 ขอบเขตของงานวิจัย

1) งานวิจัยนี้ใช้รหัสต้นฉบับ (Source code) เป็นข้อมูลนำเข้าโดยพิจารณาจำนวนของคลาสที่ต้องการเรียงลำดับการทดสอบ

2) งานวิจัยนี้พิจารณาความสัมพันธ์ระหว่างคลาสชนิด อินเฮอร์ริแทนซ์ (Inheritance) คอมโพสิตชัน (Composition) แอกกรีเกชัน (Aggregation) และความสัมพันธ์แบบเกี่ยวพัน (Association) เท่านั้น

3) การตัดแบ่งคลาสในงานวิจัยนี้ใช้เทคนิคการตัดส่วนเชิงวัตถุ

4) งานวิจัยนี้ไม่ครอบคลุมความสัมพันธ์แบบโพลีมอร์ฟิก (Polymorphic) และการยึดเหนี่ยวพลวัต (Dynamic binding)

5) ในกลุ่มของคลาสที่มีการเรียกที่ขึ้นต่อกันแบบมีวงจะต้องมีเมท็อดใดเมท็อดหนึ่งที่ไม่มีการเรียกไปยังคลาสอื่น

6) คลาสแม่ของความสัมพันธ์แบบอินเฮอร์ริแทนซ์ต้องเป็นคลาสรูปธรรม (Concrete Class) เท่านั้น ไม่ครอบคลุมถึงคลาสนามธรรม (Abstract Class)

7) งานวิจัยนี้ไม่ครอบคลุมไซเคิลที่มีความสัมพันธ์แบบแอกกรีเกชันซึ่งกันและกัน

8) การตัดส่วนคลาสในงานวิจัยนี้ไม่ครอบคลุมการตัดแบ่งคลาส โดยที่คลาสแม่ของความสัมพันธ์เป็นแบบอินเฮอร์ริแทนซ์

9) เครื่องมือที่ใช้ในการหาลำดับการทดสอบจากการตัดส่วนเชิงวัตถุนี้ มีผลลัพธ์คือลำดับของคลาสสำหรับการทดสอบ ซึ่งไม่ครอบคลุมการสร้างกรณีทดสอบ

10) งานวิจัยนี้จะใช้กรณีศึกษาอย่างน้อย 3 ระบบ

1.4 ประโยชน์ของงานวิจัย

1) ได้เครื่องมือที่สามารถเรียงลำดับคลาสสำหรับการทดสอบแบบบูรณาการ

2) วิธีการหาลำดับการทดสอบที่ได้ ทำให้จำนวนสตับที่ต้องสร้างขึ้นลดลงดังนั้นเวลาของการทดสอบการบูรณาการคลาสจึงลดลงตาม

1.5 ขั้นตอนและวิธีการวิจัย

1) ศึกษาความสัมพันธ์ระหว่างคลาส และวิธีการตัดส่วนเชิงวัตถุ

2) วิเคราะห์ความสัมพันธ์ระหว่างคลาส และการตัดส่วนคลาสด้วยวิธีการตัดส่วนเชิงวัตถุ

3) ออกแบบขั้นตอนการหาลำดับการทดสอบ และวิธีการตัดส่วนเชิงวัตถุ

4) พัฒนาเครื่องมือซอฟต์แวร์ในการเรียงลำดับคลาสจากการตัดส่วนเชิงวัตถุ

5) เปรียบเทียบลำดับการทดสอบที่ได้จากงานวิจัยนี้กับงานวิจัยอื่น โดยวัดจาก

5.1) จำนวนสตับ

5.2) จำนวนขั้นการทดสอบ (Step) โดยจำนวนขั้นการทดสอบหมายถึงจำนวนคลาสของเส้นทางการทดสอบที่ยาวที่สุด

5.3) จำนวนลักษณะประจำ (Attribute) และเมท็อดของสตับเทียบกับส่วนโปรแกรมที่ถูกแบ่งส่วน

5.4) จำนวนบรรทัดของคำสั่ง (Line of code) ของสตับกับจำนวนบรรทัดของคำสั่งที่ต้องทำสำเนาเพิ่มจากการตัดส่วน

5.5) ความซับซ้อนของลำดับการทดสอบ (Test Order Complexity)

6) ตรวจสอบผลการวิจัย

7) สรุปผลการวิจัย และจัดทำวิทยานิพนธ์

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในบทนี้ จะกล่าวถึงทฤษฎีที่สำคัญ ซึ่งได้นำมาประยุกต์ สนับสนุน และใช้อ้างอิงในการทำงานวิทยานิพนธ์ รวมถึงข้อดีและข้อจำกัดของงานวิจัยต่างๆ ที่เกี่ยวข้อง โดยมีเนื้อหาดังต่อไปนี้

2.1 ทฤษฎีที่เกี่ยวข้อง

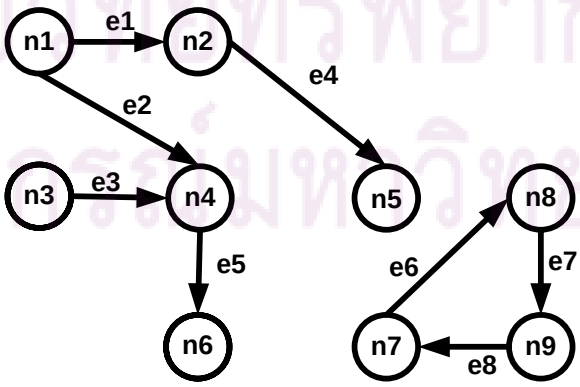
2.1.1 ทฤษฎีกราฟ (Graph Theory) [5]

ทฤษฎีกราฟเป็นสาขาหนึ่งของทอพอโลยีในเรขาคณิต ซึ่งสนใจเฉพาะโครงสร้างของรูปเรขาคณิตที่ไม่ขึ้นกับขนาด ระยะ หรือการวัดใดๆ โดยจุดเริ่มต้นของทฤษฎีกราฟนี้ มาจากปัญหาสะพานแห่งเมืองโคนิกส์เบิร์ก (The Königsberg Bridge Problem) ซึ่งเมืองนี้เกาะเล็กๆอยู่ 2 เกาะ กลางแม่น้ำ และมีสะพานเชื่อมทั้งสองฝั่งแม่น้ำกับเกาะสองเกาะนี้อยู่ 7 แห่ง คนในเมืองนั้นต้องการเดินทางออกจากบ้านข้ามสะพานทั้งเจ็ดแห่งเพียงครั้งเดียว และเดินทางกลับบ้าน ซึ่งการจำลองสถานการณ์นี้ จึงมีการเขียนออกมาเป็นกราฟ (Graph)

กราฟโดยทั่วไปแบ่งออกเป็น 2 ชนิด คือ กราฟไม่ระบุทิศทาง (Undirected Graph) และกราฟระบุทิศทาง (Directed Graph) เนื่องจากความสัมพันธ์ระหว่างคลาสเป็นการเรียกแบบมีทิศทาง ดังนั้น งานวิจัยนี้จึงใช้กราฟระบุทิศทางในการแสดงความสัมพันธ์ของคลาส

2.1.1.1 กราฟระบุทิศทาง

จากคำนิยาม กราฟระบุทิศทาง G = (V, E) โดยที่ V คือ กลุ่มของโหนดหรือจุดต่อ (Node or Vertex) V = $\{n_1, n_2, \ldots, n_m\}$ [6] โหนดหมายถึงคลาส และ E คือ กลุ่มของเส้นเชื่อม (Edge) E = $\{e_1, e_2, \ldots, e_p\}$ ในงานวิจัยนี้ เส้นเชื่อมหมายถึงความสัมพันธ์ระหว่างคลาส และในเส้นเชื่อมที่มีทิศทางใดๆ $e_k = <n_i, n_j>$ เป็นคู่อันดับของโหนด $n_i, n_j \in V$ ซึ่ง n_i คือ โหนดเริ่มต้น (Initial หรือ Start Node) และ n_j คือ โหนดปลายทาง (Terminal or Finish Node) ตัวอย่างกราฟระบุทิศทางแสดงดังรูปที่ 2.1



รูปที่ 2.1 กราฟระบุทิศทาง

2.1.1.2 ดีกรีของโหนด

โหนดในกราฟระบุทิศทางมีการแสดงระดับการเข้าและออกของโหนดดังต่อไปนี้

1) อินดีกรี (Indegree) คือ จำนวนของเส้นเชื่อมที่เข้ามาที่โหนด n ใดๆ โดยโหนดนั้นจะถูกพิจารณาเสมือนเป็นโหนดปลายทาง อินดีกรีของโหนดเขียนอยู่ในรูป indeg(n)

2) เอาท์ดีกรี (Outdegree) คือ จำนวนของเส้นเชื่อมที่ออกจากโหนด n ใดๆ โดยโหนดนั้นจะถูกพิจารณาเสมือนเป็นโหนดเริ่มต้น เอาท์ดีกรีของโหนดเขียนอยู่ในรูป outdeg(n)

2.1.1.3 ชนิดของโหนด

ในการอธิบายกราฟระบุทิศทางมีการกำหนดลักษณะของโหนดดังต่อไปนี้

1) โหนดต้นทาง (Source Node) คือ โหนดที่มีอินดีกรีเท่ากับศูนย์ จากรูปที่ 1 คือ โหนด n1 และ n3

2) โหนดปลายทาง (Sink Node) คือ โหนดที่มีเอาท์ดีกรีเท่ากับศูนย์ จากรูปที่ 1 คือ โหนด n5 และ n6

3) โหนดส่งผ่าน (Transfer Node) คือ โหนดที่มีอินดีกรีและเอาท์ดีกรีไม่เท่ากับศูนย์ จากรูปที่ 1 คือ โหนด n2 และ n4

2.1.1.4 วิถี (Path) และ กึ่งวิถี (Semipath)

ทิศทางของการเชื่อมโหนดซึ่งบอกวิถีในกราฟระบุทิศทาง มีนิยามดังต่อไปนี้

1) วิถี คือ ลำดับของเส้นเชื่อม ซึ่งคู่ของเส้นเชื่อมที่ติดกันใดๆ e_i, e_j จะเรียงลำดับก็ต่อเมื่อโหนดปลายทางของเส้นเชื่อมแรก e_i เป็นโหนดต้นทางของเส้นเชื่อมที่สอง e_j จากรูปที่ 1 ตัวอย่างวิถีคือ วิถีจาก n1 ถึง n6

2) ไซเคิล คือ วิถีระบุทิศทางที่มีโหนดเริ่มต้นและโหนดปลายทางเป็นโหนดเดียวกัน จากรูปที่ 1 ตัวอย่างไซเคิลคือ วิถีจาก n7, n8, n9

3) กึ่งวิถี คือ ลำดับของเส้นเชื่อม ซึ่งมีอย่างน้อยหนึ่งคู่เส้นเชื่อมใดๆ e_i, e_j มีโหนดต้นทางเป็นโหนดเดียวกัน หรือมีโหนดปลายทางเป็นโหนดเดียวกัน จากรูปที่ 1 ตัวอย่างกึ่งวิถีคือ กึ่งวิถีระหว่าง n1 และ n3

2.1.1.5 n-Connectedness

การเชื่อมสองโหนดใดๆ ในกราฟระบุทิศทางมีดังต่อไปนี้

1) 0-connected คือ การที่ไม่มีวิถีใดๆ ที่เชื่อมระหว่างโหนด n_i และ n_j จากรูปที่ 1 คือ n1 และ n7

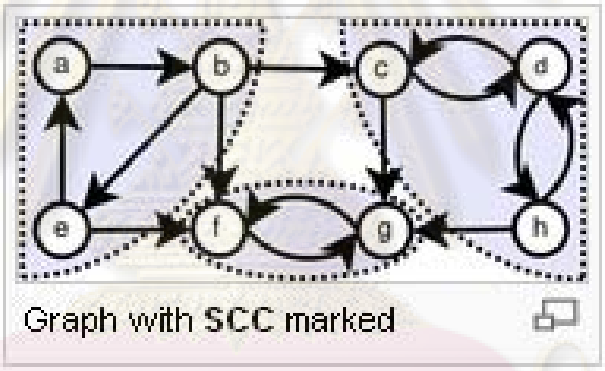
2) 1-connected คือ การที่มีกิ่งวิถีใดๆ ที่เชื่อมระหว่างโหนด n_i และ n_j แต่ไม่มีวิถีที่เชื่อมได้โดยตรง จากรูปที่ 1 คือ n2 และ n6

3) 2-connected คือ การที่มีวิถีใดๆ ที่เชื่อมระหว่างโหนด n_i และ n_j จากรูปที่ 1 คือ n1 และ n6

4) 3-connected คือ การที่มีวิถีใดๆ ที่เชื่อมจากโหนด n_i ไป n_j และมีวิถีที่เชื่อมจากโหนด n_j ไป n_i จากรูปที่ 1 คือ n7 และ n9

2.1.1.6 กลุ่มขององค์ประกอบแบบแข็ง (Strong Component)

กลุ่มขององค์ประกอบแบบแข็ง คือ กลุ่มของโหนดในกราฟระบุทิศทางที่มีการเชื่อมกันแบบ 3-connected ซึ่งเป็นลักษณะของการวนซ้ำ (loop) จากรูปที่ 2.1 คือ กลุ่มของ n3, n4, n6 ซึ่งบางครั้งเรียกอีกอย่างว่ากลุ่มของส่วนประกอบที่มีการเชื่อมต่อกันสูงหรือเอสซีซี (Strongly Connected Components: SCCs) การเชื่อมกันลักษณะนี้ก่อให้เกิดปัญหาในการหาลำดับการทดสอบ ดังนั้นจึงต้องมีการแก้ปัญหาดังกล่าวโดยเอาการลบเส้นเชื่อมบางเส้นเพื่อไม่ให้เกิดการวนซ้ำ จากนั้นจึงสร้างกราฟดีเอจีในการหาลำดับทางทอพอโลยี ตัวอย่างเอสซีซีแสดงดังรูปที่ 2.2



รูปที่ 2.2 กลุ่มของส่วนประกอบที่มีการเชื่อมต่อกันสูง หรือ เอสซีซี

2.1.2 การทดสอบซอฟต์แวร์เชิงวัตถุ (Object-Oriented Software Testing)

การพัฒนาซอฟต์แวร์ด้วยการเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming) ซึ่งมีคุณสมบัติอย่างเช่น การห่อหุ้มหรือเอนแคปซูเลชัน (Encapsulation) การรับทอดหรืออินเฮอร์ริแทนซ์ (Inheritance) และ ภาวะพหุสัณฐานหรือโพลีมอร์ฟิซึม (Polymorphism) ทำให้การทดสอบเพื่อหาข้อผิดพลาดของโปรแกรมมีความยากลำบากขึ้น ลักษณะดังกล่าวก่อให้เกิดข้อผิดพลาด [1] ดังตัวอย่างต่อไปนี้

- การยึดเหนี่ยวพลวัต (Dynamic Binding) และโครงสร้างอินเฮอร์ริแทนซ์แบบซับซ้อน ทำให้การเกิดข้อผิดพลาดจากการยึดเหนี่ยวที่ไม่ได้คาดไว้ (Unanticipated Binding) หรือการตีความที่ไม่ถูกต้องของการใช้งาน

- ข้อผิดพลาดของส่วนต่อประสาน (Interface Error) เป็นข้อผิดพลาดโดยทั่วไปที่เกิดขึ้นในภาษาเชิงกระบวนคำสั่ง (Procedural Language) แต่ในภาษาเชิงวัตถุ (Object-Oriented Language) นั้น ประกอบด้วยส่วนโปรแกรมย่อยๆ เป็นจำนวนมาก ทำให้

ส่วนต่อประสานของส่วนโปรแกรมในภาษานี้มีมากกว่าภาษาเชิงกระบวนคำสั่ง ดังนั้นข้อผิดพลาดของส่วนต่อประสานเหล่านี้ จึงมีมากตาม

- เนื่องจากวัตถุมีการคงสภาพของสถานะ แต่การควบคุมสถานะนั้น มีการกระจายอยู่ทั่วทั้งโปรแกรม ทำให้การควบคุมสถานะเกิดข้อผิดพลาดได้ง่าย

จากตัวอย่างข้อผิดพลาดดังกล่าว ทำให้การทดสอบซอฟต์แวร์เชิงวัตถุต้องการเทคนิคเฉพาะทางในการทดสอบ งานวิจัยนี้เป็นการทดสอบแบบบูรณาการ ดังนั้นจึงมีการพิจารณาถึงสาเหตุและลักษณะข้อผิดพลาดของส่วนต่อประสาน และในการทดสอบส่วนต่อประสานจะมีการสร้างสตับขึ้นเพื่อทดสอบการเชื่อมต่อของแต่ละส่วนโปรแกรม โดยสตับก็คือส่วนโปรแกรมที่สร้างขึ้นมาโดยจำลองพฤติกรรมการทำงานของโปรแกรมจริง เพื่อให้ส่วนโปรแกรมจริงอื่นๆ สามารถเรียกใช้งานได้อย่างถูกต้อง

2.2 งานวิจัยที่เกี่ยวข้อง

2.2.1 ทาร์จันอัลกอริทึม (Tarjan's Algorithm) [7]

ทาร์จันอัลกอริทึมเป็นอัลกอริทึมในทฤษฎีกราฟที่ใช้ในการหากลุ่มเอสซีซี อัลกอริทึมนี้มีพื้นฐานมาจากการค้นตามแนวลึก (Depth First Search: DFS) และมีการใช้กองซ้อน (Stack) เก็บโหนดในการหาเอสซีซี แต่ละโหนด (Vertices) จะถูกกำหนดหมายเลขดัชนีการค้นหาตามแนวลึก v.index โดยหมายเลขดังกล่าวจะต่อเนื่องกันตามลำดับการเข้าถึง และแต่ละโหนดยังมีการกำหนดค่า v.lowlink โดยให้ v.lowlink := min{v'.index: v' is reachable from v} ถ้า v.lowlink = v.index แล้วโหนดที่อยู่ในกองซ้อนคือโหนดที่อยู่ในกลุ่มเอสซีซี รหัสเทียม (Pseudocode) ของทาร์จันอัลกอริทึมแสดงดังรูปที่ 2.3

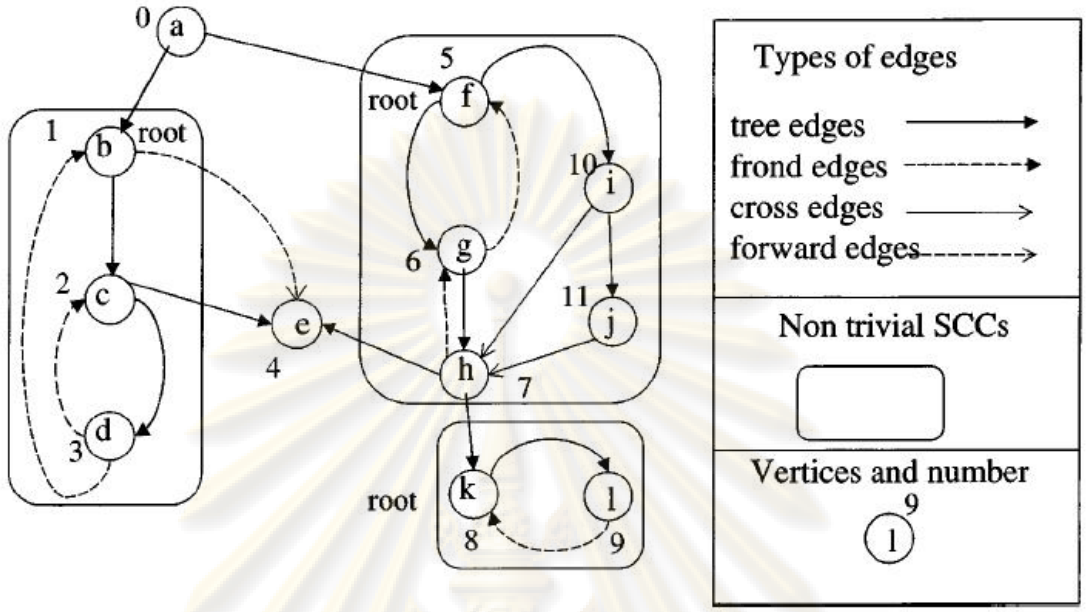
```
Input: Graph G = (V, E), Start node v0

index = 0                          // DFS node number counter
S = empty                          // An empty stack of nodes
tarjan(v0)                         // Start a DFS at the start node

procedure tarjan(v)
  v.index = index                  // Set the depth index for v
  v.lowlink = index
  index = index + 1
  S.push(v)                        // Push v on the stack
  forall (v, v') in E do           // Consider successors of v
    if (v'.index is undefined)     // Was successor v' visited?
      tarjan(v')                   // Recurse
      v.lowlink = min(v.lowlink, v'.lowlink)
    elseif (v' in S)               // Is v' on the stack?
      v.lowlink = min(v.lowlink, v'.lowlink)
  if (v.lowlink == v.index)        // Is v the root of an SCC?
    print "SCC:"
    repeat
      v' = S.pop
      print v'
    until (v' == v)
```

รูปที่ 2.3 รหัสเทียมของทาร์จันอัลกอริทึม

ตัวอย่างกลุ่มเอสซีซีที่ได้จากการใช้ทาร์จันอัลกอริทึมในการตรวจจับ แสดงดังรูปที่ 2.4



รูปที่ 2.4 ทาร์จันอัลกอริทึม และ ชนิดของเส้นเชื่อม

การใช้ทาร์จันอัลกอริทึมในการตรวจจับเอสซีซีมีการกำหนดนิยามเส้นเชื่อมเป็น 4 กลุ่ม ดังต่อไปนี้

1) เส้นเชื่อมแบบต้นไม้ (Tree Edges) คือเส้นที่ลากจากโหนดหนึ่งไปยังโหนดที่ยังไม่ได้แวะผ่าน

2) เส้นเชื่อมแบบไปข้างหน้า (Forward Edges) คือเส้นที่เชื่อมโหนดนอกเอสซีซี

3) เส้นเชื่อมฟรอนด์ (Frond Edges) คือเส้นที่เชื่อมกลับไปยังโหนดต้นทางที่แวะผ่านแล้ว

4) เส้นเชื่อมข้าม (Cross Edges) คือ เส้นที่เชื่อมระหว่างโหนดในต้นไม้ส่วนย่อย (Subtree) ที่ต่างกัน แต่อยู่ในเอสซีซีเดียวกัน

จากรูปที่ 2.4 Non trivial SCCs คือกลุ่มเอสซีซีย่อยในกราฟที่มีมากกว่า 1 โหนด หลังจากการตรวจจับด้วยทาร์จันอัลกอริทึม และโหนดตัวอย่างในกรอบด้านขวาล่างคือโหนดแอล (l) โดยมีหมายเลขการเข้าถึงคือหมายเลข 9 และในแผนภาพต้นไม้ด้านซ้ายของรูปที่ 2.4 มีโหนด b, f, k เป็นโหนดราก (Root Node) หมายถึง โหนดแรกในกลุ่มเอสซีซี ที่ถูกแวะผ่านจากการค้นตามแนวลึก

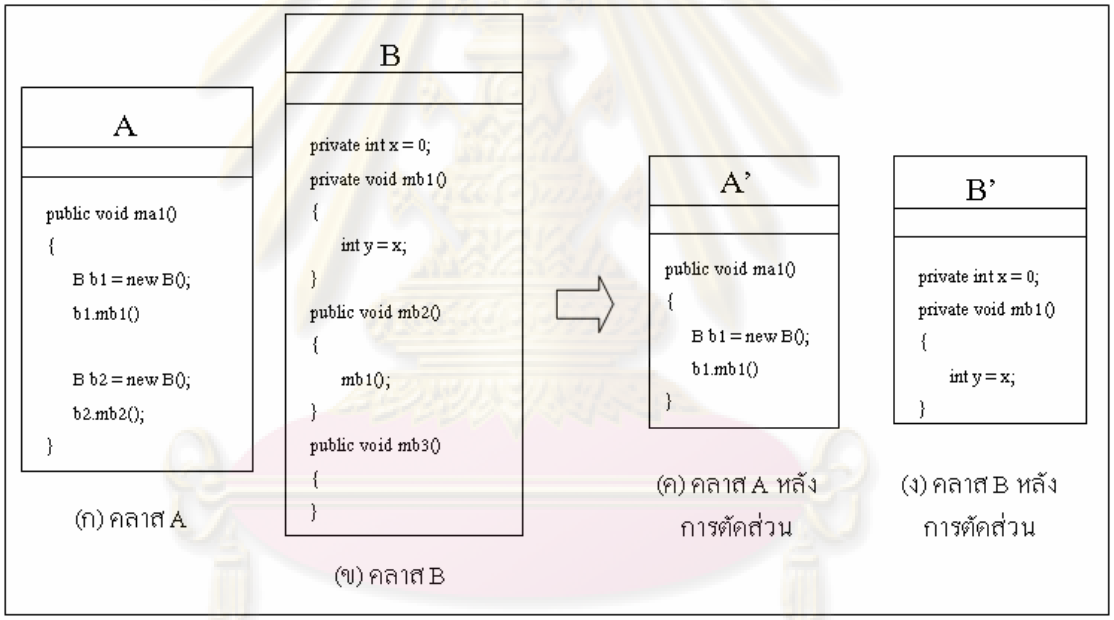
2.2.2 การตัดส่วนเชิงวัตถุ

การตัดส่วนโปรแกรม (Program Slicing) คือเทคนิคอย่างหนึ่งในการวิเคราะห์โปรแกรม โดยทำการแยกส่วนที่สนใจออกจากตัวโปรแกรม เทคนิคนี้ถูกนำมาประยุกต์ใช้งานทางวิศวกรรมซอฟต์แวร์ต่างๆ เช่นการทำความเข้าใจโปรแกรม และการแก้จุดบกพร่อง

(Debugging) เป็นต้น เทคนิคการตัดส่วนโปรแกรมนั้นถูกใช้ในโปรแกรมกระบวนคำสั่งแบบดั้งเดิม (Traditional Procedural Program) แต่สำหรับโปรแกรมเชิงวัตถุที่มีคุณสมบัติอย่างเช่น การห่อหุ้ม การรับทอด และโพลีมอร์ฟิซึม ทำให้ความสัมพันธ์ระหว่างส่วนต่างๆ ภายในโปรแกรมมีมากกว่าโปรแกรมกระบวนคำสั่ง ดังนั้นจึงเกิดแนวคิดการตัดส่วนเชิงวัตถุ [8, 9] สำหรับซอฟต์แวร์ที่พัฒนาด้วยการเขียนโปรแกรมเชิงวัตถุ Chen และ Xu [8] เสนอวิธีการสำหรับตัดส่วนโปรแกรมเชิงวัตถุภาษาจาวา ดังนี้

2.2.2.1 การตัดส่วนวัตถุ (Object Slicing)

การตัดส่วนวัตถุ คือการตัดส่วนโดยสนใจเฉพาะวัตถุใดวัตถุหนึ่งของคลาสนั้นๆ การตัดส่วนวัตถุนี้จะทำการดึงเฉพาะข้อความสั่ง(Statement) ในเมท็อดที่มีผลกระทบกับวัตถุนั้นออกมา การตัดส่วนวัตถุนี้จะทำการดึงข้อความสั่งในคลาสต้นทางที่มีการประกาศวัตถุ และตัดส่วนคลาสปลายทางเฉพาะที่วัตถุนั้นมีการเข้าถึง ตัวอย่างการตัดส่วนวัตถุแสดงดังรูปที่ 2.5



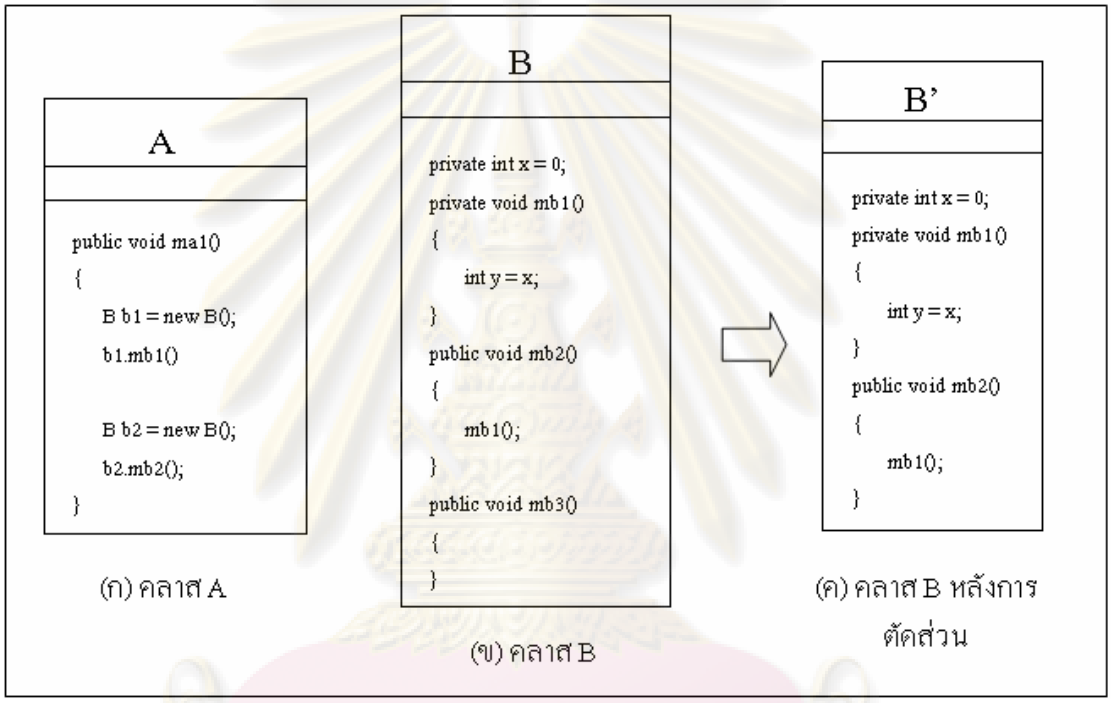
รูปที่ 2.5 การตัดส่วนวัตถุ

รูปที่ 2.5 แสดงการตัดส่วนวัตถุ b1 ของคลาส B ที่มีการประกาศในรหัสต้นฉบับของคลาส A จากรูปที่ 2.5 (ก) คลาส B มีการสร้างวัตถุ b1 และ b2 ขึ้น แต่ในการตัดส่วนวัตถุในครั้งนี้พิจารณาเฉพาะวัตถุ b1 ดังนั้นข้อความสั่งในคลาส A ที่เกี่ยวข้องกับวัตถุ b1 จึงถูกดึงออกมา ผลของการตัดส่วนวัตถุในคลาส A แสดงดังรูปที่ 2.5 (ค)

ในรหัสต้นฉบับของรูปที่ 2.5(ก) วัตถุ b1 มีการเข้าถึงเฉพาะเมท็อด mb1 ของคลาส B และรูปที่ 2.5 (ข) แสดงคลาส B ที่มีเมท็อดทั้งหมด 3 เมท็อด แต่เมื่อทำการตัดส่วนในคลาส B จึงทำการดึงเมท็อด mb1 และสมาชิก x ที่เกี่ยวข้องกับเมท็อด mb1 ออกมา ผลของการตัดส่วนวัตถุในคลาส B แสดงดังรูปที่ 2.5 (ง)

2.2.2.2 การตัดส่วนคลาส (Class Slicing)

การตัดส่วนคลาส คือการตัดบางส่วนคลาสออกมาโดยส่วนของคลาสที่ถูกตัดส่วนออกมาจะแยกจากคลาสเดิมอย่างอิสระ การตัดส่วนคลาสนี้เริ่มต้นจากการพิจารณาข้อความสั่งในคลาสต้นทางก่อน จากนั้นทำการค้นหาสมาชิกที่เป็นข้อมูลและเมท็อดของคลาสปลายทางที่อาจจะมีอิทธิพลกับข้อความสั่งของคลาสต้นทางออกมา สมาชิกและเมท็อดดังกล่าวของคลาสจะถูกตัดส่วนออกมาโดยไม่แยกความแตกต่างของวัตถุที่มาจากคลาสเดียวกัน ในงานวิจัยนี้สนใจเฉพาะการตัดส่วนคลาสเท่านั้น เนื่องจากการวิเคราะห์ความสัมพันธ์ระหว่างคลาสจะพิจารณาทุกวัตถุที่มาจากคลาสเดียวกัน ตัวอย่างการตัดส่วนวัตถุแสดงดังรูปที่ 2.6



รูปที่ 2.6 การตัดส่วนคลาส

จากรูปที่ 2.6 แสดงการตัดส่วนคลาส B ที่ถูกเรียกจากคลาส A จากรูปที่ 2.6 (ก) คลาส B มีการสร้างวัตถุ b1 และ b2 ซึ่งเรียกเมท็อดที่แตกต่างกัน แต่ในการตัดส่วนคลาสจะไม่มีการแยกความแตกต่างของวัตถุ ดังนั้นจึงพิจารณาทั้ง 2 เมท็อดนั้นรวมกัน ในรหัสต้นฉบับของคลาส B ในรูปที่ 2.6 (ข) เมท็อด mb1 และ mb2 จะถูกดึงออกมา จากนั้นจึงพิจารณาเมท็อดและสมาชิกที่เกี่ยวข้องกับ 2 เมท็อดนี้ เนื่องจากเมท็อด mb2 มีการเรียกเมท็อด mb1 แต่เมท็อด mb1 เป็นเมท็อดที่ถูกพิจารณาไปแล้วจึงข้ามไปพิจารณาส่วนอื่นที่เกี่ยวข้องต่อไป สมาชิก x เกี่ยวข้องกับเมท็อด mb1 ดังนั้นสมาชิก x จึงถูกดึงออกมาด้วย สำหรับเมท็อด mb3 ไม่ได้มีการเข้าถึงจากคลาส A และไม่มีส่วนเกี่ยวข้องกับเมท็อด mb1 และ mb2 ดังนั้นเมท็อด mb3 จึงไม่ถูกทำการตัดส่วนออกมา ผลของการตัดส่วนคลาส B ที่ถูกเรียกจากคลาส A แสดงดังรูปที่ 2.6 (ค)

2.2.3 Test Order for Inter-Class Integration Testing of Object-Oriented Software โดย K.-C. Tai และ F. J. Daniels [2]

งานวิจัยนี้ได้นำเสนอการหาลำดับการทดสอบโดยแบ่งคลาสออกเป็นระดับหลัก (Major level) และระดับย่อย (Minor level) โดยการกำหนดระดับนี้มีขั้นตอนการทำงานหลักๆ อยู่สองขั้นตอน คือ

- คลาสที่อยู่ในระดับหลักจะมีแค่ความสัมพันธ์แบบรับทอด และความสัมพันธ์แบบแอกกรีเกชันเท่านั้น
- ในระดับหลักเดียวกันจะแบ่งแต่ละคลาสออกเป็นระดับย่อย โดยมีความสัมพันธ์แบบเกี่ยวพัน (Association) เท่านั้น

การแสดงความสัมพันธ์ระหว่างคลาสของงานวิจัยนี้ใช้แผนภาพความสัมพันธ์วัตถุหรือโออาร์ดี (Object Relation Diagram or ORD) ในการนำเสนอ ซึ่งหลังจากการกำหนดระดับดังกล่าวแล้ว เมื่อการเรียกที่ขึ้นต่อกันแบบมีวงเกิดขึ้น งานวิจัยนี้จะเลือกลบความสัมพันธ์แบบเกี่ยวพันธ์เพื่อขจัดการเรียกที่ขึ้นต่อกันแบบมีวง สำหรับคลาสปลายทางของความสัมพันธ์ที่ถูกลบออกจะถูกสร้างเป็นสตับสำหรับใช้ทดสอบ เมื่อลบความสัมพันธ์จนกระทั่งไม่พบการเรียกที่ขึ้นต่อกันแบบมีวงแล้ว จะใช้การเรียงลำดับแบบทอพอโลยีในการหาลำดับการทดสอบ

สำหรับการเลือกลบความสัมพันธ์เพื่อขจัดการเรียกที่ขึ้นต่อกันแบบมีวงนั้นจะมีการใช้ฟังก์ชันถ่วงน้ำหนัก (Weight function) กำหนดค่าให้แต่ละความสัมพันธ์แบบเกี่ยวพันธ์ โดยการคำนวณการถ่วงน้ำหนัก (Weight computation) ของวิธีนี้ คือ ผลรวมของความสัมพันธ์ที่เข้ามาที่คลาสต้นทางและความสัมพันธ์ที่ออกจากคลาสปลายทาง ความสัมพันธ์ที่มีน้ำหนักมากที่สุดจะถูกลบออกเพื่อขจัดการเรียกที่ขึ้นต่อกันแบบมีวง

2.2.4 Efficient Object-Oriented Integration and Regression Testing โดย Y. Le Traon, T. Jéron. J.-M. Jézéquel, และ P. Morel [3]

งานวิจัยของ Traon et al. นี้ได้นำเสนอแบบจำลองและวิธีการวางแผนสำหรับการทดสอบแบบบูรณาการและการทดสอบกลับคืน (Regression Testing) ซึ่งแบบจำลองที่ใช้แสดงความสัมพันธ์ระหว่างคลาส เป็น กราฟแสดงความสัมพันธ์ของการทดสอบหรือทีดีจี (Test Dependency Graph: TDG) สำหรับการวางแผนสำหรับการทดสอบแบบบูรณาการคือ การหาลำดับการทดสอบโดยใช้ทาร์จันอัลกอริทึม [7] ซึ่งมีพื้นฐานมาจากการค้นตามแนวลึกในการหากลุ่มของส่วนประกอบที่มีการเชื่อมต่อกันสูง และสำหรับการขจัดการเรียกที่ขึ้นต่อกันแบบมีวง วิธีนี้จะเลือกลบความสัมพันธ์ที่เข้ามายังคลาสที่มีค่าถ่วงน้ำหนักสูงที่สุด ซึ่งค่าดังกล่าวคือจำนวนความสัมพันธ์แบบฟรอนด์ (Frond Dependency) ของคลาสนั้นๆ ซึ่งความสัมพันธ์แบบฟรอนด์คือความสัมพันธ์ที่ถูกกำหนดจากคลาสหนึ่งไปยังอีกคลาสหนึ่ง โดยคลาสนั้นได้ถูกเดินทางผ่านไปแล้วจากการค้นหาตามแนวลึก

กราฟแสดงความสัมพันธ์ของการทดสอบใช้ในการอธิบายการตัดส่วนคลาสในงานวิจัยนี้ เพราะการตัดส่วนคลาสนั้นมีการตัดในระดับของเมท็อด และกราฟนี้ก็แสดง

ความสัมพันธ์ของคลาสในระดับ เมท็อด แต่เนื่องจากการแปลงความสัมพันธ์แบบ คอมโพสิตชัน (Composition) และภาพรวมกลุ่มหรือแอกกรีเกชัน (Aggregation) จากแผนภาพคลาสของยูเอ็มแอล (UML Class Diagram) ไปเป็นกราฟแสดงความสัมพันธ์ของการทดสอบของ Traon et al. [5] จะอยู่ในลักษณะการเรียกซึ่งกันและกัน แต่งานวิจัยนี้จะกำหนดความสัมพันธ์เหล่านี้ให้อยู่ในลักษณะการเรียกทางเดียว สำหรับการนิยามความสัมพันธ์แบบคอมโพสิตชันและแบบแอกกรีเกชันนั้น ให้คลาสด้านที่ผูกติดกับสัญลักษณ์เพชรเป็นคลาสต้นทาง ในขณะที่คลาสอีกด้านนึงที่ผูกติดกับเส้นทึบเป็นคลาสปลายทาง

กราฟแสดงความสัมพันธ์ของการทดสอบเป็นแบบจำลองความสัมพันธ์ที่ขึ้นต่อกันตามโครงสร้างหลักระหว่างคอมโพเน็นต์ ซึ่งก็คือ คลาสหรือ เมท็อด ในระบบเชิงวัตถุ นิยามโดยทั่วไปของความสัมพันธ์ที่ขึ้นต่อกันของการทดสอบในกราฟแสดงความสัมพันธ์ของการทดสอบ มีดังนี้

- TD หรือ Test Dependency คือ การขึ้นต่อกันในการทดสอบ
- CD หรือ Contractual Dependency คือ การขึ้นต่อกันโดยมีสัญญาผูกพัน
- ID หรือ Implementation Dependency คือ การขึ้นต่อกันแบบมีการทำให้เกิดผล
- CCD หรือ Client Contractual Dependency คือ การขึ้นต่อกันโดยมีสัญญาผูกพันกับผู้เรียก

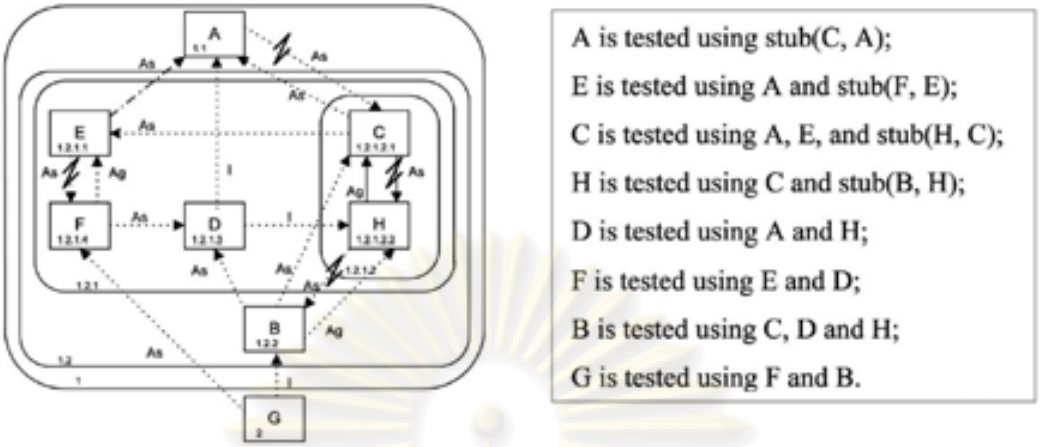
ชนิดของการขึ้นต่อกันนี้ถูกแบ่งกลุ่มเป็น 3 ชนิดของการขึ้นต่อกันในการทดสอบ ดังนี้

1. จากคลาสไปที่คลาส (Class-to-Class)
2. จากเมท็อดไปที่คลาส (Method-to-Class)
3. จากเมท็อดไปที่เมท็อด (Method-to-Method)

2.2.5 Revisiting Strategies for Ordering Class Integration Testing in the Presence of Dependency Cycles โดย L.C Briand, Y. Labiche และ Y. Wang [4]

งานวิจัยของ Briand et al. นี้มีพื้นฐานมาจากวิธีของ Y. Le Traon et al. และ วิธีของ Tai กับ Daniels โดยอันดับแรก มีการใช้ทาร์จันอัลกอริทึมในการหากลุ่มของเอสซีซี หลังจากนั้นในการขจัดการเรียกที่ขึ้นต่อกันแบบมีวง จะเลือกลบความสัมพันธ์เฉพาะความสัมพันธ์แบบเกี่ยวพันเท่านั้น ซึ่งความสัมพันธ์นั้นจะต้องมีค่าถ่วงน้ำหนักที่สูงที่สุดในกลุ่มของคลาสที่พิจารณา และจะทำซ้ำไปเรื่อยๆ จนกว่าการเรียกที่ขึ้นต่อกันแบบมีวงจะหมดไป

สำหรับการคำนวณการถ่วงน้ำหนักจะมีความคล้ายคลึงกับวิธีของ Tai กับ Daniels แต่มีความแตกต่างกัน นั่นคือ ค่าถ่วงน้ำหนักเกิดจากจำนวนความสัมพันธ์ที่เข้ามาที่คลาสต้นทางคูณกับจำนวนความสัมพันธ์ที่ออกจากคลาสปลายทาง ตัวอย่างการจัดลำดับการทดสอบของงานวิจัยนี้ แสดงดังรูปที่ 2.7



รูปที่ 2.7 ความสัมพันธ์ที่ถูกลบออกและลำดับการทดสอบของ Briand et al. [4]

จากรูปที่ 2.7 การทดสอบการเชื่อมต่อคลาส A กับ C เริ่มด้วยการทดสอบคลาส A โดยใช้ stub(C, A) ซึ่งหมายถึงสตับของคลาส C สำหรับทดสอบคลาส A จากการเชื่อมต่อดังกล่าว ยังไม่ใช่การเชื่อมต่อจริงๆ เนื่องจากคลาส A ไม่ได้ทดสอบด้วยตัวของคลาส C เอง ทำให้การทดสอบนี้ยังไม่ครอบคลุมเพียงพอ คลาสที่ทดสอบโดยใช้สตับอาจจะยังมีข้อผิดพลาดในการเชื่อมต่อจริงอยู่ และถ้ามีคลาสที่ต้องทดสอบโดยใช้สตับมาก หมายความว่าข้อผิดพลาดจะมีมากตาม นอกจากนั้นการสร้างสตับสำหรับการทดสอบมักมีความซับซ้อนและใช้เวลานาน เนื่องจากข้อมูลส่งออกของสตับมีผลต่อการทำงานของคลาสที่เรียกใช้สตับ ซึ่งแต่ละกรณีทดสอบสตับต้องให้ค่าที่แตกต่างกัน ผู้ทดสอบต้องมีความรู้ของคลาสนั้นเป็นอย่างดีในการกำหนดค่าให้สตับ และยิ่งกรณีทดสอบมีเป็นจำนวนมาก ทำให้ต้องใช้เวลามากในการสร้างสตับ ดังนั้นงานวิจัยนี้จึงเสนอแนวทางในการทดสอบโดยใช้คลาสจริง เพื่อให้การทดสอบมีความถูกต้องมากยิ่งขึ้น และลดการสร้างสตับ ซึ่งเป็นการลดแรงงานและเวลาที่ใช้ในการทดสอบ

2.2.6 ตัววัดความซับซ้อนของลำดับการทดสอบ (Test Order Complexity Metric)

ตัววัดความซับซ้อนของลำดับการทดสอบเป็นวิธีวัดค่าความซับซ้อนจากผลรวมของค่าความซับซ้อนของสตับที่ต้องสร้างขึ้น ซึ่งการวัดค่าดังกล่าวมาจากงานวิจัยของ A. Abdurazik และ J. Offutt [10] ซึ่งมีพื้นฐานมากจากตัววัดของ Briand et al. ในงานวิจัย [11]

การวัดคัปปลิง (Coupling measure) เป็นการกำหนดมาตรวัดความสัมพันธ์ระหว่างคลาสต้นทางและคลาสปลายทางใน 4 ลักษณะ ดังนี้

1) จำนวนของตัวแปรที่ถูกเรียกที่ไม่ซ้ำกัน, V_d
2) จำนวนของเมท็อดที่ถูกเรียกที่ไม่ซ้ำกัน, M_d (รวมถึงตัวสร้าง (Constructor))
3) จำนวนของรีเทิร์นไทป์ที่ส่งที่ไม่ซ้ำกัน, R_d
4) จำนวนของตัวแปรที่ไม่ซ้ำกัน, P_d

ซึ่งแสดงดังสมการที่ 1 โดยการวัด 4 ลักษณะดังกล่าวใช้สัญกรณ์แบบจุด (Dot notation) ในการแสดงว่ามาตรวัดทั้ง 4 ตัวนั้นอิสระจากกันแต่ยังมีความเกี่ยวข้องกัน

$$CM(c_i, c_j) = C \cdot V_d \cdot M_d \cdot R_d \cdot P_d \quad (1)$$

กำหนดให้ c_i และ c_j เป็นตัวแทนของทั้งสองคลาสที่คัปปลิงด้วยกัน ชนิดของคัปปลิงมี 9 ชนิด ได้แก่ Association Coupling, Aggregation Coupling, Composition Coupling, Usage Dependency Call Coupling, Global Coupling, Inheritance Coupling, Interface Realization Coupling, External Coupling, Exception Coupling ซึ่งค่าคงที่สำหรับแต่ละความสัมพันธ์ดังนี้

$$C = \begin{cases} 5, & \text{when coupling is based on} \\ & \text{inheritance or composition} \\ 1, & \text{for other coupling types} \end{cases}$$

การวัดคัปปลิงกับความสัมพันธ์ระหว่างคลาส (edge or e_i) วัดจากผลรวมของชนิดของคัปปลิงทั้ง 9 ชนิด ซึ่งการวัดคัปปลิงดังกล่าวแสดงดังสมการที่ 2

$$\begin{aligned} cm_{e_i} &= \left\{ \sum_{k=1}^{9} CM_k(v_m, v_n) \mid v_m, v_n \in V, e_i = v_m \rightarrow v_n, e_i \in E \right\} \\ &= \max(C) \cdot \sum_{k=1}^{9} V_{d_k} \cdot \sum_{k=1}^{9} M_{d_k} \cdot \sum_{k=1}^{9} R_{d_k} \cdot \sum_{k=1}^{9} P_{d_k} \end{aligned} \quad (2)$$

การวัดค่าความซับซ้อน $Cplx()$ มีการทำให้เป็นบรรทัดฐาน (Normalization) ซึ่งอยู่ในรูป $\overline{Cplx()}$ ดังสมการที่ 3

$$\overline{Cplx(i,j)} = \frac{Cplx(i,j)}{(Cplx_{\max} - Cplx_{\min})} \quad (3)$$

กำหนดให้ $Cplx(i,j)$ เป็นตัวแทนค่าความซับซ้อน ค่าความซับซ้อนที่น้อยที่สุดคำนวณจาก $Cplx_{\min} = Min\{Cplx(i,j), i, j = 1,2,...\}$ ค่าความซับซ้อนที่มากที่สุดคำนวณจาก $Cplx_{\max} = Max\{Cplx(i,j), i, j = 1,2,...\}$ จากการคำนวณค่าความซับซ้อนดังกล่าว เมื่อนำมาประยุกต์กับการวัดคัปปลิงจะเป็นการคำนวณความซับซ้อนของสตับ ตามสมการที่ 4

$$\begin{aligned} SCplx(i,j) = \quad & C + (W_V \times \overline{V}(i,j)^2 + W_M \times \overline{M}(i,j)^2 \\ & + W_R \times \overline{R}(i,j)^2 + W_P \times \overline{P}(i,j)^2)^{1/2} \end{aligned} \quad (4)$$

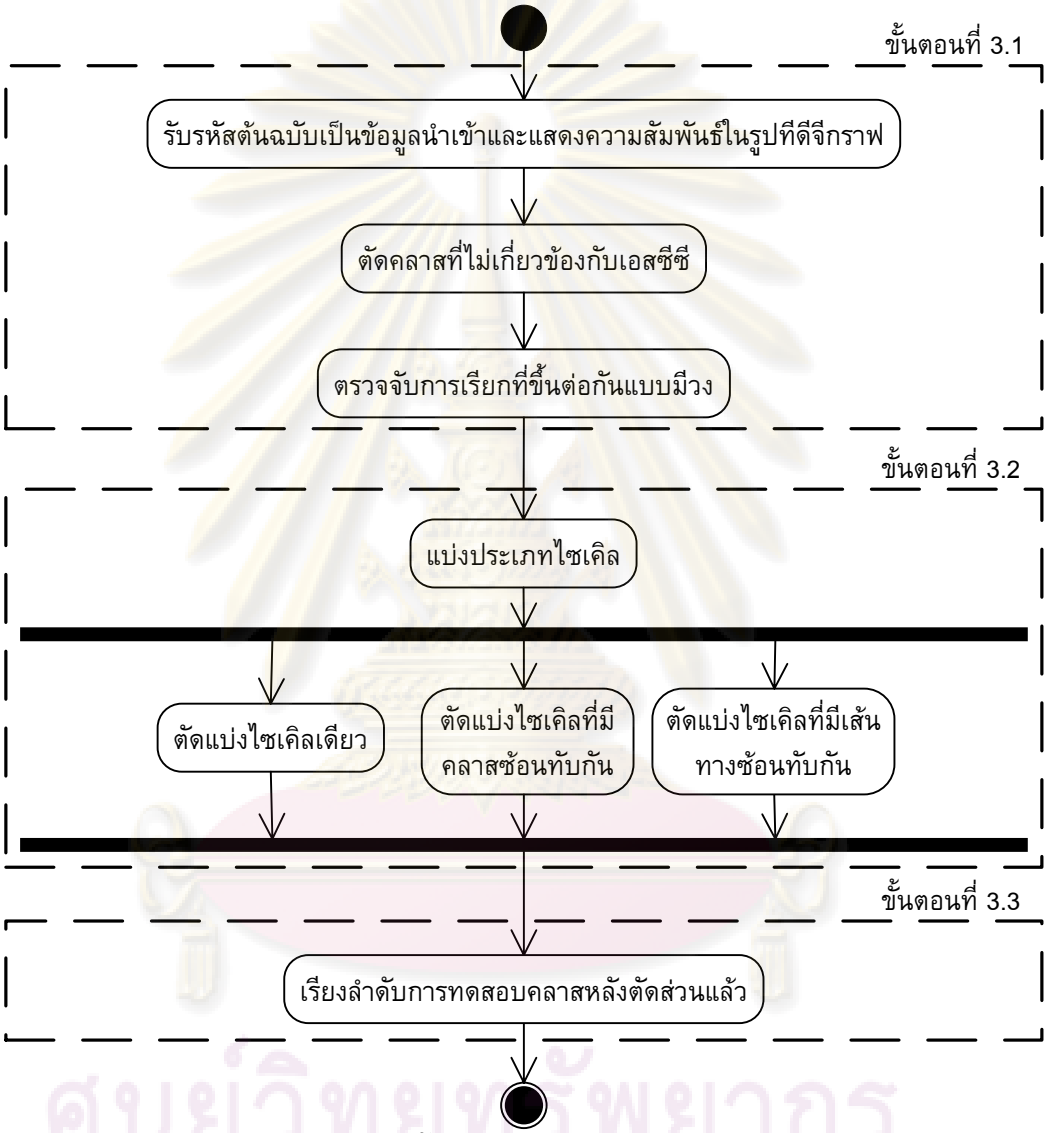
กำหนดให้ค่า W_V, W_M, W_R, W_P เป็นตัวแทนค่าถ่วงน้ำหนัก และค่าดังกล่าวมีคุณสมบัติ $W_V + W_M + W_R + W_P = 1$ สำหรับค่า $\overline{V}(i,j), \overline{M}(i,j), \overline{R}(i,j), \overline{P}(i,j)$ เกิดจากการคำนวณจากสมการที่ 3 โดยใช้ค่าในสมการที่ 2

ค่าความซับซ้อนของลำดับการทดสอบ (test order : o) เกิดจากค่าของผลรวมค่าความซับซ้อนของความสัมพันธ์ (dependencies : d) ที่ถูกลบ ดังนั้นค่าความซับซ้อนดังกล่าวจึงแสดงดังสมการที่ 5

$$OCplx(o) = \sum_{k=1}^{d} SCplx(k) \quad (5)$$

บทที่ 3
วิธีการเรียงลำดับการบูรณาการคลาสโดยใช้เทคนิคการตัดส่วนเชิงวัตถุ

งานวิจัยนี้แบ่งขั้นตอนในการดำเนินงานวิจัยออกเป็น 3 ขั้นตอน สามารถแสดงได้ด้วยแผนภาพกิจกรรม ดังรูปที่ 3.1



รูปที่ 3.1 ขั้นตอนการดำเนินการวิจัย

ในส่วนของรายละเอียดของขั้นตอนต่าง ๆ สามารถอธิบายได้ดังต่อไปนี้

3.1 การวิเคราะห์ความสัมพันธ์ระหว่างคลาส

ขั้นตอนนี้เป็นการวิเคราะห์ความสัมพันธ์ระหว่างคลาสจากรหัสต้นฉบับซึ่งเป็นข้อมูลนำเข้า ความสัมพันธ์ระหว่างคลาสสำหรับการเรียงลำดับการบูรณาการคลาสแสดงในรูปของทีดีจีกราฟ แนวทางการวิเคราะห์ความสัมพันธ์ในขั้นตอนนี้ มีขั้นตอนย่อยดังต่อไปนี้

3.1.1 การแปลงรหัสต้นฉบับเป็นทีดีจีกราฟ

ขั้นตอนนี้จะรับรหัสต้นฉบับเป็นข้อมูลนำเข้า และทำการแปลงรหัสต้นฉบับเป็นทีดีจีกราฟ ซึ่งแสดงการเรียกในระดับของคลาสและเมท็อด คลาสแสดงในรูปของโหนด ความสัมพันธ์แสดงในรูปของเส้นเชื่อมที่ชี้ไปยังโหนดที่มีความสัมพันธ์กัน

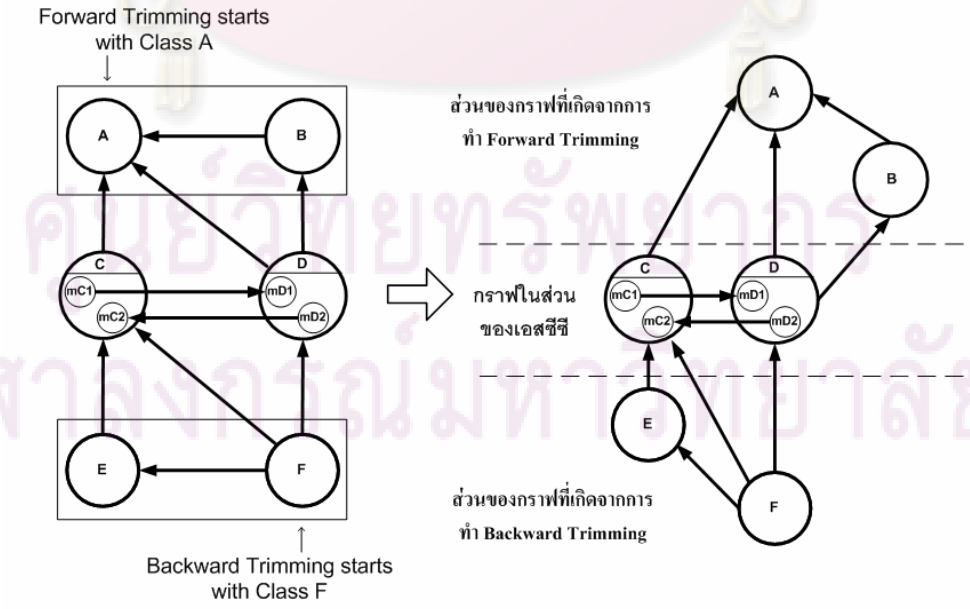
3.1.2 การตัดคลาสที่ไม่เกี่ยวข้องกับเอสซีซี

ก่อนเริ่มขั้นตอนนี้จะต้องทำการแปลงแผนภาพคลาสเป็นทีดีจีกราฟก่อน จากนั้นจึงทำการตัดคลาสที่เป็นคลาสที่ไม่เกี่ยวข้องกับเอสซีซีออก มีการใช้อัลกอริทึมดีซีเอสซี (DCSC Algorithm) [12] ซึ่งต้องอ่านความสัมพันธ์ของทุกคลาสก่อน เพื่อหาคลาสที่ไม่มีคลาสแม่ และคลาสที่ไม่มีคลาสลูก หลังจากนั้นเริ่มการตัดคลาสที่ไม่เกี่ยวข้องกับเอสซีซี 2 แบบ ดังนี้

1) เริ่มตัดคลาสที่ไม่เกี่ยวข้องกับเอสซีซีแบบไปข้างหน้า (Forward Trimming) โดยเริ่มจากคลาสที่ไม่มีคลาสแม่ เอาคลาสเหล่านี้ออกไปเรื่อยๆ จนในกลุ่มของคลาสเหลือแต่คลาสที่มีคลาสแม่ หลังจากนั้นใช้การเรียงลำดับแบบทอพอโลยีย้อนหลัง หาลำดับการทดสอบของคลาสที่ไม่เกี่ยวข้องกับเอสซีซีที่ถูกเอาออก

2) เริ่มตัดคลาสที่ไม่เกี่ยวข้องกับเอสซีซีแบบย้อนหลัง (Backward Trimming) โดยเริ่มจากคลาสที่ไม่มีคลาสลูก เอาคลาสเหล่านี้ออกไปเรื่อยๆ จนในกลุ่มของคลาสเหลือแต่คลาสที่มีคลาสลูก หลังจากนั้นใช้การเรียงลำดับแบบทอพอโลยีตามปกติ หาลำดับการทดสอบของคลาสที่ไม่เกี่ยวข้องกับเอสซีซีที่ถูกเอาออก

จุดประสงค์ของขั้นตอนนี้คือการแยกคลาสที่ไม่มีการเรียกที่ขึ้นต่อกันแบบมีวงออกก่อน และทำการเรียงลำดับขั้นต้น ถ้าคลาสทั้งหมดไม่มีการเรียกที่ขึ้นต่อกันแบบมีวงการหาลำดับการทดสอบจะสิ้นสุดที่ขั้นตอนนี้ รูปที่ 3.2 แสดงการตัดคลาสและการเรียงลำดับขั้นต้น ซึ่งกลุ่มของเอสซีซีจะอยู่ในระดับที่สอง และคลาสที่ไม่เกี่ยวข้องกับเอสซีซีคือคลาส A, B, E, F



รูปที่ 3.2 การตัดคลาสที่ไม่เกี่ยวข้องกับเอสซีซี

3.1.3 การตรวจจับการเรียกที่ขึ้นต่อกันแบบมีวง

ทาร์จันอัลกอริทึมถูกใช้หาการเรียกที่ขึ้นต่อกันแบบมีวงของกลุ่มคลาสที่เหลือหลังจากการตัดคลาสที่ไม่เกี่ยวข้องกับเอสซีซี กลุ่มของคลาสที่ได้หลังจากผ่านทาร์จันอัลกอริทึม คือ กลุ่มคลาสที่มีการเรียกที่ขึ้นต่อกันแบบมีวง ซึ่งก็คือกลุ่มเอสซีซีนั่นเอง

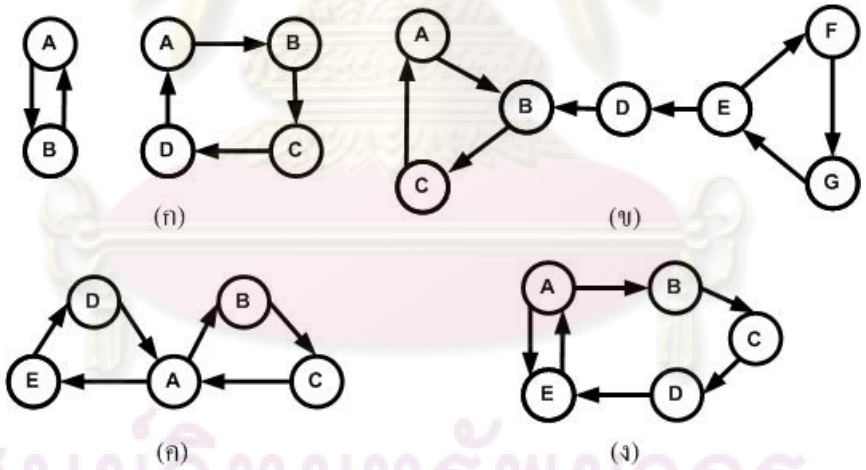
3.2 การตัดแบ่งคลาสในไซเคิลตามประเภทของไซเคิล

จุดประสงค์ของขั้นตอนนี้คือการขจัดการเรียกที่ขึ้นต่อกันแบบมีวง โดยก่อนการขจัดการเรียกที่ขึ้นต่อกันแบบมีวงจะต้องทำการแบ่งประเภทไซเคิลก่อน หลังจากนั้นจึงทำการตัดแบ่งคลาสตามประเภทของไซเคิล การตัดแบ่งคลาสสามารถทำได้โดยการหาคลาสที่สามารถตัดแบ่งออกมาได้ เมื่อพบคลาสดังกล่าวแล้วจึงทำการตัดแบ่งคลาสนั้น หลังจากนั้นจึงทำการเรียงลำดับคลาสในไซเคิล

3.2.1 การแบ่งประเภทไซเคิล

กลุ่มเอสซีซีที่ได้หลังจากทำการตรวจจับความสัมพันธ์จะถูกนำมาแบ่งประเภทไซเคิล ซึ่งมี 3 แบบ ดังต่อไปนี้

1) แบบหนึ่งไซเคิล หรือไซเคิลที่ไม่มีส่วนที่ซ้อนทับกัน ดังรูปที่ 3.3 (ก), (ข)
2) แบบไซเคิลที่มีคลาสซ้อนทับกัน ดังรูปที่ 3.3 (ค)
3) แบบไซเคิลที่มีเส้นทางซ้อนทับกัน ดังรูปที่ 3.3 (ง)



รูปที่ 3.3 ลักษณะของคลาสที่มีการเรียกที่ขึ้นต่อกันแบบมีวง

3.2.2 การตัดแบ่งไซเคิลเดียว

การตัดแบ่งไซเคิลเดียวมีขั้นตอนการทำงานดังต่อไปนี้ แสดงดังรูปที่ 3.4

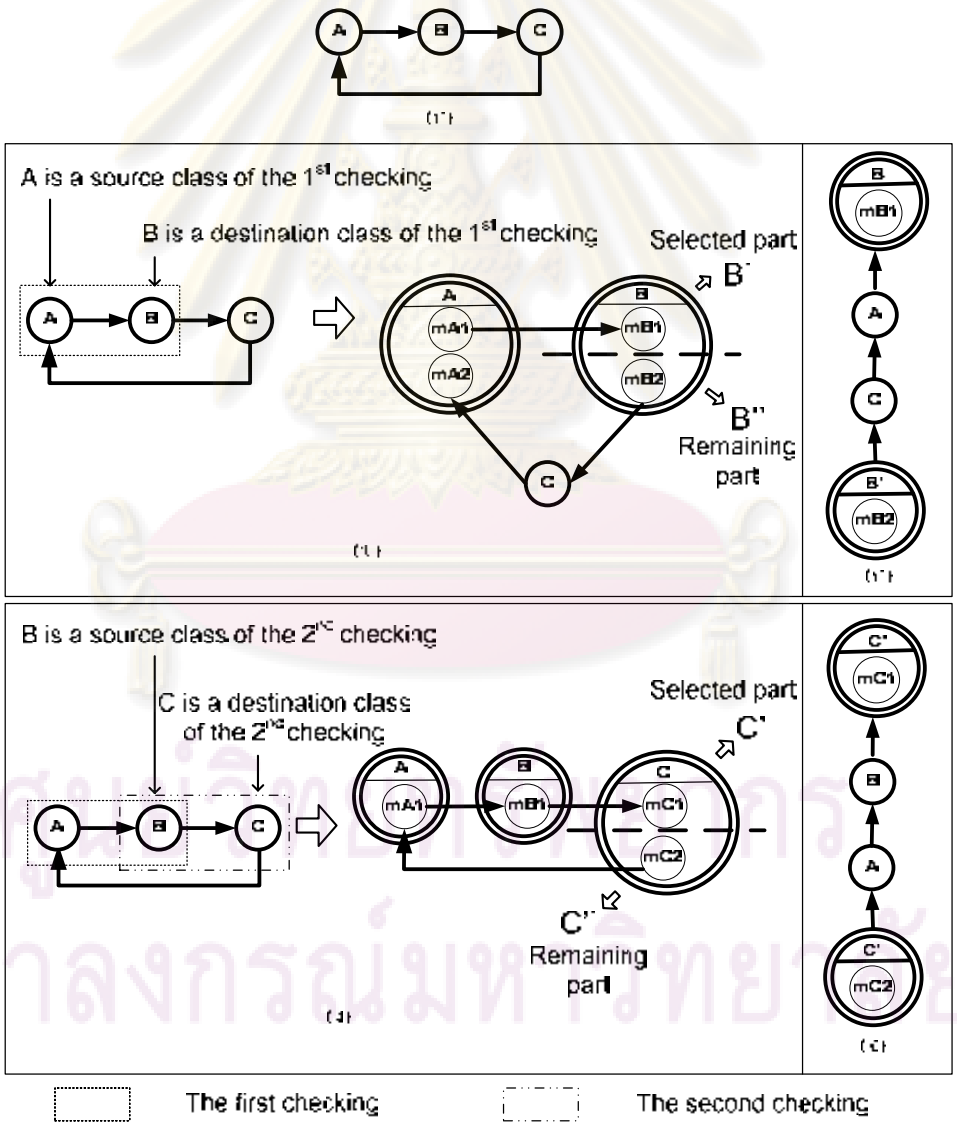
1) เลือกคลาสใดคลาสหนึ่งในไซเคิลเป็นคลาสต้นทาง และคลาสลูกของคลาสดังกล่าวเป็นคลาสปลายทาง ซึ่งตัวอย่างไซเคิลแสดงดังรูปที่ 3.4 โดยรูปที่ 3.4 (ก) คลาสต้นทาง คือ A และคลาสปลายทาง คือ B

2) วิเคราะห์ที่คลาสต้นทางเพื่อหาเมท็อดที่เรียกคลาสปลายทาง

3) ถ้าเมท็อดที่คลาสปลายทางนั้น ไม่มีการเรียกไปยังคลาสอื่นในไซเคิล เมท็อดนั้นและสมาชิกที่เป็นข้อมูลที่เกี่ยวข้องกับเมท็อดนั้นจะถูกตัดแบ่งออกมา ซึ่งทำให้การเรียกที่ขึ้นต่อกันแบบมีวงถูกขจัด ซึ่งแสดงในรูปที่ 3.4 (ข)

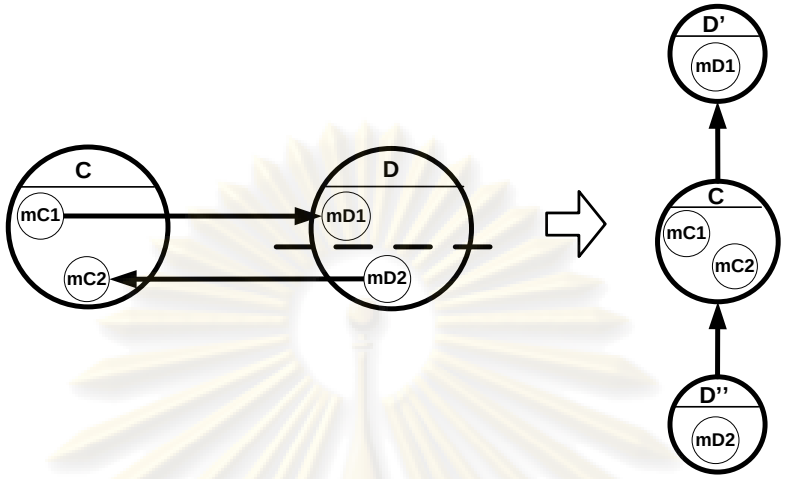
4) ถ้าคลาสปลายทางไม่สามารถแบ่งออกมาได้ คลาสลูกของโหนดนี้จะถูกพิจารณาเป็นคลาสปลายทางแทน และคลาสที่ตอนแรกเป็นคลาสปลายทางจะถูกเปลี่ยนเป็นคลาสต้นทาง และทำซ้ำที่ขั้นตอนที่สาม จนกระทั่งหาคลาสที่ตัดแบ่งได้ ดังรูปที่ 3.4 (ง) กรณีที่คลาสปลายทางที่กำหนดใหม่กลับไปซ้ำกับคลาสปลายทางที่ได้กำหนดไว้ในตอนแรก แสดงว่าไซเคิลนี้ไม่สามารถตัดแบ่งได้

5) เรียงลำดับคลาสโดยใช้การเรียงลำดับแบบทอพอโลยีย้อนหลัง จากรูปที่ 3.4 (ข) เมื่อเรียงลำดับคลาสแล้วลำดับคลาสแสดงดังรูปที่ 3.4 (ค) แต่ถ้าการตัดคลาสเป็นดังรูปที่ 3.4 (ง) ลำดับคลาสหลังการเรียงลำดับแสดงดังรูปที่ 3.4 จ)



รูปที่ 3.4 การตัดแบ่งไซเคิล

ตัวอย่างจากรูปที่ 3.2 กลุ่มเอสซีซีหลังการตรวจจับเป็นประเภทไซเคิลเดียว ดังนั้นการตัดแบ่งไซเคิลเดียวจากตัวอย่างนี้ แสดงดังรูปที่ 3.5



รูปที่ 3.5 การตัดไซเคิลจากตัวอย่างของรูปที่ 3.2

3.2.3 การตัดแบ่งไซเคิลที่มีคลาสซ้อนทับกัน

การตัดแบ่งไซเคิลที่มีคลาสซ้อนทับกันมีขั้นตอนการทำงานดังต่อไปนี้

1) เลือกไซเคิลหนึ่งขึ้นมาพิจารณาก่อน จากรูปที่ 3.3 (ค) เลือกไซเคิลที่มีคลาส D, A, E ขึ้นมาพิจารณาก่อน

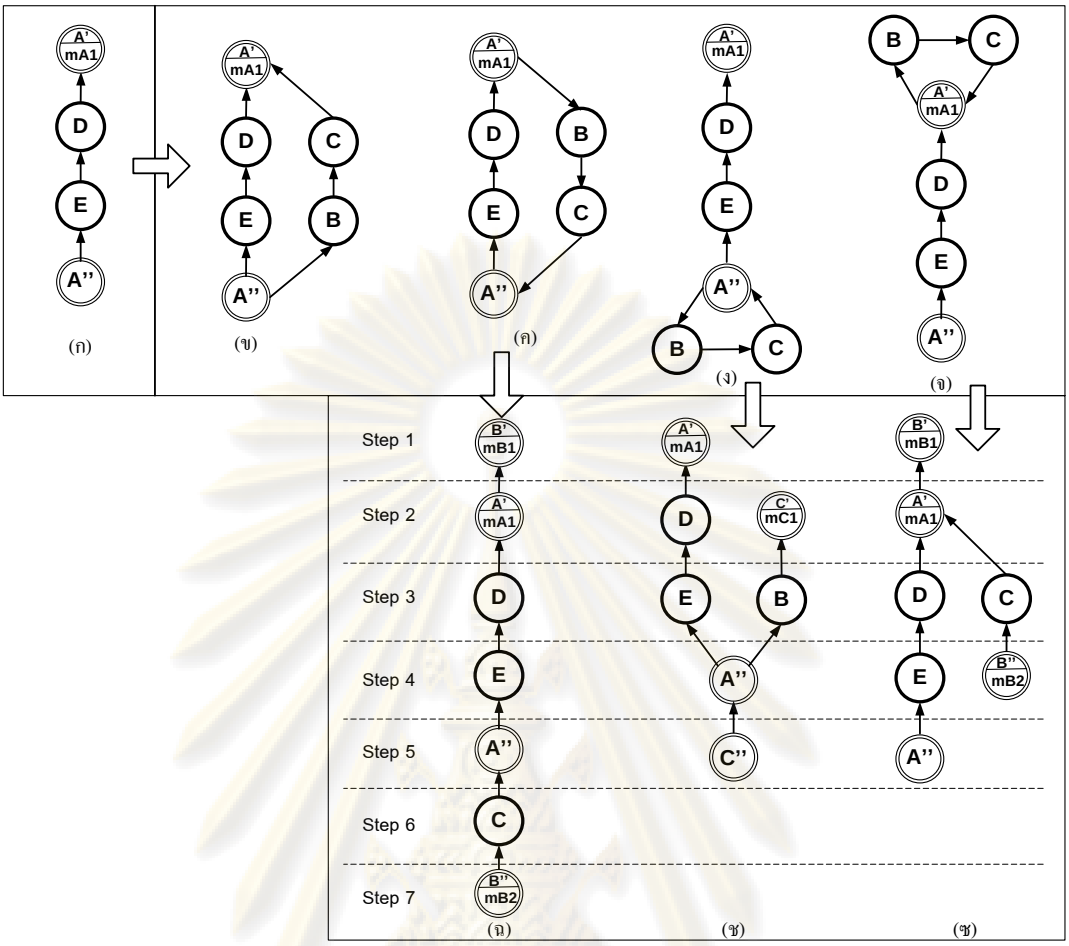
2) กำหนดคลาสที่การซ้อนทับเป็นคลาสปลายทาง และคลาสแม่ของคลาสนั้นเป็นคลาสต้นทาง จากรูปที่ 3.3 (ค) คลาสต้นทางคือ D และคลาสปลายทางคือ A

3) ทำงานเหมือนขั้นตอนที่ 2 ของการตัดแบ่งไซเคิลเดียว เพื่อขจัดการเรียกที่ขึ้นต่อกันแบบมีวงของไซเคิลนี้ ซึ่งลักษณะของการขจัดการเรียกที่ขึ้นต่อกันแบบมีวงจะแบ่งออกเป็น 2 แบบ โดยแบบแรกคือการตัดที่คลาสที่มีการซ้อนทับ แบบที่สองคือตัดที่คลาสอื่นที่ไม่ใช่คลาสที่มีการซ้อนทับ

4) นำอีกไซเคิลมาเชื่อมติดกับผลลัพธ์ที่ได้จากการขจัดไซเคิลแรก ซึ่งการเชื่อมอีกไซเคิลกับผลลัพธ์ 2 แบบที่ได้จากขั้นตอนก่อนหน้านี้ แสดงดังต่อไปนี้

4.1) กรณีที่ตัดแบ่งคลาสที่มีการซ้อนทับ

จากรูปที่ 3.3 (ค) เมื่อตัดคลาสที่มีการซ้อนทับกันแล้ว ผลของการเรียงลำดับไซเคิลแรกจะเป็นตามรูปที่ 3.6 (ก) จากนั้น ทำการเชื่อมอีกไซเคิลผลของการเชื่อมจะเป็นแบบใดแบบหนึ่งใน 4 แบบดังต่อไปนี้



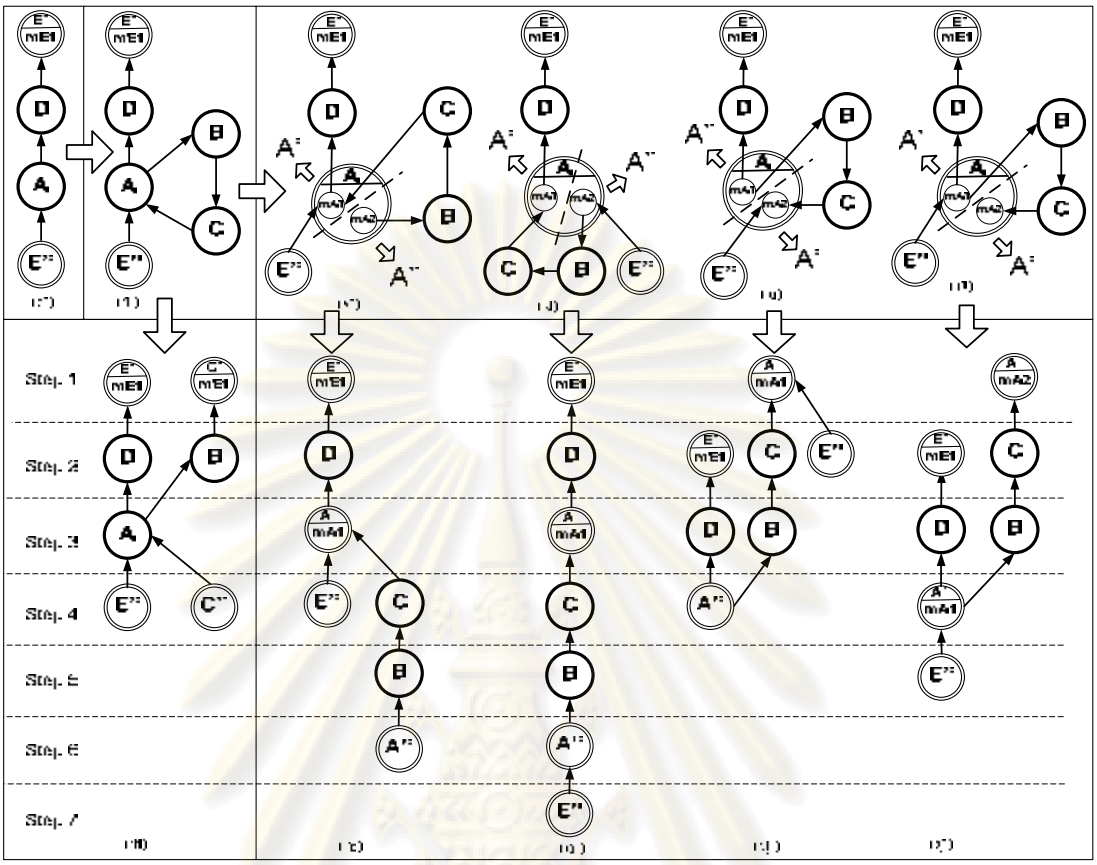
รูปที่ 3.6 การตัดแบ่งคลาสที่มีการซ้อนทับกัน

- แบบแรก ชิ้นส่วนคลาสที่เหลือมีการเรียกไปยังคลาสอื่นในไซเคิล และคลาสอื่นในไซเคิลเรียกไปที่ชิ้นส่วนคลาสที่ถูกแยกออกมา ตามรูปที่ 3.6 (ข)
- แบบที่สอง ชิ้นส่วนคลาสที่ถูกแยกออกมามีการเรียกไปยังคลาสอื่นในไซเคิล และคลาสอื่นในไซเคิลเรียกไปที่ชิ้นส่วนคลาสที่เหลือ ตามรูปที่ 3.6 (ค)
- แบบที่สาม ชิ้นส่วนคลาสที่เหลือมีการเรียกไปยังคลาสอื่นในไซเคิล และคลาสในไซเคิลเรียกกลับไปยังชิ้นส่วนเดิม ตามรูปที่ 3.6 (ง)
- แบบสุดท้าย ชิ้นส่วนคลาสที่ถูกแยกออกมามีการเรียกไปยังคลาสอื่นในไซเคิล และคลาสในไซเคิลเรียกกลับไปยังชิ้นส่วนเดิม ตามรูปที่ 3.6 (จ)

หลังจากเชื่อมอีกไซเคิลแล้ว แบบแรกเท่านั้นที่ไม่มีการเรียกที่ขึ้นต่อกันแบบมีวงเหลืออยู่ ทำให้สามารถข้ามขั้นตอนที่ 5 ไปขั้นตอนที่ 6 เรียงลำดับการทดสอบได้ทันที สำหรับสามแบบที่เหลือ เมื่อเชื่อมอีกไซเคิลแล้วยังมีการเรียกที่ขึ้นต่อกันแบบมีวงเหลืออยู่

4.2) กรณีที่ตัดแบ่งคลาสอื่นที่ไม่ใช่คลาสที่มีการซ้อนทับ

จากรูปที่ 3.3 (ค) เมื่อตัดแบ่งคลาสอื่นที่ไม่ใช่คลาสที่มีการซ้อนทับแล้ว ผลของการเรียงลำดับไซเคิลแรกจะเป็นตามรูปที่ 3.7 (ก) เมื่อเชื่อมอีกไซเคิลกับผลลัพธ์ที่ได้จากการตัดแบ่งกรณีนี้ การเชื่อมต่อจะเป็นตามรูปที่ 3.7 (ข) จากรูปดังกล่าว เมื่อเชื่อมต่ออีกไซเคิลแล้วยังปรากฏการเรียกที่ขึ้นต่อกันแบบมีวงอยู่



รูปที่ 3.7 การตัดแบ่งคลาสอื่นที่ไม่ใช่คลาสที่มีการซ้อนทับ

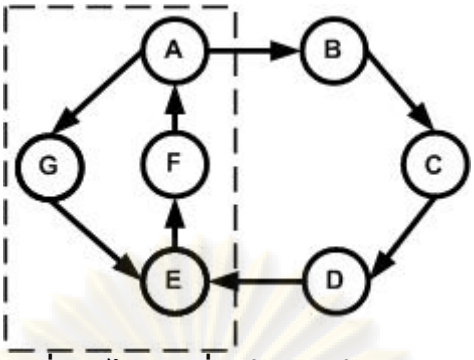
5) กลับไปที่ขั้นตอนที่สาม และทำซ้ำจนกระทั่งไม่มีการเรียกที่ขึ้นต่อกันแบบมีวงเหลืออยู่ จากรูปที่ 3.7 (ข) เมื่อเกิดการตัดแบ่งคลาสแล้ว ถ้าการตัดแบ่งนั้น ทำการตัดคลาสที่มีการซ้อนทับ (คลาส A) การตัดคลาสนั้นสามารถเป็นไปได้ใน 4 ลักษณะ ตามรูปที่ 3.7 (ค), (ง), (จ), (ฉ) ในขณะที่ รูปที่ 3.7 (ช) แสดงการตัดคลาสที่ไม่ใช่คลาสที่มีการซ้อนทับ

6) เรียงลำดับคลาสแบบทอพอโลยี กรณีที่ตัดแบ่งคลาสที่มีการซ้อนทับ จากรูปที่ 3.6 (ค), (ง), (จ) มีผลลัพธ์จากการเรียงลำดับการทดสอบตามรูปที่ 3.6 (ฉ), (ช), (ซ) ในขณะที่กรณีที่ตัดแบ่งคลาสอื่นที่ไม่ใช่คลาสที่มีการซ้อนทับ จากรูปที่ 3.7 (ค), (ง), (จ), (ฉ) มีผลลัพธ์จากการเรียงลำดับการทดสอบแสดงดังรูปที่ 3.7 (ซ), (ฌ), (ญ), (ฎ)

3.2.4 การตัดแบ่งไซเคิลที่มีเส้นทางซ้อนทับกัน

การตัดแบ่งไซเคิลที่มีเส้นทางซ้อนทับกันมีขั้นตอนการทำงานดังต่อไปนี้

1) เลือกไซเคิลที่เล็กที่สุดจากแผนภาพขึ้นมาพิจารณาก่อน จากรูปที่ 3.8 เลือกคลาส A, G, E, F พิจารณาก่อน



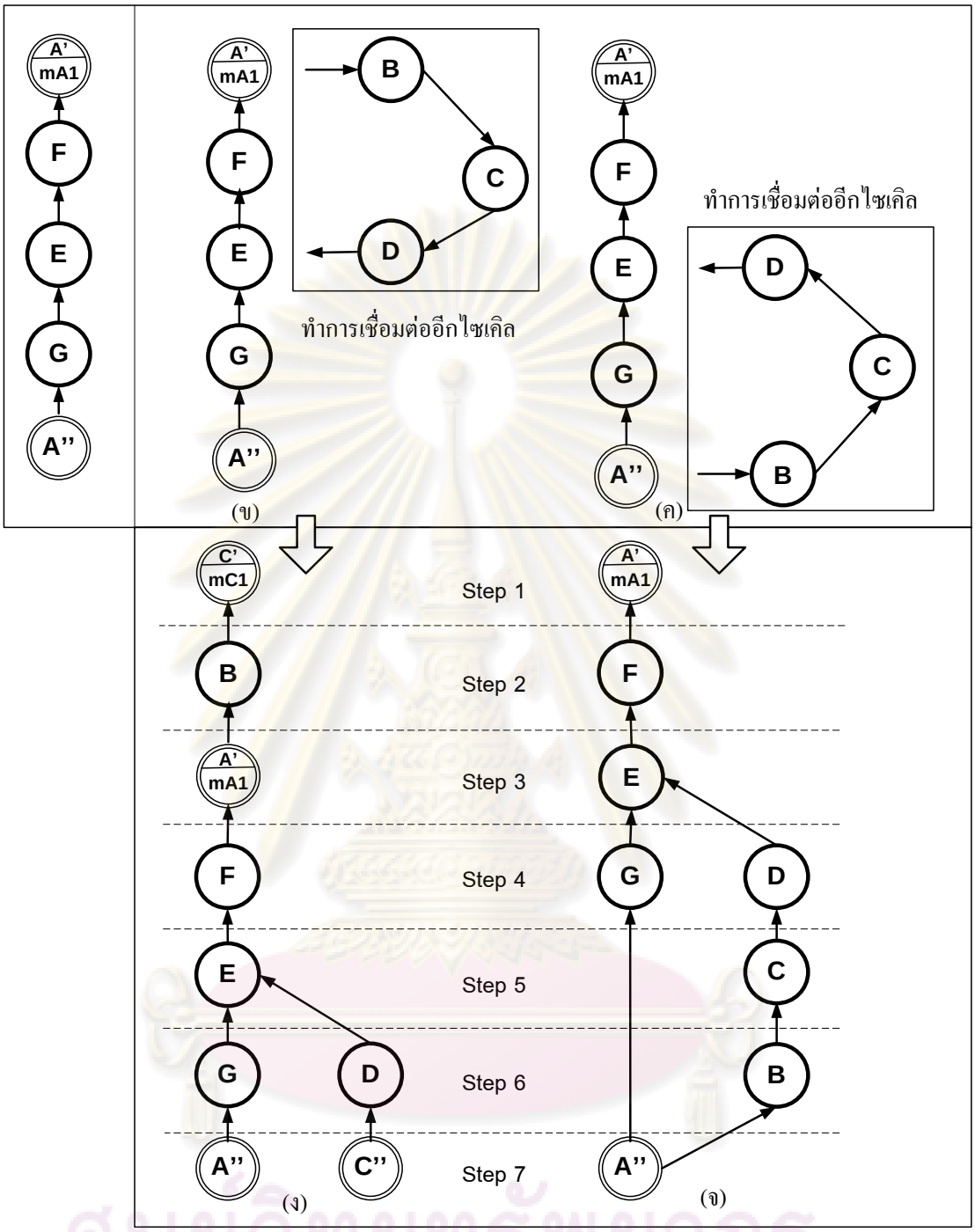
รูปที่ 3.8 ไซเคิลที่มีเส้นทางซ้อนทับกัน

2) ทำงานเหมือนขั้นตอนที่ 2 ของการตัดแบ่งไซเคิลเดียว จนกระทั่งขจัดการเรียกที่ขึ้นต่อกันแบบมีวงของไซเคิลนี้ได้ ซึ่งจะมี 3 ลักษณะที่เป็นไปได้ของโหนดที่ถูกตัดแบ่งดังต่อไปนี้

- โหนดเชื่อม (Joint Node) คือโหนดที่มีค่าของอินดีกรี หรือ เอาท์ดีกรี อย่างใดอย่างหนึ่ง มีค่ามากกว่าหนึ่ง จากรูปที่ 3.8 คือ โหนด A กับ E
- โหนดที่อยู่บนเส้นทางที่ซ้อนทับกัน (Overlap Path Node) คือโหนดที่อยู่บนเส้นทางที่มีการซ้อนทับกัน โดยที่ไม่ใช่โหนดเชื่อม จากรูปที่ 3.8 คือ โหนด F
- โหนดที่อยู่ในไซเคิลเดียว (Single Cycle Node) คือโหนดที่อยู่ในไซเคิลเดียวเท่านั้น จากรูปที่ 3.8 คือ โหนด G

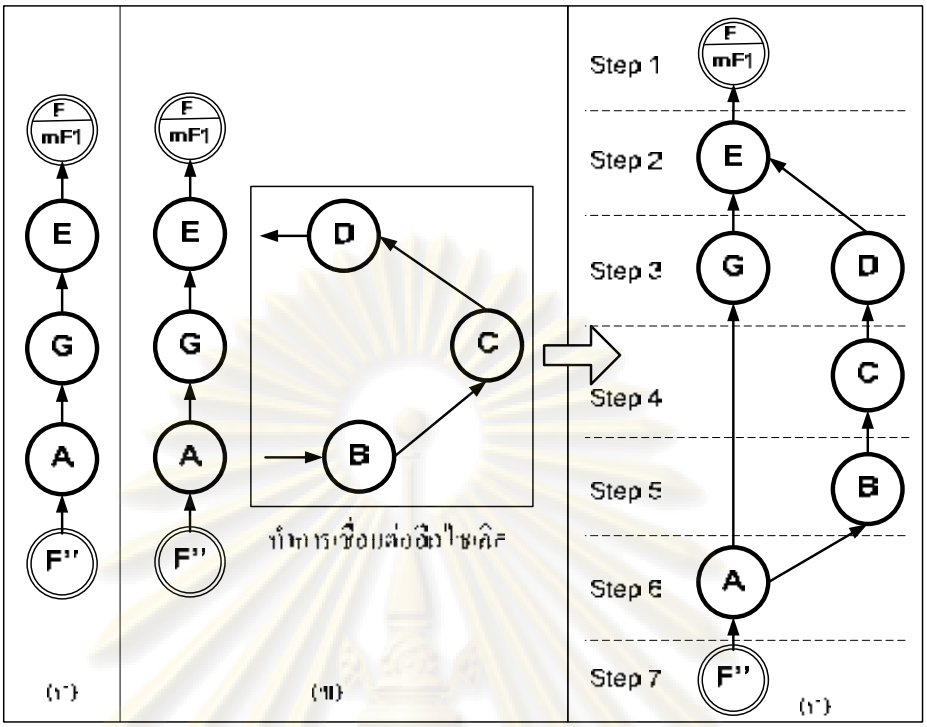
3) เชื่อมอีกไซเคิลกับผลลัพธ์ที่ได้จากการขจัดการเรียกที่ขึ้นต่อกันแบบมีวงของไซเคิลแรก เมื่อเชื่อมแล้วผลลัพธ์จะแตกต่างกันขึ้นอยู่กับโหนดที่ถูกตัดแบ่งดังนี้

3.1) ถ้าโหนดที่ถูกตัดแบ่งเป็นโหนดเชื่อมแล้ว ผลลัพธ์จากการเชื่อมอีกไซเคิลอาจจะยังมีการเรียกที่ขึ้นต่อกันแบบมีวงอยู่ หรือไม่มีก็ได้ จากรูปที่ 3.8 เมื่อตัดคลาสที่โหนดเชื่อม ซึ่งก็คือ โหนด A จะมีผลจากตัดคลาสตามรูปที่ 3.9 (ก) ผลลัพธ์จากการเชื่อมอีกไซเคิลแล้วยังมีการเรียกที่ขึ้นต่อกันแบบมีวงอยู่ แสดงดังรูปที่ 3.9 (ข) ส่วนผลลัพธ์จากการเชื่อมอีกไซเคิลแล้วไม่มีการเรียกที่ขึ้นต่อกันแบบมีวง แสดงดังรูปที่ 3.9 (ค)



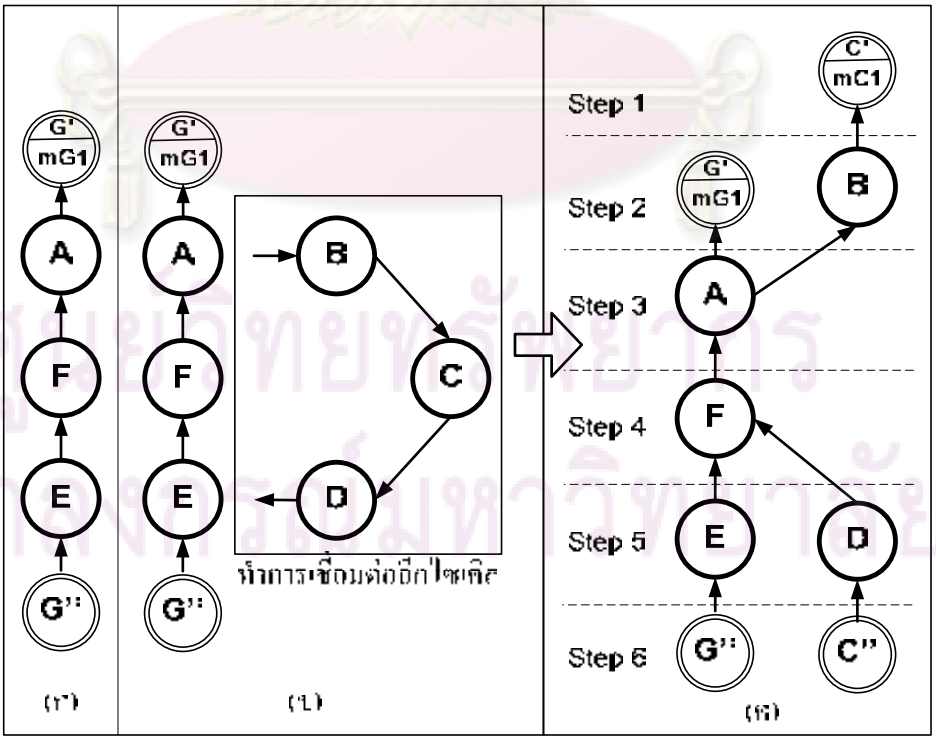
รูปที่ 3.9 การตัดคลาสที่โหนดเชื่อม และการเชื่อมต่ออีกไซเคิล

3.2) ถ้าโหนดที่ถูกตัดแบ่งเป็นโหนดที่อยู่บนเส้นทางที่ซ้อนทับกันแล้ว ผลลัพธ์จากการเชื่อมอีกไซเคิลจะไม่มีการเรียกที่ขึ้นต่อกันแบบมีวงเหลืออยู่ จากรูปที่ 3.8 เมื่อตัดคลาสที่โหนดที่อยู่บนเส้นทางที่ซ้อนทับกัน ซึ่งก็คือ โหนด F จะมีผลจากตัดคลาสตามรูปที่ 3.10 (ก) เมื่อเชื่อมอีกไซเคิลแล้ว ผลลัพธ์จากการเชื่อมแสดงดังรูปที่ 3.10 (ข) จากรูปจะเห็นได้ว่าไม่ปรากฏการเรียกที่ขึ้นต่อกันแบบมีวง



รูปที่ 3.10 การตัดคลาสที่โหนดที่อยู่บนเส้นทางที่ซ้อนทับกัน และการเชื่อมต่ออีกไซเคิล

3.3) ถ้าโหนดที่ถูกตัดแบ่งเป็นโหนดที่อยู่ในไซเคิลเดียวแล้ว ผลลัพธ์จากการเชื่อมอีกไซเคิลจะยังมีการเรียกที่ขึ้นต่อกันแบบมีวงอยู่ จากรูปที่ 3.8 เมื่อตัดคลาสที่โหนดที่อยู่ในไซเคิลเดียว ซึ่งก็คือ โหนด G จะมีผลจากตัดคลาสตามรูปที่ 3.11 (ก) เมื่อเชื่อมอีกไซเคิลแล้ว ผลลัพธ์จากการเชื่อมแสดงดังรูปที่ 3.11 (ข) จากรูปจะเห็นได้ว่ายังปรากฏการเรียกที่ขึ้นต่อกันแบบมีวงอยู่



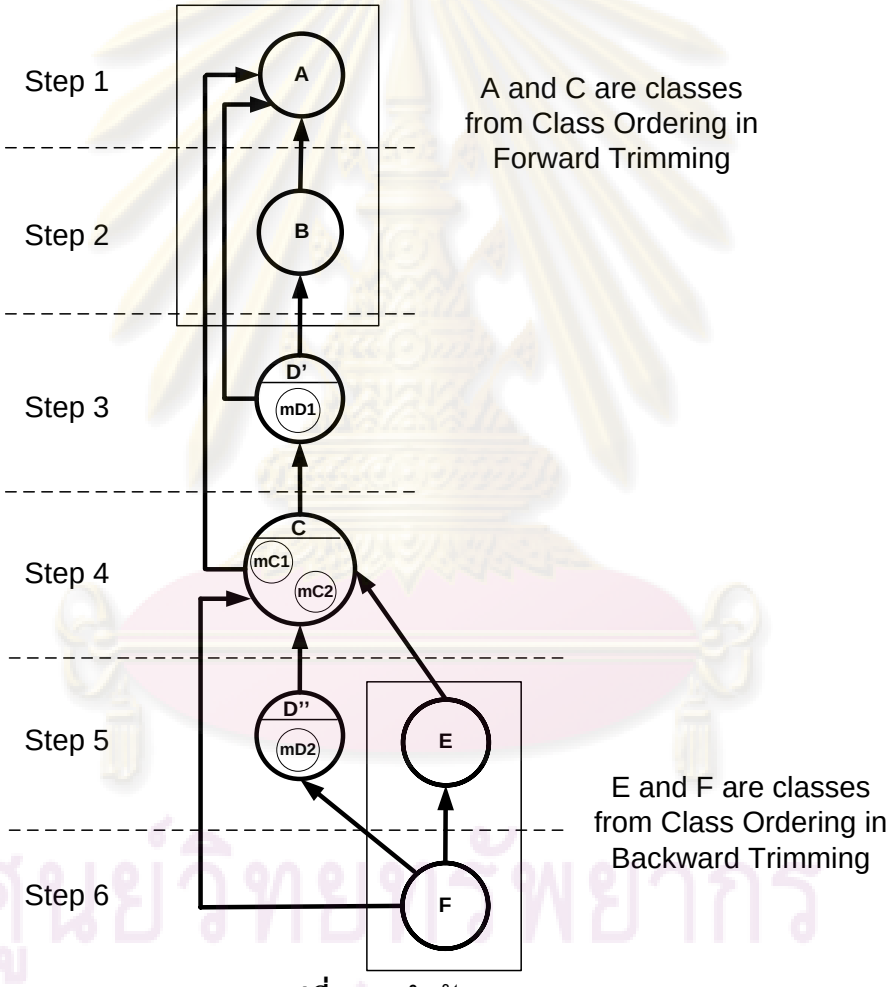
รูปที่ 3.11 การตัดคลาสที่โหนดที่อยู่ในไซเคิลเดียว และการเชื่อมต่ออีกไซเคิล

4) กลับไปที่ขั้นตอนที่หนึ่ง และทำซ้ำจนกระทั่งไม่มีการเรียกที่ขึ้นต่อกันแบบมีวงเหลืออยู่

5) เรียงลำดับคลาสแบบทอพอโลยี โดยผลลัพธ์ที่ได้จากการเรียงลำดับของการตัดคลาสที่โหนดเชื่อมแสดงดังรูปที่ 3.9 (ง), (จ) ส่วนผลลัพธ์จากการเรียงลำดับของการตัดคลาสที่โหนดที่อยู่บนเส้นทางที่ซ้อนทับกันแสดงดังรูปที่ 3.10 (ค) และผลลัพธ์จากการเรียงลำดับของการตัดคลาสที่โหนดที่อยู่ในไซเคิลเดียว แสดงดังรูปที่ 3.11 (ค)

3.3 การเรียงลำดับการทดสอบ

หลังจากการตัดส่วนคลาสแล้ว เมื่อเชื่อมคลาสที่ไม่เกี่ยวข้องกับเอสซีซีที่ตัดออกในขั้นตอนที่ 3.2 ลำดับของคลาสจะแสดงดังรูปที่ 3.12



รูปที่ 3.12 ลำดับการทดสอบ

จากรูปที่ 3.12 ลำดับคลาสสำหรับทดสอบ มีดังนี้

1. คลาส A: การทดสอบในระดับนี้เป็นการทดสอบหน่วย
2. คลาส B: ทำการทดสอบการเชื่อมต่อกับคลาส A
3. คลาส D’ (คลาสที่ทำการตัดส่วนออกมา): ทำการทดสอบการเชื่อมต่อกับคลาส A และ คลาส B

4. คลาส C: ทำการทดสอบการเชื่อมต่อกับคลาส A และ คลาส D’
5. มี 2 คลาส คือ
   - คลาส D’’ (คลาสที่เหลือจากการตัดส่วน): ทำการทดสอบการเชื่อมต่อกับคลาส C
   - คลาส E: ทำการทดสอบการเชื่อมต่อกับคลาส C
6. คลาส F: ทำการทดสอบการเชื่อมต่อกับคลาส C คลาส D’’ คลาส E

บทที่ 4
การพัฒนาเครื่องมือ

ในบทนี้จะกล่าวถึงรายละเอียดของการพัฒนาเครื่องมือสำหรับการเรียงลำดับการบูรณาการคลาสโดยใช้เทคนิคการตัดส่วนเชิงวัตถุ โดยจะกล่าวถึงสภาพแวดล้อมที่ใช้ในการพัฒนาเครื่องมือ สถาปัตยกรรมของเครื่องมือ และโครงสร้างของเครื่องมือ ซึ่งมีรายละเอียดดังต่อไปนี้

4.1 สภาพแวดล้อมที่ใช้ในการพัฒนาเครื่องมือ

ฮาร์ดแวร์ (Hardware) และซอฟต์แวร์ (Software) ที่ใช้ในการพัฒนาระบบมีดังนี้

4.1.1 ฮาร์ดแวร์

1) เครื่องคอมพิวเตอร์ส่วนบุคคล หน่วยประมวลผล อินเทล เพนเทียม4 3.4 กิกะเฮิร์ท โปรเซสเซอร์ 630 (Intel Pentium M 3.4 GHz Processor 630)

2) หน่วยความจำ (Memory) 2 กิกะไบต์

3) จานบันทึกแบบแข็ง (Hard disk) ความจุ 120 กิกะไบต์

4) จอภาพ 19 นิ้ว

4.1.2 ซอฟต์แวร์

1) ระบบปฏิบัติการ

(1) ระบบปฏิบัติการไมโครซอฟท์วินโดวส์ เอ็กซ์พี โปรเฟสชันแนล (Microsoft Window XP Professional)

2) ซอฟต์แวร์ที่ใช้ในจัดทำเอกสาร

(1) ไมโครซอฟท์ออฟฟิส 2003 (Microsoft Office 2003)

(2) ไมโครซอฟท์ออฟฟิสวิซิโอ โปรเฟสชันแนล 2003 (Microsoft Office Visio Professional 2003)

3) ซอฟต์แวร์ที่ใช้ในการเขียนโปรแกรมเพื่อพัฒนาเครื่องมือ

(1) ไมโครซอฟท์ วิชวล สตูดิโอ 2005 (Microsoft Visual Studio 2005)

4.2 การออกแบบสถาปัตยกรรมของเครื่องมือ

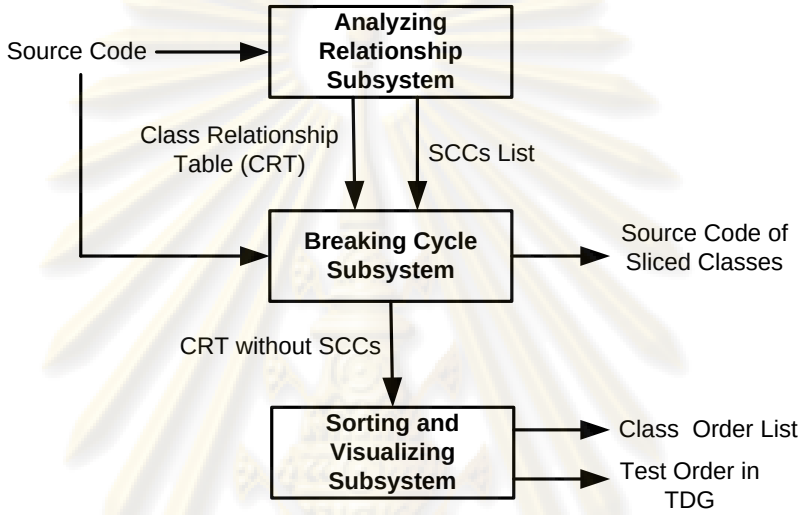
ระบบที่พัฒนามีการออกแบบสถาปัตยกรรม ดังรูปที่ 4.1 ซึ่งสามารถแบ่งออกเป็น 3 ส่วน ได้แก่

1) ส่วนการวิเคราะห์ความสัมพันธ์ระหว่างคลาส (Analyzing Relationship Subsystem) เป็นส่วนที่ทำหน้าที่ตรวจหาความสัมพันธ์ระหว่างคลาสเพื่อสร้างตารางความสัมพันธ์ระหว่างคลาส หรือ ซีอาร์ที (Class Relationship Table or CRT) และตรวจจับการเรียกที่ขึ้นต่อกันแบบมีวงเพื่อสร้างรายการของกลุ่มเอสซีซี (SCCs List)

2) ส่วนการขจัดการเรียกที่ขึ้นต่อกันแบบมีวง (Breaking Cycle Subsystem) เป็นส่วนกลางที่ทำหน้าที่วิเคราะห์ซีอาร์ที รายการของกลุ่มเอสซีซี และรหัสต้นฉบับ เพื่อขจัดการ

เรียกที่ขึ้นต่อกันแบบมีวง หลังจากขจัดการเรียกที่ขึ้นต่อกันแบบมีวงแล้ว เอาต์พุตที่ได้จากส่วนนี้คือ รหัสต้นฉบับของคลาสที่ได้ทำการตัดส่วนออกมา และตารางความสัมพันธ์ระหว่างคลาสที่ปราศจากเอสซีซี ซึ่งเป็นตารางความสัมพันธ์ระหว่างคลาสที่ไม่มีการเรียกที่ขึ้นต่อกันแบบมีวงแล้วนั่นเอง

3) ส่วนการเรียงลำดับคลาสและการสร้างมโนภาพ (Sorting and Visualizing Subsystem) เป็นส่วนที่ทำหน้าที่เรียงลำดับคลาสหลังการขจัดการเรียกที่ขึ้นต่อกันแบบมีวงและทำการสร้างมโนภาพของความสัมพันธ์ระหว่างคลาส



รูปที่ 4.1 สถาปัตยกรรมของเครื่องมือสำหรับการเรียงลำดับการบูรณาการคลาสโดยใช้เทคนิคการตัดส่วนเชิงวัตถุ

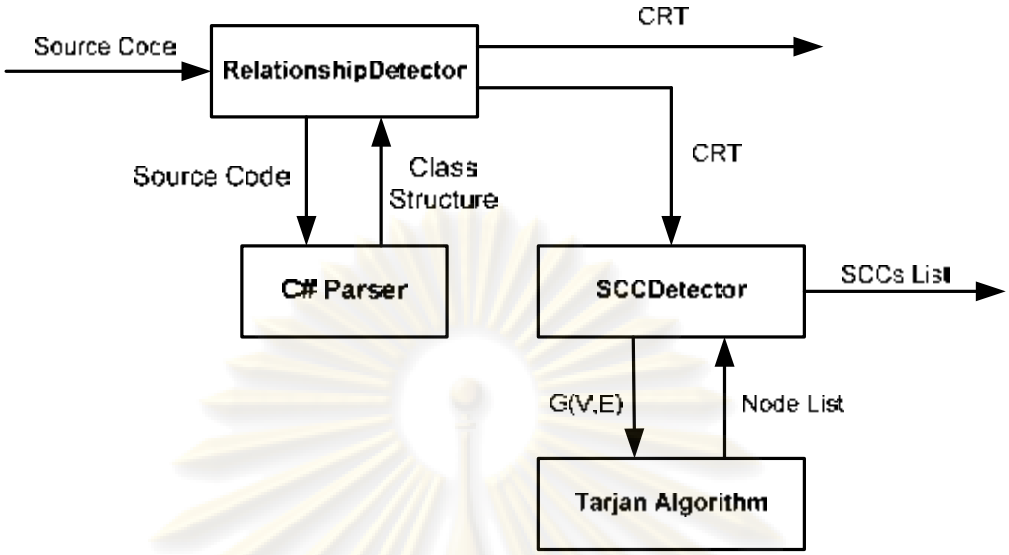
รายละเอียดของเครื่องมือสำหรับการเรียงลำดับการบูรณาการคลาสโดยใช้เทคนิคการตัดส่วนเชิงวัตถุ อธิบายในหัวข้อย่อยดังต่อไปนี้

4.2.1 ส่วนการวิเคราะห์ความสัมพันธ์ระหว่างคลาส

ส่วนการวิเคราะห์ความสัมพันธ์ระหว่างคลาสทำหน้าที่วิเคราะห์ความสัมพันธ์ระหว่างคลาสจากรหัสต้นฉบับ เพื่อแสดงความสัมพันธ์ในรูปของซีอาร์ที หลังจากนั้น จึงทำการตรวจสอบหาการเรียกที่ขึ้นต่อกันแบบมีวงจากซีอาร์ที ส่วนประกอบต่างๆของส่วนการวิเคราะห์ความสัมพันธ์ระหว่างคลาส แสดงในรูปที่ 4.2 ซึ่งแต่ละส่วนประกอบ มีรายละเอียดดังนี้

1) ตัวตรวจหาความสัมพันธ์ (Relationship Detector)

ตัวตรวจหาความสัมพันธ์ทำการรับรหัสต้นฉบับของทุกๆ คลาสภายใต้การทดสอบ (Classes Under Test or CUT) เข้ามาและทำการเรียกตัวแจงส่วนภาษาซีชาร์ป [13] เพื่อทำการแจงส่วนรหัสต้นฉบับให้ออกมาในรูปโครงสร้างคลาส จากนั้นจึงทำการตรวจหาความสัมพันธ์ระหว่างคลาสจากโครงสร้างดังกล่าว เมื่อทำการตรวจหาเสร็จสิ้น ความสัมพันธ์ระหว่างคลาสจะถูกแสดงในรูปของซีอาร์ที ซึ่งซีอาร์ทีเป็นส่วนสำคัญในการตรวจหาการเรียกที่ขึ้นต่อกันแบบมีวง และในส่วนประกอบของการขจัดการเรียกที่ขึ้นต่อกันแบบมีวง



รูปที่ 4.2 ส่วนการวิเคราะห์ความสัมพันธ์ระหว่างคลาส

2) ตัวแจงส่วนภาษาซีชาร์ป (C# Parser)

ตัวแจงส่วนภาษาซีชาร์ปมีหน้าที่แจงส่วนรหัสต้นฉบับภาษาซีชาร์ปให้อยู่ในรูปแบบของโครงสร้างคลาส ซึ่งการแจงส่วนทำตามมาตรฐานอีซีเอ็มเอ 334 ซึ่งเป็นข้อกำหนดไวยากรณ์ภาษาซีชาร์ป (ECMA-334 C# Grammar) [14]

3) ตัวตรวจหาเอสซีซี (SCC Detector)

ตัวตรวจหาเอสซีซีทำหน้าที่รับซีอาร์ทีซึ่งเป็นข้อมูลนำเข้าจากตัวตรวจหาความสัมพันธ์ และแปลงซีอาร์ทีนั้นให้อยู่ในรูปของโหนดและเส้นเชื่อมตาม G(V, E) [5] จากนั้นทำการตัดโหนดที่ไม่เกี่ยวข้องกับเอสซีซี ดังที่ได้อธิบายในหัวข้อ 3.2 และเรียกทาร์จันอัลกอริทึมเพื่อตรวจหาเอสซีซี จากนั้นจึงรับรายการโหนด (Node List) ซึ่งเป็นข้อมูลส่งออกจากทาร์จันอัลกอริทึม และแปลงรายการโหนดเป็นรายการของกลุ่มเอสซีซี

4) ทาร์จันอัลกอริทึม

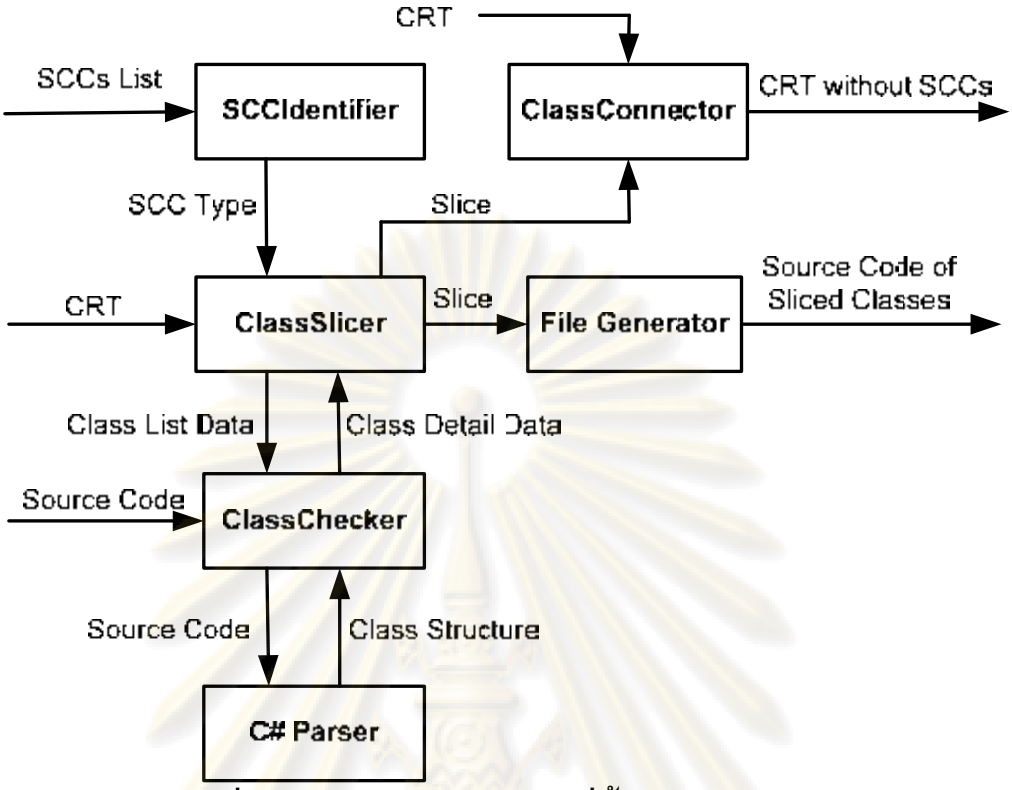
ทาร์จันอัลกอริทึมใช้ในการหาเอสซีซี รายละเอียดอยู่ในบทที่ 2 หัวข้อ 2.2.1

4.2.2 ส่วนการขจัดการเรียกที่ขึ้นต่อกันแบบมีวง

จุดประสงค์ของส่วนการขจัดการเรียกที่ขึ้นต่อกันแบบมีวงคือ การตัดส่วนที่สามารถแยกออกมาได้ในคลาส เพื่อขจัดการเรียกที่ขึ้นต่อกันแบบมีวง ซึ่งกระบวนการในส่วนนี้จะดำเนินการต่อไปจนกระทั่งการเรียกที่ขึ้นต่อกันแบบมีวงถูกทำลายจนหมด หลังจากกระบวนการในส่วนนี้เสร็จสิ้นตารางความสัมพันธ์ที่ปราศจากเอสซีซี และรหัสต้นฉบับของคลาสที่ถูกตัดส่วนออกมาจะถูกสร้างขึ้นเป็นข้อมูลนำออก ส่วนประกอบต่างๆของส่วนการขจัดการเรียกที่ขึ้นต่อกันแบบมีวง แสดงในรูปที่ 4.3 ซึ่งแต่ละส่วนประกอบ มีรายละเอียดดังนี้

1) ตัวระบุเอสซีซี (SCC Identifier)

ตัวระบุเอสซีซีทำหน้าที่กำหนดประเภทของเอสซีซี โดยมีรายการเอสซีซีเป็นข้อมูลนำเข้า ซึ่งประเภทของเอสซีซีมี 3 แบบคือ แบบหนึ่งไซเคิล แบบไซเคิลที่มีคลาสซ้อนทับกัน และแบบไซเคิลที่มีเส้นทางซ้อนทับกัน



รูปที่ 4.3 ส่วนการขจัดการเรียกที่ขึ้นต่อกันแบบมีวง

2) ตัวตัดคลาส (Class Slicer)

ตัวตัดคลาสทำหน้าที่ตัดส่วนคลาสเพื่อขจัดการเรียกที่ขึ้นต่อกันแบบมีวง โดยการตัดแบ่งคลาสจะแบ่งตามประเภทของไซเคิล ดังที่ได้อธิบายในหัวข้อ 3.4 เมื่อกระบวนการตัดคลาสเสร็จสิ้น คลาสหลังจากการตัดส่วน (Slice) จะถูกสร้างออกมาเป็นข้อมูลนำออก ในตัวตัดคลาสมีเมท็อดหลักๆ ดังนี้

2.1) เมท็อดสำหรับตัดแบ่งไซเคิลเดียว

การตัดแบ่งแบบไซเคิลเดียวนั้นต้องวิเคราะห์ความสัมพันธ์ระหว่างคลาสต้นทางและคลาสปลายทางตามเงื่อนไขการตัดแบ่งคลาส ซึ่งก็คือเมท็อดในคลาสปลายทางที่ถูกคลาสต้นทางเรียกนั้น จะต้องไม่มีการเรียกไปยังคลาสอื่นในไซเคิล การวิเคราะห์ดังกล่าวต้องอาศัยข้อมูลรายละเอียดคลาส (Class Detail Data) ที่ได้จากตัวตรวจสอบคลาส โดยส่งคลาสต้นทางและคลาสปลายทางเป็นข้อมูลนำเข้า

2.2) เมท็อดสำหรับแบ่งไซเคิลที่มีการซ้อนทับออกเป็นไซเคิลเดียว

เป็นอัลกอริทึมที่มีพื้นฐานมาจากการค้นหาแนวลึก ซึ่งมีหน้าที่แตกกลุ่มเอสซีซีที่มีการเรียกที่ขึ้นต่อกันแบบมีวงประเภทไซเคิลที่มีการซ้อนทับออกเป็นไซเคิลเดียว

2.3) เมท็อดสำหรับตัดแบ่งไซเคิลที่มีคลาสซ้อนทับกัน

ทำการเรียกเมท็อด 2.2) ก่อนเพื่อหากลุ่มเอสซีซีที่เป็นไซเคิลเดียว หลังจากนั้นจึงทำการเรียกการเมท็อด 2.1) โดยส่งคลาสที่มีการซ้อนทับเป็นคลาสปลายทาง ส่วนคลาสต้นทางคือคลาสที่เรียกไปยังคลาสที่มีการซ้อนทับ

2.4) เมท็อดสำหรับตัดแบ่งไซเคิลที่มีเส้นทางซ้อนทับกัน

ทำการเรียกเมท็อด 2.2) ก่อนเช่นเดียวกับเมท็อด 2.3) หลังจากนั้นจึงทำการหากลุ่มเอสซีซีประเภทไซเคิลเดียวที่มีจำนวนคลาสน้อยที่สุด แล้วจึงทำการเรียกการเมท็อด 2.1)

3) ตัวตรวจสอบคลาส (Class Checker)

ตัวตรวจสอบคลาสทำหน้าที่ตรวจสอบรหัสต้นฉบับตามรายการคลาสที่ได้รับเพื่อพิจารณาความสัมพันธ์ระหว่างคลาสต้นทางและคลาสปลายทาง โดยทำการเรียกตัวแจงส่วนภาษาซีชาร์ปเพื่อส่งรหัสต้นฉบับไปแจงส่วน จากนั้นจึงรับโครงสร้างคลาสที่ถูกส่งกลับมา และพิจารณาความสัมพันธ์โดยส่งรายละเอียดคลาสเป็นข้อมูลส่งออก

4) ตัวแจงส่วนภาษาซีชาร์ป

ตัวแจงส่วนภาษาซีชาร์ปในส่วนนี้ทำหน้าที่เดียวกับตัวแจงส่วนในส่วนการวิเคราะห์ความสัมพันธ์ระหว่างคลาส

5) ตัวเชื่อมคลาส (Class Connector)

ตัวเชื่อมคลาสทำหน้าที่สร้างความสัมพันธ์ระหว่างคลาสใหม่หลังจากทำการตัดส่วน โดยมีคลาสหลังจากการตัดส่วนและตารางความสัมพันธ์ระหว่างคลาสเป็นข้อมูลนำเข้า หลังจากทำการเชื่อมคลาสแล้วความสัมพันธ์ระหว่างคลาสหลังการตัดส่วนจะอยู่ในรูปของตารางความสัมพันธ์ระหว่างคลาสที่ปราศจากเอสซีซีเป็นข้อมูลนำออก

6) ตัวสร้างไฟล์ (File Generator)

ตัวสร้างไฟล์ทำหน้าที่สร้างรหัสต้นฉบับของคลาสหลังจากการตัดส่วน โดยสร้างจากคลาสหลังการตัดส่วนที่ได้รับจากตัวตัดคลาส จุดประสงค์ของส่วนประกอบนี้คือ การสร้างไฟล์ของคลาสหลังการตัดส่วนซึ่งถูกเรียกโดยคลาสอื่นๆ ในการทดสอบแบบบูรณาการ

4.2.3 ส่วนการเรียงลำดับคลาสและการสร้างมโนภาพ

ส่วนการเรียงลำดับคลาสและการสร้างมโนภาพทำหน้าที่เรียงลำดับคลาสหลังจากการตัดส่วน และทำการแปลงลำดับคลาสดังกล่าวในรูปของทีดีจีกราฟเพื่อช่วยให้ผู้ทำการทดสอบเห็นลำดับได้ชัดเจนยิ่งขึ้น ส่วนประกอบต่างๆของส่วนการเรียงลำดับคลาสและการสร้างมโนภาพ แสดงในรูปที่ 4.4 ซึ่งแต่ละส่วนประกอบ มีรายละเอียดดังนี้

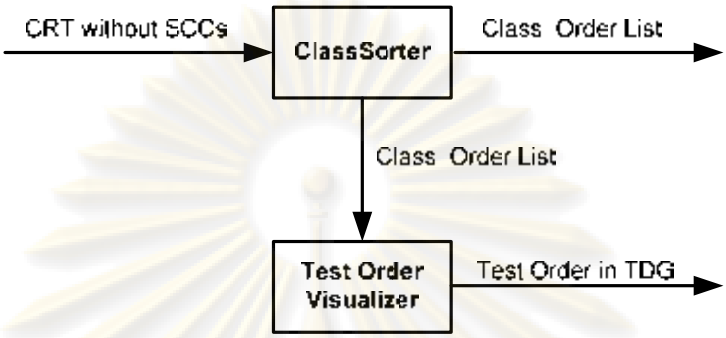
1) ตัวเรียงคลาส (Class Sorter)

ตัวเรียงคลาสทำหน้าที่เรียงคลาสให้อยู่ในรูปของรายการลำดับคลาสจากตารางความสัมพันธ์ระหว่างคลาสที่ปราศจากเอสซีซี โดยดำเนินการหลังจากที่ส่วนการขจัดการเรียกที่ขึ้นต่อกันแบบมีวงดำเนินการเสร็จสิ้น การเรียงคลาสของตัวเรียงคลาสนี้ใช้การเรียงลำดับแบบทอพอโลยีย้อนหลัง โดยเริ่มต้นอ่านจากคลาสที่ไม่ถูกเรียกโดยคลาสใดๆ ซึ่งการเรียงทอพอโลยีแบบย้อนหลังนี้ คลาสแรกที่อ่านจะอยู่เป็นลำดับสุดท้าย ดังนั้นคลาสที่ไม่มีคลาสแม่จะเป็นคลาสแรกที่จะถูกทดสอบ และคลาสลูกของคลาสดังกล่าวจะถูกทดสอบเป็นลำดับถัดไป

2) ตัวสร้างมโนภาพลำดับการทดสอบ (Test Order Visualizer)

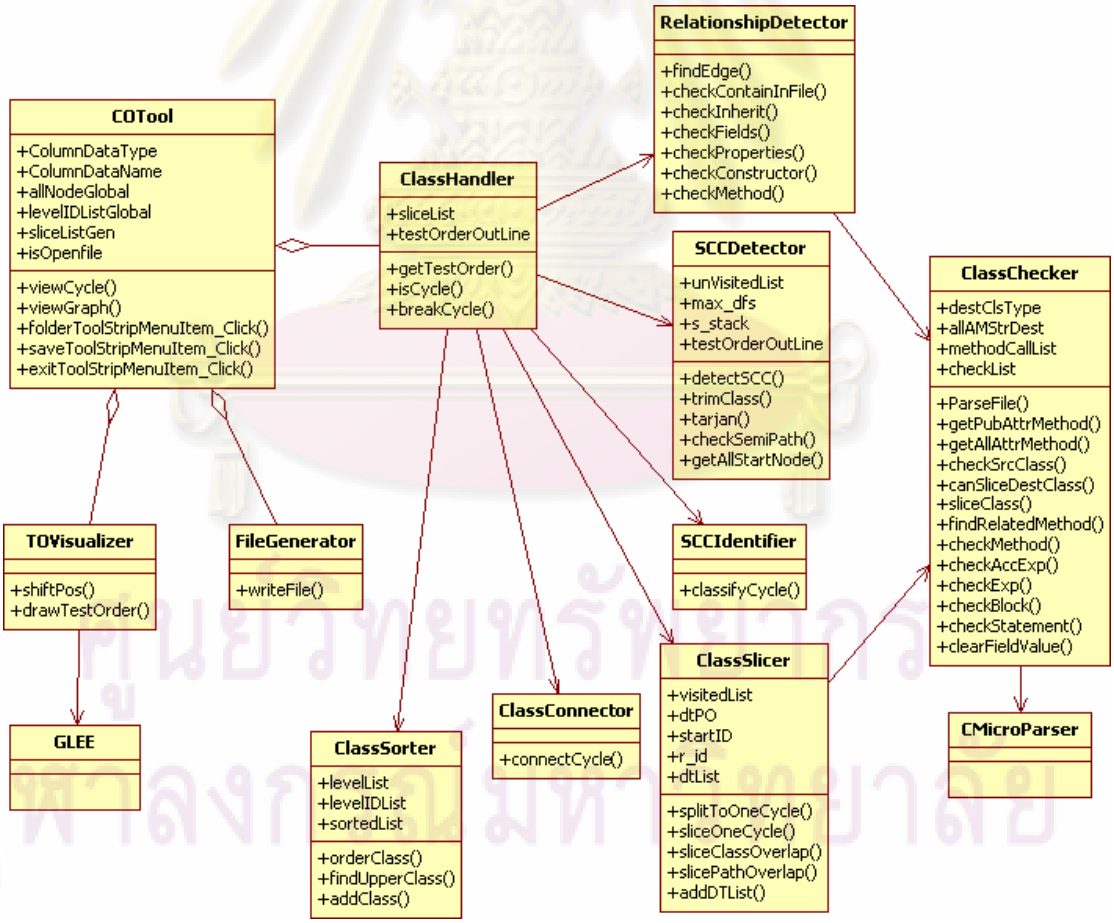
ตัวสร้างมโนภาพลำดับการทดสอบทำหน้าที่แสดงภาพของทีดีจีของความสัมพันธ์ระหว่างคลาสภายใต้การทดสอบ รายการลำดับคลาสที่ได้รับจากตัวเรียงคลาสจะ

ถูกแปลงให้อยู่ในรูปทีดีจีเพื่อแสดงลำดับการทดสอบในลักษณะรูปภาพ การวาดกราฟของตัวสร้างมโนภาพนี้มีพื้นฐานมาจากงานของ Nachmanson et al. [15] และตัวสร้างมโนภาพนี้ใช้จีแอลอีอี (GLEE or Graph Layout Execution Engine) ซึ่งเป็นคลังโปรแกรมหนึ่งสำหรับการวางผังกราฟและการสร้างภาพซึ่งพัฒนาโดยทีมวิจัยของไมโครซอฟท์ [15, 16]



รูปที่ 4.4 ส่วนการเรียงลำดับคลาสและการสร้างมโนภาพ

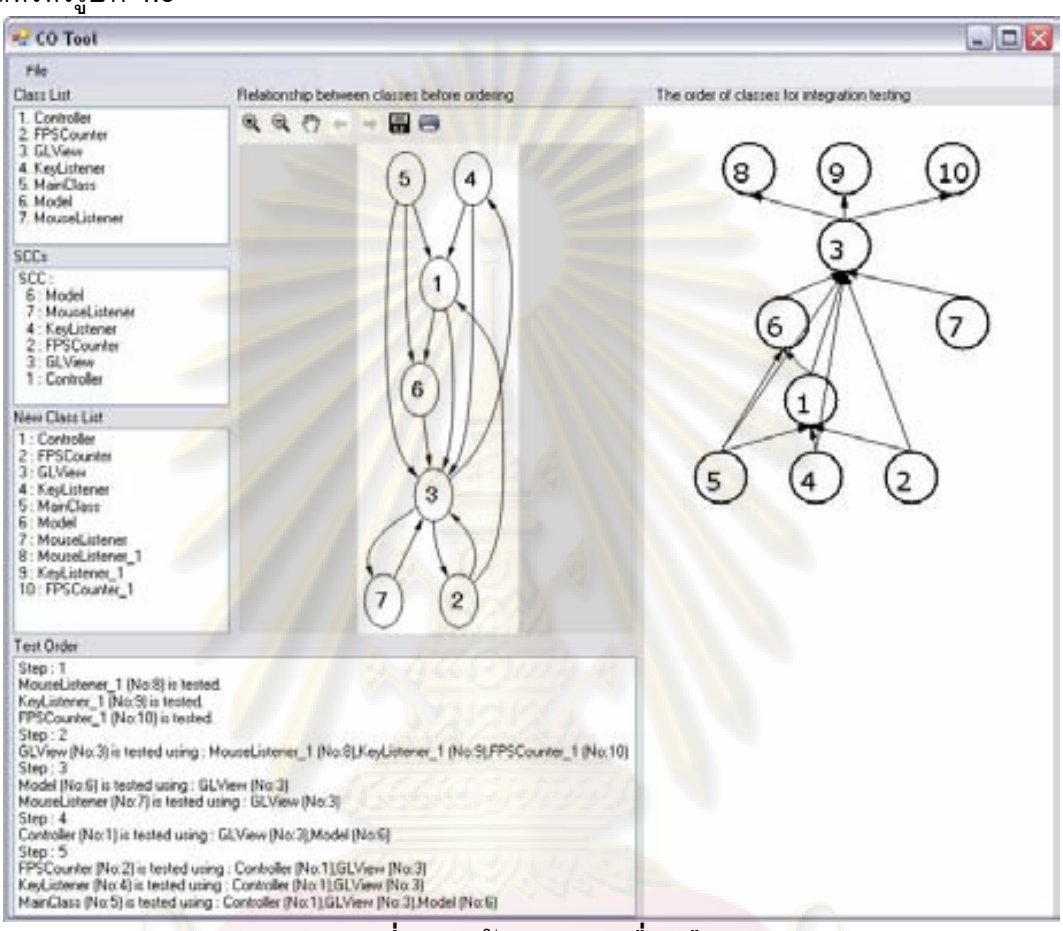
รูปที่ 4.5 แสดงแผนภาพคลาสของเครื่องมือสำหรับการเรียงลำดับการบูรณาการคลาส ซึ่งรายละเอียดของแต่ละคลาสอธิบายในภาคผนวก ก.



รูปที่ 4.5 แผนภาพคลาสของเครื่องมือสำหรับการเรียงลำดับการบูรณาการคลาส

4.3 โปรแกรมประยุกต์ของเครื่องมือ

โปรแกรมประยุกต์ของเครื่องมือสำหรับการเรียงลำดับการบูรณาการคลาสเป็นเครื่องมือที่ถูกพัฒนาขึ้นด้วยภาษาซีชาร์ปเพื่อหาลำดับการทดสอบโดยอัตโนมัติ หน้าจอของเครื่องมือแสดงดังรูปที่ 4.6



รูปที่ 4.6 หน้าจอของเครื่องมือ

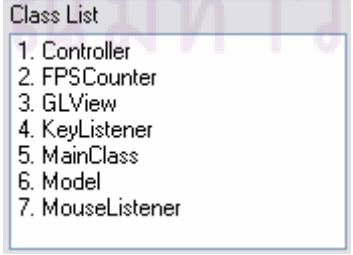
โปรแกรมประยุกต์ของเครื่องมือที่พัฒนาขึ้นประกอบด้วย 3 ส่วนหลัก ดังต่อไปนี้

4.3.1 ส่วนของหน้าต่างที่แสดงรายละเอียดของข้อมูลนำเข้า

หลังจากไฟล์รหัสต้นฉบับถูกนำเข้าไปเป็นข้อมูลนำเข้าของเครื่องมือเพื่อพิจารณารายละเอียดของรหัสต้นฉบับเหล่านี้แสดงใน 3 หน้าต่าง ดังต่อไปนี้

1) หน้าต่างรายการคลาส (Class List window)

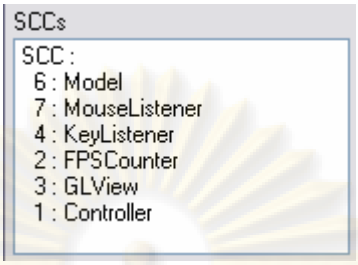
หน้าต่างรายการคลาสอยู่ด้านบนซ้ายของโปรแกรมประยุกต์ ซึ่งแสดงรายชื่อคลาสทั้งหมดในการพิจารณาหาลำดับคลาสสำหรับการทดสอบ หน้าต่างนี้แสดงดังรูปที่ 4.7



รูปที่ 4.7 หน้าต่างรายการคลาส

2) หน้าต่างเอสซีซี (SCCs window)

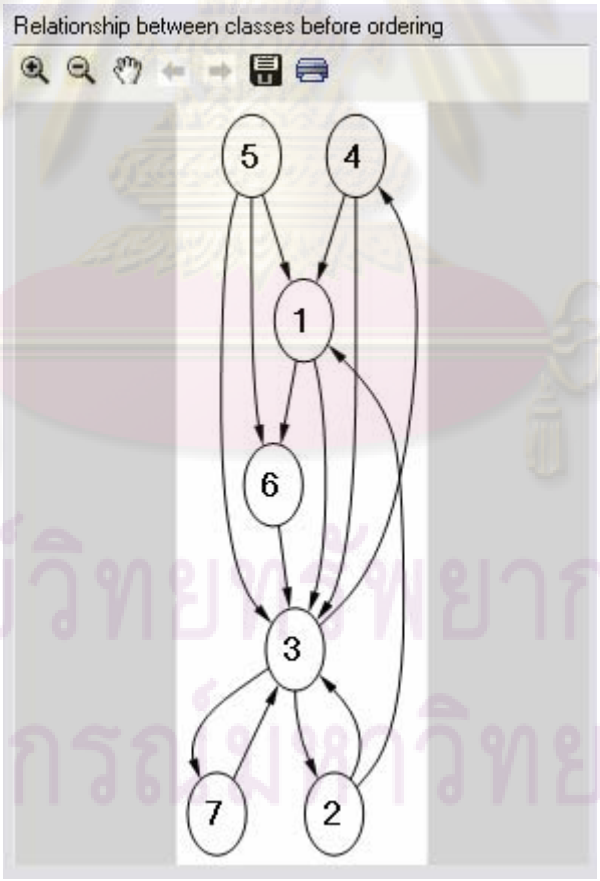
หน้าต่างเอสซีซีอยู่ด้านล่างของหน้าต่างรายการคลาส หน้าต่างนี้แสดงรายชื่อคลาสในแต่ละเอสซีซี หน้าต่างนี้แสดงดังรูปที่ 4.8



รูปที่ 4.8 หน้าต่างเอสซีซี

3) หน้าต่างกราฟเอสซีซี (SCC Graph window)

หน้าต่างกราฟเอสซีซีอยู่ตรงกลางของโปรแกรมประยุกต์ กราฟในหน้าต่างนี้แสดงความสัมพันธ์ระหว่างคลาสก่อนทำการเรียงลำดับคลาส โดยหมายเลขที่โหนดในกราฟจะตรงกับหมายเลขหน้าคลาสในหน้าต่างรายการคลาส ทำให้ทราบว่าโหนดในหมายเลขนั้นมีชื่อคลาสอะไร จุดประสงค์ของหน้าต่างนี้คือการแสดงการเรียกที่ขึ้นต่อกันแบบมีวงของคลาสในรูปของกราฟเพื่อให้ผู้ใช้เห็นภาพได้ชัดเจนขึ้น หน้าต่างนี้แสดงดังรูปที่ 4.9



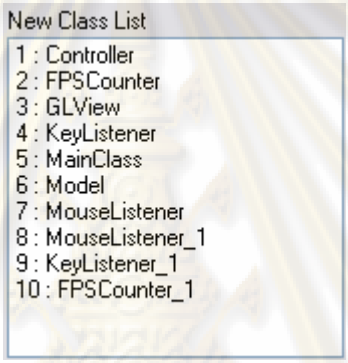
รูปที่ 4.9 หน้าต่างกราฟเอสซีซี

4.3.2 ส่วนของหน้าต่างที่แสดงรายละเอียดของข้อมูลส่งออก

หน้าต่างในส่วนนี้ แสดงลำดับคลาสสำหรับการทดสอบที่เป็นข้อมูลนำออก ซึ่งแสดงในรูปแบบของข้อความ และในรูปแบบของกราฟ รายละเอียดดังกล่าวแสดงใน 3 หน้าต่างดังต่อไปนี้

1) หน้าต่างรายการคลาสใหม่ (New Class List window)

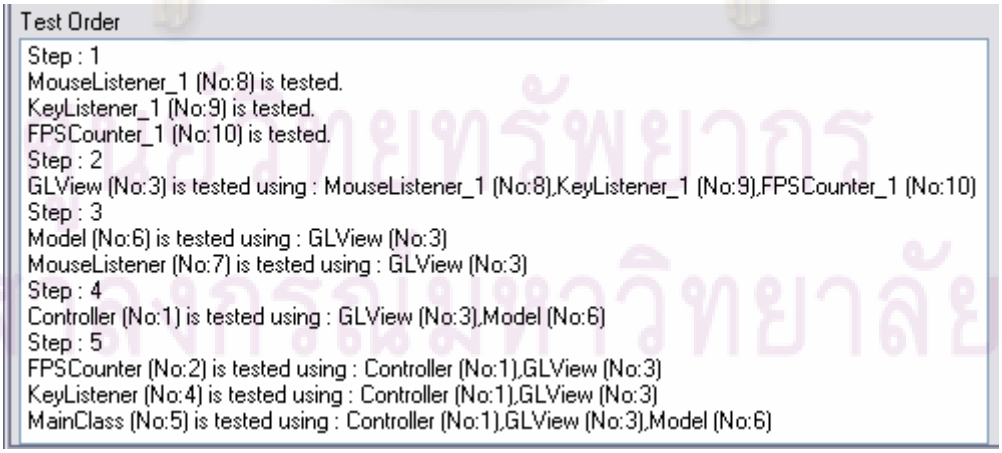
หน้าต่างรายการคลาสใหม่อยู่ด้านล่างของหน้าต่างเอสซีซี หน้าต่างนี้แสดงรายการของคลาสใหม่ ซึ่งหมายเลขและชื่อคลาสของรายการในหน้าต่างนี้ยังไม่ใช่ลำดับการทดสอบ หน้าต่างนี้ถูกใช้เพื่อแสดงหมายเลขและชื่อคลาสของคลาสที่ถูกทำการตัดส่วนเท่านั้น คลาสที่ถูกทำการตัดส่วนแล้วจะกลายเป็นคลาสใหม่ขึ้นมา ซึ่งชื่อคลาสใหม่นี้จะใช้ชื่อคลาสเดิมแต่ตามด้วย _1 ต่อท้าย ส่วนคลาสที่เหลือจากการตัดส่วนยังคงใช้ชื่อเดิม หน้าต่างนี้แสดงดังรูปที่ 4.10



รูปที่ 4.10 หน้าต่างรายการคลาสใหม่

2) หน้าต่างลำดับการทดสอบ (Test Order window)

หน้าต่างลำดับการทดสอบอยู่ด้านล่างซ้ายของโปรแกรมประยุกต์ หน้าต่างนี้แสดงลำดับคลาสสำหรับการทดสอบโดยแสดงเป็นขั้น (Step) โดยจำนวนขั้นการทดสอบหมายถึงจำนวนคลาสของเส้นทางการทดสอบที่ยาวที่สุด คลาสในขั้นเดียวกันสามารถทำการทดสอบขนานกันได้ หน้าต่างนี้แสดงดังรูปที่ 4.11



รูปที่ 4.11 หน้าต่างลำดับการทดสอบ

จากตัวอย่างขั้นการทดสอบในหน้าต่างที่ 4.11 มีทั้งหมด 5 ขั้น ดังต่อไปนี้

ขั้นที่ 1. มี 3 คลาส ที่ต้องทดสอบคลาสในระดับการทดสอบหน่วย คือ

- คลาส MouseListener_1
- คลาส KeyListener_1
- คลาส FPSListener_1

ขั้นที่ 2. คลาส GLView ทำการทดสอบการเชื่อมต่อกับคลาส MouseListener_1, คลาส KeyListener_1, คลาส FPSListener_1

ขั้นที่ 3. มี 2 คลาส

- คลาส Model ทดสอบการเชื่อมต่อกับคลาส GLView
- คลาส MouseListener ทดสอบการเชื่อมต่อกับคลาส GLView

ขั้นที่ 4. คลาส Controller ทดสอบการเชื่อมต่อกับคลาส GLView และคลาส Model

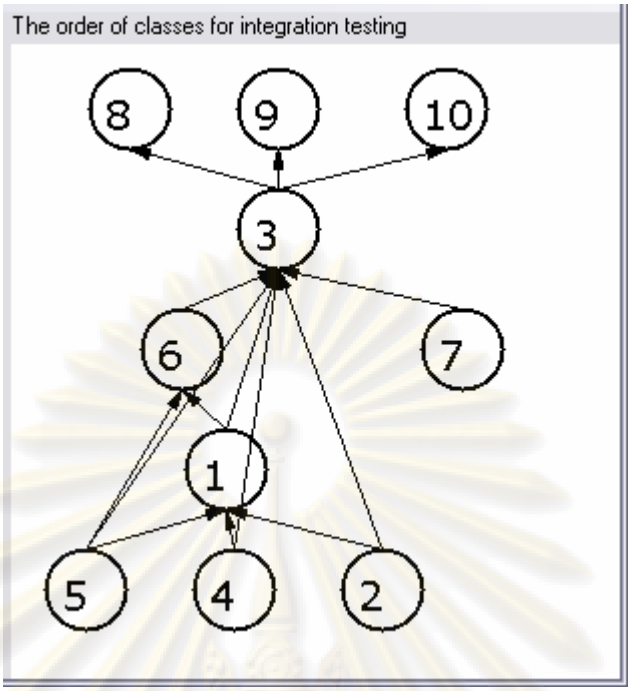
ขั้นที่ 5. มี 3 คลาส คือ

- คลาส MainClass ทดสอบการเชื่อมต่อกับคลาส Controller และคลาส GLView
- คลาส KeyListener ทดสอบการเชื่อมต่อกับคลาส Controller และคลาส GLView
- คลาส FPSListener ทดสอบการเชื่อมต่อกับคลาส Controller, คลาส GLView, คลาส Model

เส้นทางการทดสอบที่ยาวที่สุด คือ เส้นทางการทดสอบคลาสตั้งแต่คลาส KeyListener_1 (หมายเลข 9), GLView (หมายเลข 3), Model (หมายเลข 6), Controller (หมายเลข 1), KeyListener (หมายเลข 4) เนื่องจากจำนวนขั้นการทดสอบ (Step) คือจำนวนคลาสในเส้นทางการทดสอบที่ยาวที่สุด และจากเส้นทางดังกล่าวมีคลาสในเส้นทางทั้งหมด 5 คลาส ดังนั้นจำนวนขั้นการทดสอบของระบบนี้คือ 5 ขั้นนั่นเอง สามารถดูลำดับในรูปแบบของกราฟได้จากรูปที่ 4.12

3) หน้าต่างกราฟลำดับการทดสอบ (Test Order Graph window)

หน้าต่างกราฟลำดับการทดสอบอยู่ด้านขวาของโปรแกรมประยุกต์ เป็นหน้าต่างสำหรับแสดงลำดับคลาสในการทดสอบแบบบูรณาการ การแสดงลำดับคลาสในรูปของกราฟทำให้ผู้ใช้งานสามารถมองเห็นภาพการทดสอบได้ชัดเจนขึ้น หน้าต่างนี้แสดงดังรูปที่ 4.12



รูปที่ 4.12 หน้าต่างกราฟลำดับการทดสอบ

4.3.3 ส่วนของเมนูที่จัดการไฟล์ข้อมูล

ส่วนการจัดการแฟ้มข้อมูลนำเข้าและส่งออกอยู่ในเมนูไฟล์ ซึ่งแสดงดังรูปที่ 4.13 เมนูในส่วนนี้มี 2 เมนู ดังต่อไปนี้

1) เมนูสำหรับเปิดโฟลเดอร์ (Open Folder menu item)

เมนูนี้ทำการเลือกโฟลเดอร์ที่บรรจุไฟล์รหัสต้นฉบับที่ต้องการหาลำดับการทดสอบ จากนั้นรหัสต้นฉบับเหล่านั้นจะถูกนำเข้ามาเป็นข้อมูลนำเข้า เพื่อทำการเรียงคลาสสำหรับการทดสอบ

2) เมนูสำหรับสร้างคลาสหลังการตัดส่วน (Generate Slice menu item)

เมนูนี้ถูกใช้สำหรับสร้างไฟล์รหัสต้นฉบับของคลาสใหม่ ซึ่งได้จากคลาสหลังการตัดส่วน ชื่อไฟล์ของคลาสใหม่จะเป็นชื่อที่มีส่วนต่อท้าย คือตามด้วย _1 อย่างไรก็ตามชื่อของคลาสจริงๆ ในเนื้อหาของไฟล์ใหม่ที่ได้หลังจากการสร้างนั้น ยังคงเป็นชื่อเดิม

รูปที่ 4.13 รายการเมนูของเครื่องมือ

ขั้นตอนวิธีการใช้เครื่องมือที่พัฒนาขึ้นแสดงในภาคผนวก ข

บทที่ 5
การทดลอง

ในบทนี้จะกล่าวถึงรายละเอียดของการทดลอง เพื่อทำการทดสอบแนวคิดที่ได้นำเสนอด้วยเครื่องมือซึ่งผู้วิจัยพัฒนาขึ้นมา ประเด็นต่างๆ ในการทดลองประกอบด้วย ขั้นตอนการทดลอง สภาพแวดล้อมของการทดลอง ผลการทดลอง และการวิเคราะห์ผลการทดลอง ซึ่งรายละเอียดมีดังนี้

5.1 ขั้นตอนการทดลอง

ขั้นตอนการทดลองเครื่องมือที่พัฒนาขึ้น ประกอบไปด้วยขั้นตอนต่างๆ ดังนี้

1) เลือกระบบสำหรับเป็นกรณีศึกษา 3 ระบบ โดยรับรหัสต้นฉบับเป็นข้อมูลนำเข้า
2) หาลำดับคลาสสำหรับการทดสอบแบบบูรณาการด้วยเครื่องมือที่พัฒนาขึ้น
3) ตรวจสอบผลการทดลอง
4) เปรียบเทียบผลการทดลองที่ได้กับผลการทดลองจากวิธีการเรียงลำดับคลาสอื่นๆ
5) วิเคราะห์ผลการทดลอง

5.2 สภาวะที่ใช้ทดสอบเครื่องมือ

เครื่องคอมพิวเตอร์ที่ใช้ในการทดลอง มีรายละเอียดดังนี้

คอมพิวเตอร์พีซี Pentium 4 3.4 GHz

หน่วยความจำหลัก 1024 MB

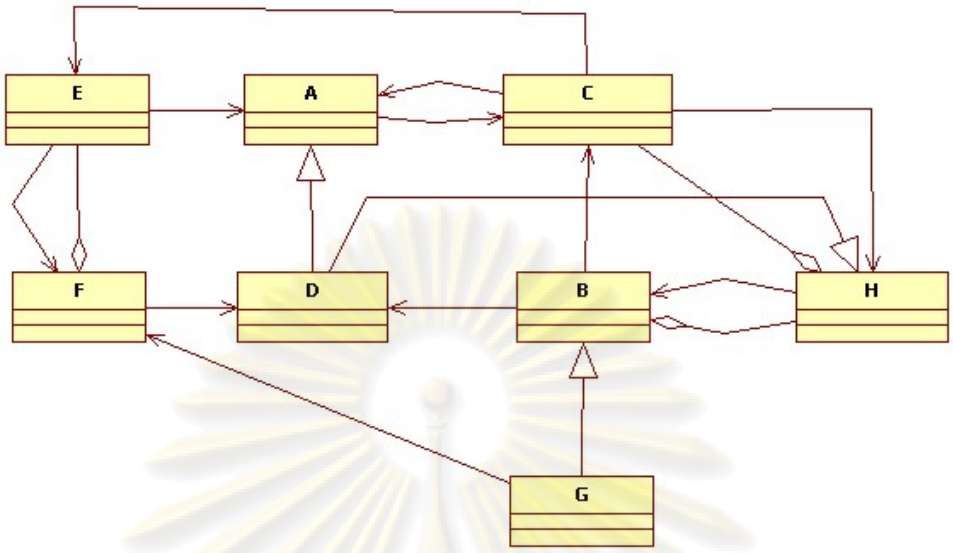
ฮาร์ดดิสก์ความจุ 120 GB

5.3 วิธีการทดลอง

การทดลองเครื่องมือที่พัฒนาขึ้นนั้นใช้รหัสต้นฉบับจากกรณีศึกษา 3 ระบบ ซึ่งแต่ละกรณีศึกษา มีรายละเอียดดังนี้

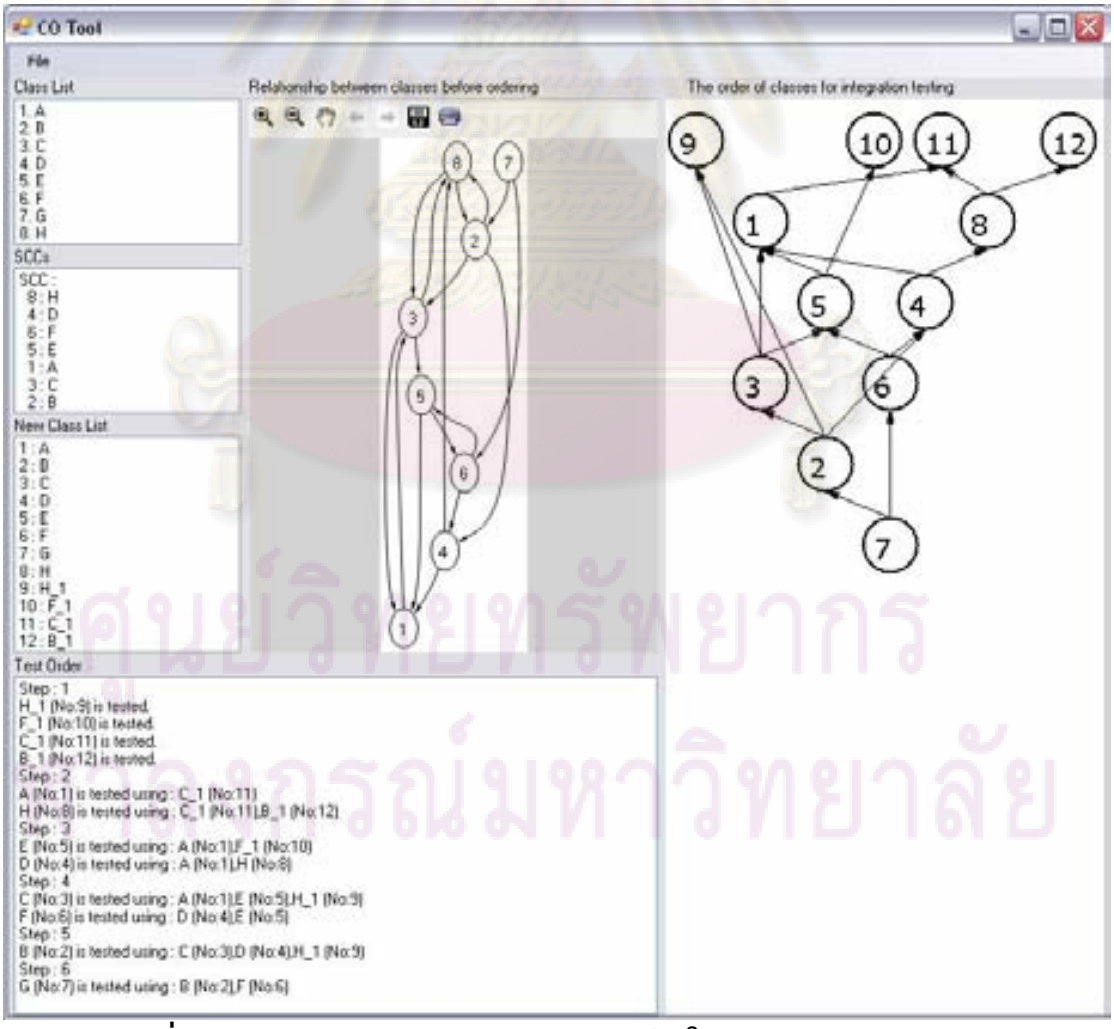
5.3.1 ตัวอย่างจากงานวิจัยของ Briand et al.

กรณีศึกษากรณีนี้เป็นตัวอย่างจากงานวิจัยเรื่อง An Investigation of Graph-Based Class Integration Test Order Strategies [17] ของ L.C Briand, Y. Labiche และ Y. Wang ตัวอย่างนี้ประกอบไปด้วยคลาสจำนวน 8 คลาส โดยกลุ่มคลาสที่มีการเรียกที่ขึ้นต่อกันแบบมีวงคือ A, B, C, D, E, F, H รวมเป็น 1 เอสซีซี ซึ่งเป็นไซเคิลแบบที่มีเส้นทางซ้อนทับกัน คลาสจากตัวอย่างแสดงดังแผนภาพคลาสในรูปที่ 5.1



รูปที่ 5.1 แผนภาพคลาสตัวอย่างจากงานวิจัยของ Briand et al. [15]

หลังจากทำการเรียงคลาสด้วยเครื่องมือที่พัฒนาขึ้น คลาส H, F, C, B ถูกตัดส่วนเพื่อขจัดการเรียกที่ขึ้นต่อกันแบบมีวง ซึ่งลำดับการทดสอบคลาสเป็นไปตามรูปที่ 5.2



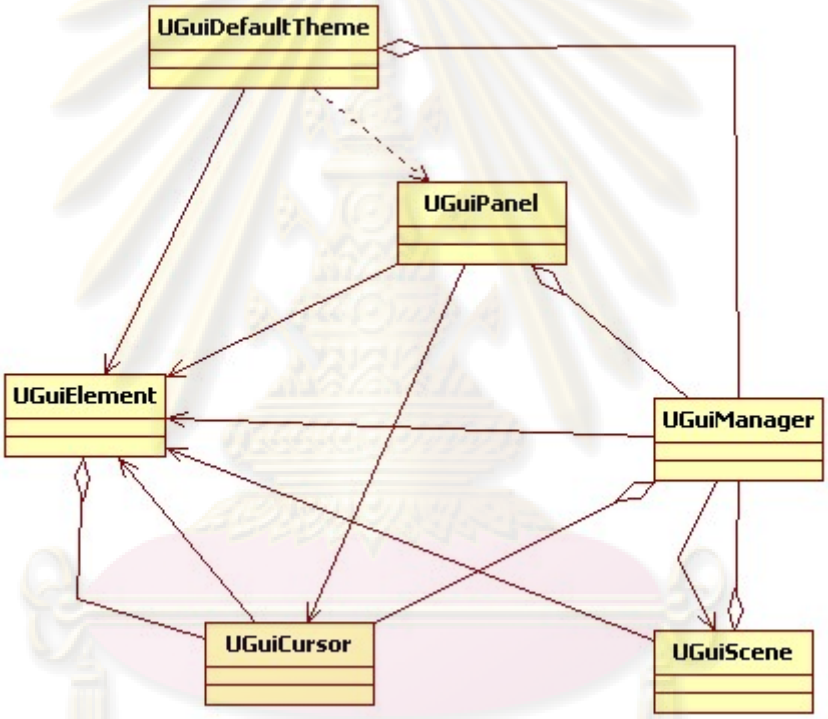
รูปที่ 5.2 ลำดับการทดสอบคลาสจากตัวอย่างในวิจัยของ Briand et al.

5.3.2 ระบบ Uggi3D

ระบบ Uggi3D [18] เป็นกลไกการวาดภาพสามมิติซึ่งเขียนด้วยภาษาซีชาร์ปโดยใช้ไดเรกเอกซ์ (DirectX) ระบบนี้ช่วยในการโปรแกรมเกมสองมิติและสามมิติ งานวิจัยนี้เลือกคลาสจำนวน 6 คลาส ซึ่งมีความเกี่ยวข้องกับกลุ่มเอสซีซีเท่านั้น ซึ่งกลุ่มเอสซีซีดังกล่าวประกอบไปด้วย 2 เอสซีซี ดังนี้

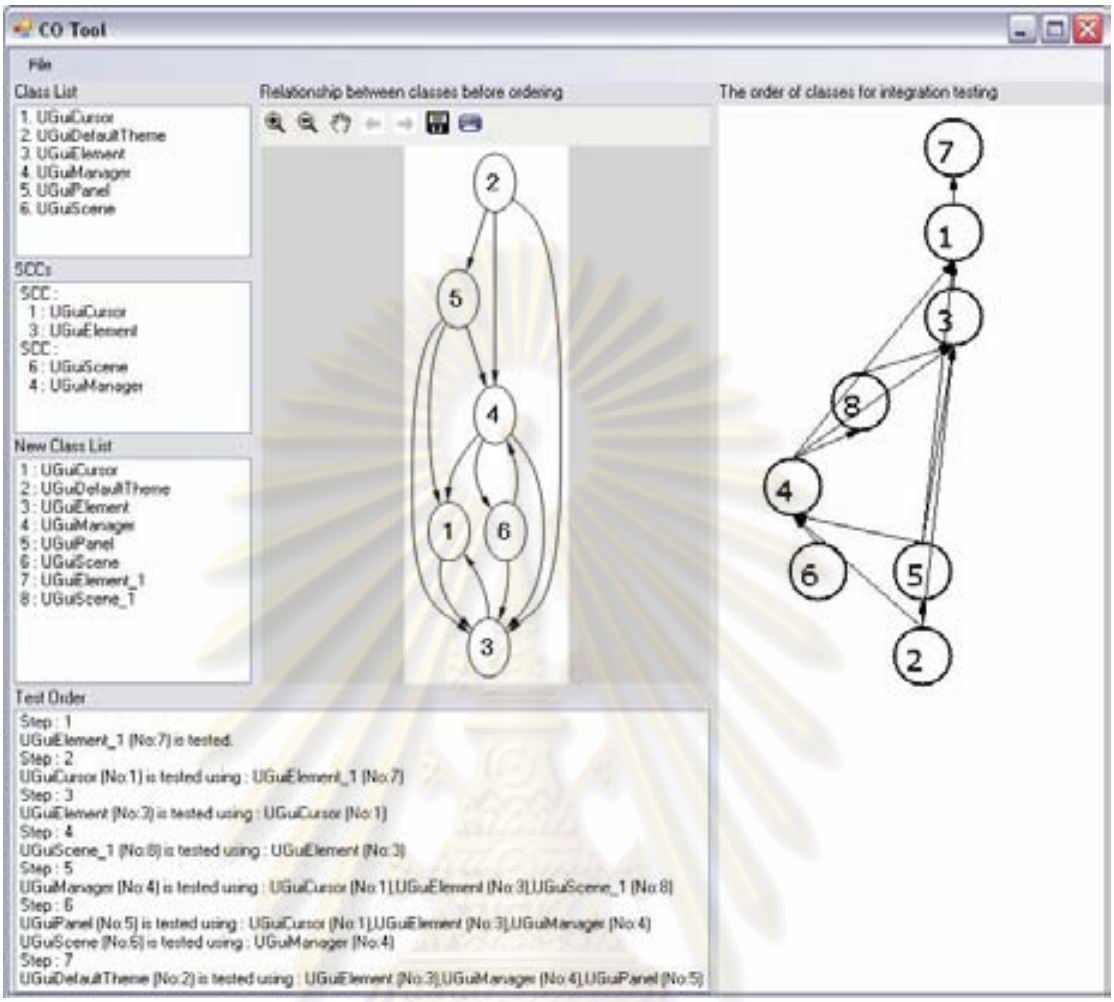
1) UGuiElement และ UGuiCursor เป็นเอสซีซีที่เป็นแบบหนึ่งไซเคิล
2) UGuiManager และ UGuiScene เป็นเอสซีซีที่เป็นไซเคิลแบบหนึ่งไซเคิล

คลาสของระบบนี้แสดงดังแผนภาพคลาสในรูปที่ 5.3



รูปที่ 5.3 แผนภาพคลาสของระบบ Uggi3D

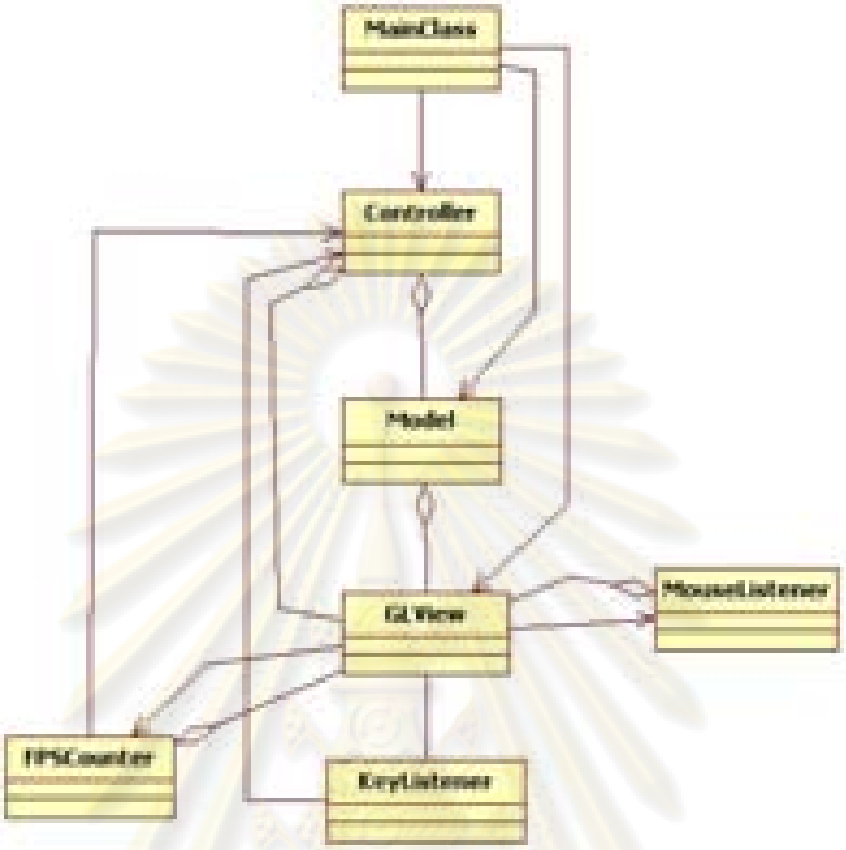
หลังจากทำการเรียงคลาสด้วยเครื่องมือที่พัฒนาขึ้น คลาส UGuiScene และ UGuiElement ถูกตัดส่วนเพื่อขจัดการเรียกที่ขึ้นต่อกันแบบมีวง ซึ่งลำดับการทดสอบคลาสเป็นไปตามรูปที่ 5.4



รูปที่ 5.4 ลำดับการทดสอบคลาสของระบบ Uggi3D

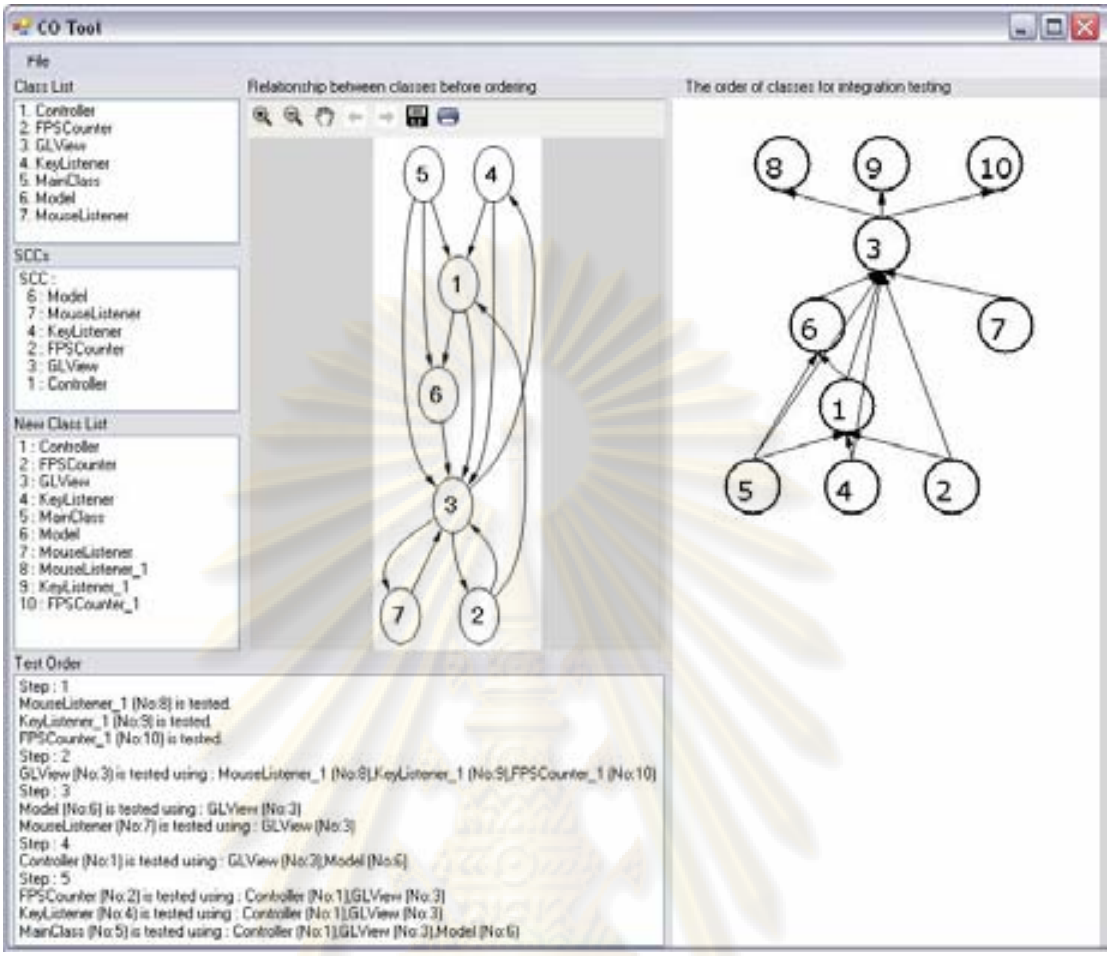
5.3.3 ระบบ Triangulated Surface Drawing Utility

ระบบ Triangulated Surface Drawing Utility (TSDU) [19] เป็นโปรแกรมอรรถประโยชน์ซึ่งเป็นโอเพนจีแอลและเขียนด้วยภาษาซีชาร์ป โปรแกรมนี้ใช้ในการวาดพื้นที่สามเหลี่ยมและประยุกต์เอฟเฟกต่างๆ ระบบนี้ประกอบด้วยคลาสเป็นจำนวนมาก แต่ในงานวิจัยนี้เลือกเฉพาะกลุ่มคลาสที่มีการเรียกที่ขึ้นต่อกันแบบมีวงเท่านั้น ซึ่งประกอบไปด้วยคลาสจำนวน 7 คลาส โดยกลุ่มคลาสที่มีการเรียกที่ขึ้นต่อกันแบบมีวงคือ Controller, Model, GLView, MouseListener, FPSCounter, KeyListener รวมเป็น 1 เอสซีซี ซึ่งเป็นไซเคิลแบบที่มีเส้นทางซ้อนทับกัน คลาสของระบบนี้แสดงดังแผนภาพคลาสในรูปที่ 5.5



รูปที่ 5.5 แผนภาพคลาสของระบบ TSDU

หลังจากทำการเรียงคลาสด้วยเครื่องมือที่พัฒนาขึ้น คลาส MouseListener, KeyListener, FPSCounter ถูกตัดส่วนเพื่อขจัดการเรียกที่ขึ้นต่อกันแบบมีวง ซึ่งลำดับการทดสอบคลาสเป็นไปตามรูปที่ 5.6



รูปที่ 5.6 ลำดับการทดสอบคลาสของระบบ TSDU

5.4 ผลการทดลอง

เมื่อนำผลการทดลองจากทั้ง 3 กรณีศึกษา มาเปรียบเทียบกับวิธีการเรียงคลาสสำหรับการทดสอบแบบบูรณาการอื่นๆ โดยวัดจาก

1) จำนวนสตับ

2) จำนวนขั้นการทดสอบ (Step) โดยจำนวนขั้นการทดสอบหมายถึงจำนวนคลาสของเส้นทางการทดสอบที่ยาวที่สุด ตัวอย่างจำนวนขั้นการทดสอบอธิบายอยู่ในบทที่ 4 หน้าที่ 36

3) จำนวนลักษณะประจำ และเมท็อดของสตับเทียบกับส่วนโปรแกรมที่ถูกแบ่งส่วน

4) จำนวนบรรทัดของคำสั่งของสตับกับจำนวนบรรทัดของคำสั่งที่ต้องทำสำเนาเพิ่มจากการตัดส่วน

5) ความซับซ้อนของลำดับการทดสอบ (Test Order Complexity) รายละเอียดการคำนวณอธิบายในภาคผนวก ค

ผลการเปรียบเทียบแสดงในตารางที่ 5.1 ตารางที่ 5.2 และตารางที่ 5.3 ตามลำดับ

ตารางที่ 5.1 ผลการเปรียบเทียบจากตัวอย่างงานวิจัยของ Briand et al.

Step	Tested classes in each step			
	Tai's approach	**Traon's approach**	**Briand's approach**	**Our approach**
1	Construct: stub(C, A) stub(D, F) stub(H, C) stub(F, E) stub(B, H)	Construct: stub(C, A) stub(B, H) stub(C, H) stub(F, E)	Construct: stub(C, A) stub(H, C) stub(F, E) stub(B, H)	Unit test : slice(C_1, A) slice(F_1, E) slice(B_1, H) slice(H_1, C)
2	- A tested using : stub(C, A)	- A tested using : stub(C, A) - H is tested using : (C, H), stub(B, H)	- A tested using : stub(C, A)	- A is tested using : C_1 - H is tested using : C_1 ,B_1
3	- E is tested using : A, stub(F, E)	- D is tested using : A, H - E is tested using : A, stub(F, E)	- E is tested using : A, stub(F, E)	- E is tested using : A , F_1 - D is tested using : A , H
4	- C is tested using : A, E, stub(H, C) - F is tested using : E, stub(D, F)	- F is tested using : E, D - C is tested using : A, E, H	- C is tested using : A, E, stub(H, C)	- C is tested using : A ,E ,H_1 - F is tested using : D ,E
5	- H is tested using : C, stub(B, H)	- B is tested using : D, C, H	- H is tested using : C, stub(B, H)	- B is tested using : C ,D ,H_1
6	- D is tested using : A, H	- G is tested using : B, F	- D is tested using : A, H	- G is tested using : B ,F
7	- B is tested using : D, C, H		- F is tested using : E, D - B is tested using : D, C, H	
8	- G is tested using : B, F		- G is tested using : B, F	
# Steps	8	6	8	6
Specific stubs	- 1 for C - 1 for H - 1 for F - 1 for D - 1 for B	- 2 for C - 1 for B - 1 for F	- 1 for C - 1 for F - 1 for H - 1 for B	None
# Stubs	5	4	4	0

ตารางที่ 5.2 ผลการเปรียบเทียบของระบบ Uggi3D

Step	Tested classes in each step			
	Tai's approach	**Traon's approach**	**Briand's approach**	**Our approach**
1	Construct: stub(UGuiElement,UGuiCursor) stub(UGuiScene,UGuiManager)	Construct: stub(UGuiCursor,UGuiElement) stub(UGuiManager,UGuiScene)	Construct: stub(UGuiElement,UGuiCursor) stub(UGuiScene,UGuiManager)	Unit test : slice(UGuiElement_1,UGuiCursor) slice(UGuiScene_1,UGuiManager)
2	- UGuiCursor is tested using : stub(UGuiElement,UGuiCursor)	- UGuiElement is tested using : stub(UGuiCursor,UGuiElement)	- UGuiCursor is tested using : stub(UGuiElement,UGuiCursor)	- UGuiCursor is tested using : UGuiElement_1
3	- UGuiElement is tested using : UGuiCursor	- UGuiCursor is tested using : UGuiElement - UGuiScene is tested using : UGuiElement , stub(UGuiManager,UGuiScene)	- UGuiElement is tested using : UGuiCursor	- UGuiElement is tested using : UGuiCursor
4	- UGuiManager is tested using : UGuiCursor , UGuiElement , stub(UGuiScene , UGuiManager)	- UGuiManager is tested using : UGuiCursor , UGuiScene , UGuiElement	- UGuiManager is tested using : UGuiCursor ,UGuiElement ,stub(UGuiScene,UGuiManager)	- UGuiScene_1 is tested using : UGuiElement
5	- UGuiPanel is tested using : UGuiCursor , UGuiManager , UGuiElement - UGuiScene is tested using : UGuiElement , UGuiManager	- UGuiPanel is tested using : UGuiCursor , UGuiManager , UGuiElement	- UGuiPanel is tested using : UGuiCursor , UGuiManager , UGuiElement - UGuiScene is tested using : UGuiElement , UGuiManager	- UGuiManager is tested using : UGuiCursor , UGuiElement , UGuiScene_1

ตารางที่ 5.2 ผลการเปรียบเทียบของระบบ Uggi3D (ต่อ)

Step	Tested classes in each step			
	Tai's approach	**Traon's approach**	**Briand's approach**	**Our approach**
6	-UGuiDefaultTheme is tested using : UGuiElement , UGuiManager , UGuiPanel	- UGuiDefaultTheme is tested using : UGuiElement , UGuiManager , UGuiPanel	- UGuiDefaultTheme is tested using : UGuiElement , UGuiManager , UGuiPanel	- UGuiPanel is tested using : UGuiCursor , UGuiElement , UGuiManager - UGuiScene is tested using : UGuiManager
7				- UGuiDefaultTheme is tested using: UGuiElement , UGuiManager , UGuiPanel
# Steps	6	6	6	7
Specific stubs	- 1 for UGuiElement - 1 for UGuiScene	- 1 for UGuiCursor - 1 for UGuiManager	- 1 for UGuiElement - 1 for UGuiScene	None
# Stubs	2	2	2	0
# Attr	2	0	2	16
# Methods	8	0	8	9
Stubs' LOC	61	29	61	0
Slices' LOC	0	0	0	414
Test Order Complexity	2.72	2	2.72	0

ตารางที่ 5.3 ผลการเปรียบเทียบของระบบ TSDU

Step	Tested classes in each step			
	Tai's approach	**Traon's approach**	**Briand's approach**	**Our approach**
1	Construct: stub(MouseListener,GLView) stub(FPSCounter,GLView) stub(KeyListener,GLView) stub(Controller , MainClass) stub(Model , MainClass) stub(Controller , KeyListener) stub(Controller , FPSCounter)	Construct: stub(GLView,Model) stub(GLView,MouseListener) stub(GLView , Controller) stub(GLView , FPSCounter) stub(GLView , KeyListener)	Construct: stub(KeyListener,GLView) stub(FPSCounter,GLView) stub(MouseListener,GLView)	Unit test : slice(MouseListener_1,GLView) slice(KeyListener_1,GLView) slice(FPSCounter_1,GLView)
2	- GLView is tested using : stub(MouseListener,GLView) , stub(FPSCounter,GLView) , stub(KeyListener,GLView)	- Model is tested using : stub(GLView,Model) - MouseListener is tested using : stub(GLView,MouseListener)	- GLView is tested using : stub(KeyListener,GLView) , stub(FPSCounter,GLView) , stub(MouseListener,GLView)	- GLView is tested using: MouseListener_1 , KeyListener_1 , FPSCounter_1
3	- MainClass is tested using : GLView , stub(Controller, MainClass) , stub(Model, MainClass) - Model is tested using : GLView - MouseListener is tested using : GLView - KeyListener is tested using : GLView , stub(Controller , KeyListener) - FPSCounter is tested using : GLView , stub(Controller , FPSCounter)	- Controller is tested using : Model , stub(GLView , Controller)	- Model is tested using : GLView - MouseListener is tested using : GLView	- Model is tested using : GLView - MouseListener is tested using : GLView

ตารางที่ 5.3 ผลการเปรียบเทียบของระบบ TSDU (ต่อ)

Step	Tested classes in each step			
	Tai's approach	**Traon's approach**	**Briand's approach**	**Our approach**
4	Controller is tested using : Model ,GLView	- FPSCounter is tested using : Controller , stub(GLView , FPSCounter) - KeyListener is tested using : Controller , stub(GLView , KeyListener)	- Controller is tested using : Model ,GLView	- Controller is tested using : Model ,GLView
5		- GLView is tested using : MouseListener , KeyListener , FPSCounter	- MainClass is tested using : Controller ,Model ,GLView - FPSCounter is tested using : Controller ,GLView - KeyListener is tested using : Controller ,GLView	- MainClass is tested using : Controller ,Model ,GLView - FPSCounter is tested using : Controller ,GLView - KeyListener is tested using : Controller ,GLView
6		- MainClass is tested using : Controller , Model , GLView		
# Steps	4	6	5	5
Specific stubs	- 3 for Controller - 1 for MouseListener - 1 for Model - 1 for FPSCounter - 1 for MouseListener	- 5 for GLView	- 1 for KeyListener - 1 for FPSCounter - 1 for MouseListener	None
# Stubs	7	5	3	0

ตารางที่ 5.3 ผลการเปรียบเทียบของระบบ TSDU (ต่อ)

	Tai's approach	Traon's approach	Briand's approach	Our approach
# Attr	4	19	1	16
# Methods	14	16	10	10
Stubs' LOC	129	241	61	0
Slices' LOC	0	0	0	190
Test Order Complexity	8.32	7.11	3.94	0

5.5 การวิเคราะห์ผลการทดลอง

จากผลการทดลองหาลำดับการทดสอบการบูรณาการคลาสด้วยเครื่องมือที่พัฒนาขึ้นเปรียบเทียบกับวิธีการหาลำดับวิธีอื่นๆ นำมาวิเคราะห์ได้ดังต่อไปนี้

1) จากตารางที่ 5.1 เนื่องจากกรณีศึกษานี้เป็นตัวอย่างจากงานวิจัยสำหรับเปรียบเทียบผลการทดลองในเรื่องการหาลำดับการทดสอบคลาส จึงไม่มีรหัสต้นฉบับจริงสำหรับทำการตัดส่วน ดังนั้นการเปรียบเทียบในตารางนี้จึงเปรียบเทียบเฉพาะจำนวนสตับและจำนวนขั้นการทดสอบเท่านั้น ซึ่งเมื่อเปรียบเทียบจากลำดับขั้นการทดสอบ ผลการเรียงลำดับที่ได้จากเครื่องมือมีจำนวนขั้นเท่ากับผลจากงานวิจัยของ Traon et al. โดยการเรียงลำดับคลาสที่ได้จากเครื่องมือกำหนดให้สามารถตัดส่วนคลาสได้ทุกคลาสในไซเคิล ดังนั้นจึงไม่จำเป็นต้องใช้สตับ ซึ่งผลการทดลองจากงานวิจัยอื่นๆ นั้นจำเป็นต้องใช้สตับในการเรียงลำดับคลาส

2) จากตารางที่ 5.2 เมื่อเปรียบเทียบผลที่ได้จากเครื่องมือกับผลจากงานวิจัยอื่นๆ ในแง่ของจำนวนสตับนั้น จากตัวอย่างกรณีศึกษาสามารถทำการตัดส่วนคลาสได้ทุกไซเคิล ดังนั้นจึงไม่จำเป็นต้องใช้สตับ เมื่อเปรียบเทียบระบบ Uggi3D จากตัววัดอื่นๆ ผลที่ได้มีดังต่อไปนี้

2.1) จำนวนขั้นการทดสอบของผลที่ได้จากเครื่องมือมีจำนวนขั้นมากกว่าวิธีอื่นๆ อยู่ 1 ขั้น แต่ถ้า slice(UGuiElement_1,UGuiCursor)และ slice(UGuiScene_1,UGuiManager) ผ่านการทดสอบแบบหน่วยแล้ว สามารถเริ่มต้อนการทดสอบได้ที่ขั้นที่ 2 ดังนั้นในกรณีนี้จำนวนขั้นการทดสอบจะเท่ากับผลที่ได้จากงานวิจัยอื่นๆ

2.2) จำนวนสตับในงานวิจัยอื่นๆ จะต้องใช้ 2 สตับเท่ากัน แต่ผลที่ได้จากเครื่องมือไม่จำเป็นต้องใช้สตับ

2.3) เมื่อเปรียบเทียบจำนวนลักษณะประจำและเมท็อดของสตับ จากงานวิจัยของ Traon et al. เนื่องจากสตับทั้งสองสร้างจากความสัมพันธ์แบบแอกกรีเกชันโดยไม่มีการเข้าถึงลักษณะประจำหรือเมท็อด ดังนั้นจำนวนลักษณะประจำและเมท็อดจึงมีค่าเป็นศูนย์

2.4) จำนวนลักษณะประจำและเมท็อดที่ได้จากเครื่องมือมีจำนวนมากกว่าวิธีอื่นๆ เนื่องจากเมท็อดของคลาส UGuiElement_1 และ UGuiScene_1 ที่ถูกเรียกนั้นมีการเรียกเมท็อดและลักษณะประจำอื่นๆ ในคลาสด้วย ทำให้จำนวนบรรทัดคำสั่งของคลาสที่ถูกทำการตัดส่วนมีมากกว่าจำนวนบรรทัดคำสั่งของสตับตามไปด้วย

2.5) ความซับซ้อนของลำดับการทดสอบวัดจากจำนวนสตับ ซึ่งผลการทดลองจากงานวิจัยของ Tai et al. และ Briand et al. มีค่าความซับซ้อนเท่ากัน เนื่องจากสตับที่ได้จากทั้งสองงานวิจัยนั้นเป็นตัวเดียวกัน ถึงแม้ว่าสตับจากงานวิจัยของ Traon et al. จะไม่มีการสร้างลักษณะประจำและเมท็อด แต่ยังคงต้องสร้างคลาสสำหรับทดสอบ ดังนั้นค่าความซับซ้อนจึงเกิดจากค่าคงที่สำหรับการสร้างสตับเท่านั้น สำหรับค่าความซับซ้อนของผลการทดลองที่ได้จากเครื่องมือมีค่าเป็นศูนย์เนื่องจากไม่มีการสร้างสตับ

3) จากตารางที่ 5.3 ผลการทดลองที่ได้จากเครื่องมือ เนื่องจากคลาสในทุกไซเคิลสามารถทำการตัดส่วนได้ ดังนั้นวิธีนี้จึงไม่จำเป็นต้องใช้สตับเช่นกัน เมื่อเปรียบเทียบระบบ TSDU จากตัววัดอื่นๆ ผลที่ได้มีดังต่อไปนี้

3.1) จากกรณีศึกษากรณีนี้ จำนวนขั้นการทดสอบของผลการทดลองจากงานวิจัยของ Tai et al. มีจำนวนขั้นน้อยที่สุด เนื่องจากวิธีนี้ลบความสัมพันธ์ระหว่างคลาสมากที่สุด ทำให้จำนวนขั้นการทดสอบมีจำนวนลดลง

3.2) จำนวนสตับจากงานวิจัยของ Tai et al. มีมากที่สุดคือ 7 สตับ รองลงมาคือผลจากงานวิจัยของ Traon et al. มี 5 สตับ และสุดท้ายคืองานวิจัยของ Briand et al. มี 3 สตับ สำหรับผลที่ได้จากเครื่องมือไม่จำเป็นต้องใช้สตับ แต่เนื่องคลาสที่ถูกตัดส่วนทั้ง 3 คลาส ไม่มีการเรียกไปยังคลาสอื่น ทำให้ลำดับขั้นการทดสอบของผลที่ได้จากเครื่องมือ และของ Briand et al. มีลำดับที่เหมือนกัน

3.3) จำนวนลักษณะประจำและเมท็อดของผลการทดลองจากงานวิจัยของ Traon et al. มีมากที่สุด เนื่องจากความสัมพันธ์ที่ถูกลบนั้นมีการเรียกแบบดีลีเกต (delegate) ดังนั้นในการสร้างสตับจำนวนลักษณะประจำและเมท็อดที่เกี่ยวข้องจึงมีมากขึ้น สำหรับผลการทดลองจากงานวิจัยของ Tai et al. ที่มีจำนวนลักษณะประจำและเมท็อดมากรองลงมาเนื่องจากจำนวนสตับที่ต้องสร้างมากที่สุด ดังนั้นจำนวนลักษณะประจำและเมท็อดจึงมีมากตาม

3.4) สตับจากผลการทดลองของ Briand et al. และคลาสที่ถูกตัดส่วนจากเครื่องมือเป็นคลาสเดียวกัน จำนวนเมท็อดจึงมีจำนวนเท่ากัน แต่เมท็อดของคลาสหลังการตัดส่วนนั้นมีการเรียกลักษณะประจำภายในด้วย ดังนั้นจำนวนลักษณะประจำจึงมีมากกว่า

3.5) ความซับซ้อนของลำดับการทดสอบจากตัวอย่างนี้แปรผันตามจำนวนสตับ ยิ่งจำนวนสตับมีมากความซับซ้อนจึงมากตาม ผลการทดลองของ Tai et al. มีจำนวน 7 สตับ ค่าความซับซ้อนมีค่าเท่ากับ 8.32 ซึ่งมีค่ามากที่สุด รองลงมาคือผลการทดลองของ Traon et al. มีจำนวน 5 สตับ ค่าความซับซ้อนมีค่าเท่ากับ 7.11 และค่าที่น้อยเป็นลำดับถัดไปคือผลการทดลองของ Briand et al. มีจำนวน 3 สตับ ค่าความซับซ้อนมีค่าเท่ากับ 3.94 และเนื่องจากผลการทดลองที่ได้จากเครื่องมือไม่มีการสร้างสตับ ดังนั้นค่าความซับซ้อนจึงมีค่าเป็นศูนย์

บทที่ 6
สรุปผลงานวิจัย

ในบทนี้กล่าวถึงส่วนสุดท้ายที่ได้จากผลงานวิจัยนั่นคือ บทสรุปของผลงานวิจัย รวมทั้งงานวิจัยในอนาคต และบทความวิชาการที่ตีพิมพ์ ซึ่งมีรายละเอียดดังต่อไปนี้

6.1 สรุปผลงานวิจัย

งานวิจัยนี้นำเสนอวิธีการเรียงลำดับการบูรณาการคลาสโดยใช้เทคนิคการตัดส่วนเชิงวัตถุ โดยขั้นตอนการเรียงลำดับการบูรณาการคลาสประกอบด้วยสามขั้นตอนคือ การวิเคราะห์ความสัมพันธ์ระหว่างคลาส การขจัดการเรียกที่ขึ้นต่อกันแบบมีวง และการเรียงลำดับการทดสอบ แต่ละขั้นตอนอธิบายดังต่อไปนี้

1) การวิเคราะห์ความสัมพันธ์ระหว่างคลาส

ในส่วนของการวิเคราะห์ความสัมพันธ์ระหว่างคลาส เริ่มต้นด้วยการรับรหัสต้นฉบับมาเป็นข้อมูลนำเข้า ทำการตัดคลาสที่ไม่เกี่ยวข้องกับเอสซีซีออก จากนั้นจึงตรวจหาการเรียกที่ขึ้นต่อกันแบบมีวงและแบ่งประเภทไซเคิล

2) การขจัดการเรียกที่ขึ้นต่อกันแบบมีวง

หลังจากแบ่งประเภทไซเคิลแล้วจึงทำการตัดส่วนคลาส เพื่อขจัดการเรียกที่ขึ้นต่อกันแบบมีวงตามประเภทไซเคิล ซึ่งแบ่งการตัดส่วนออกเป็นสามประเภทคือ การตัดส่วนแบบไซเคิลเดียว การตัดส่วนแบบไซเคิลที่มีคลาสซ้อนทับกัน และการตัดส่วนแบบไซเคิลที่มีเส้นทางซ้อนทับกัน ซึ่งการตัดส่วนคลาสนี้ใช้เทคนิคการตัดส่วนเชิงวัตถุ

3) การเรียงลำดับการทดสอบ

เมื่อทำการตัดส่วนคลาสในทุกไซเคิลแล้ว จึงทำการเรียงลำดับคลาสสำหรับทำการทดสอบแบบบูรณาการ โดยใช้การเรียงลำดับแบบทอพอโลยีย้อนหลัง

เมื่อได้แนวคิดในงานวิจัยเรียบร้อยแล้ว จึงได้ทำการออกแบบและพัฒนาเครื่องมือที่เหมาะสมเพื่อสนับสนุนแนวคิดดังกล่าว หลังจากพัฒนาเครื่องมือเรียบร้อยแล้ว ผู้วิจัยได้ทำการทดลองเพื่อประเมินผลการทดลองของวิธีการที่นำเสนอ โดยเปรียบเทียบกับวิธีการเรียงลำดับคลาสของ Tai et al. ของ Traon et al. และของ Briand et al. โดยเปรียบเทียบด้วยจำนวนสตับ จำนวนขั้นการทดสอบ จำนวนลักษณะประจำ จำนวนเมท็อด จำนวนบรรทัดของคำสั่ง และความซับซ้อนของลำดับการทดสอบ

ผลลัพธ์ที่ได้จากการทดลองแสดงออกมาว่า วิธีการเรียงลำดับการบูรณาการคลาสโดยใช้เทคนิคการตัดส่วนเชิงวัตถุ มีการใช้สตับน้อยกว่าวิธีการอื่นๆ แต่ถ้าจำนวนสตับและจำนวนคลาสที่ทำการตัดส่วนมีจำนวนเท่ากัน จำนวนลักษณะประจำ จำนวนเมท็อด และจำนวนบรรทัดคำสั่ง จากวิธีนี้จะมีมากกว่าวิธีอื่นๆ เนื่องจากเมท็อดในคลาสที่ทำการตัดส่วนมีการเรียกลักษณะประจำและเมท็อดภายในอื่นๆ ทำให้จำนวนบรรทัดคำสั่งมีมากขึ้นตาม อย่างไรก็ตาม จำนวน

บรรทัดคำสั่งขึ้นอยู่กับจำนวนลักษณะประจำและจำนวนเมท็อด ซึ่งเป็นคัปปลิงระหว่างคลาส สำหรับค่าความซับซ้อนของลำดับการทดสอบนั้นเกิดจากผลรวมของค่าซับซ้อนในการสร้างสตับทั้งหมด ดังนั้นค่าความซับซ้อนของลำดับการทดสอบจึงขึ้นอยู่กับจำนวนสตับเป็นหลัก

สำหรับกลุ่มเอสซีซีใดก็ตามที่มีความสัมพันธ์แบบแอกกรีเกชันซึ่งกันและกัน โดยไม่มีการลักษณะประจำหรือเมท็อดใดเลย การเรียงลำดับคลาสโดยใช้การตัดส่วนเชิงวัตถุนี้จะไม่สามารถทำการเรียงลำดับได้ เนื่องจากการตัดส่วนคลาสจะสามารถทำได้เมื่อความสัมพันธ์ระหว่างคลาสที่มีการเรียกลักษณะประจำหรือเมท็อด สำหรับวิธีการเรียงลำดับคลาสของ Tai et al. และ Briand et al. นั้นไม่สามารถเรียงลำดับคลาสในกรณีนี้ได้เช่นกัน เนื่องจากทั้งสองวิธีนี้จะทำการลบเฉพาะความสัมพันธ์แบบเกี่ยวพันเท่านั้นจึงไม่สามารถลบความสัมพันธ์ในกลุ่มเอสซีซีที่มีแต่ความสัมพันธ์แบบแอกกรีเกชันได้ ในกรณีนี้วิธีการเรียงลำดับคลาสของ Traon et al. เท่านั้นที่สามารถเรียงลำดับคลาสได้ เนื่องจากวิธีของ Traon et al. จะทำการลบความสัมพันธ์ที่มีค่าถ่วงน้ำหนักสูงสุดเท่านั้น โดยไม่สนใจชนิดของความสัมพันธ์ ดังนั้นการเรียงลำดับคลาสในกรณีนี้จึงยังจำเป็นต้องใช้สตับสำหรับการทดสอบแบบบูรณาการอยู่

6.2 ปัญหาและข้อจำกัดของระบบ

1) กลุ่มคลาสที่นำมาทำการเรียงลำดับคลาสต้องอยู่ในเนมสเปซ (Namespace) ใดเนมสเปซหนึ่งเท่านั้น เครื่องมือที่พัฒนาไม่สามารถเรียงลำดับคลาสข้ามเนมสเปซได้

2) ความสัมพันธ์ระหว่างคลาสในกลุ่มเอสซีซีที่ต้องการเรียงลำดับคลาสต้องไม่มีความสัมพันธ์แบบโพลีมอร์ฟิก เนื่องจากความสัมพันธ์ดังกล่าวไม่อยู่ในขอบเขตของงานวิจัย

3) การตัดส่วนคลาสของเครื่องมือที่พัฒนาขึ้นสามารถทำการตัดส่วนได้เฉพาะความสัมพันธ์ระหว่างคลาสที่มีการเรียกลักษณะประจำหรือเมท็อดเท่านั้น ซึ่งถ้าการคัปปลิงระหว่างคลาสโดยไม่มีการเรียกลักษณะประจำหรือเมท็อด จะไม่สามารถทำการตัดส่วนได้

4) จำนวนคลาสของการเรียงลำดับการบูรณาการคลาสที่ได้จากเครื่องมือที่พัฒนาจะมีจำนวนมากตามจำนวนไซเคิลที่ตรวจจับได้

6.3 งานวิจัยในอนาคต

1) กลุ่มคลาสที่มีความสัมพันธ์แบบโพลีมอร์ฟิกนั้นทำให้เกิดความสัมพันธ์แบบพลวัต (Dynamic dependencies) [20, 21] ทำให้เกิดความซับซ้อนในการหาลำดับการทดสอบมากขึ้น ดังนั้นการหาลำดับการทดสอบอาจจะใช้แผนภาพซีเควนซ์ของยูเอ็มแอลมาประกอบการพิจารณา เนื่องจากแผนภาพซีเควนซ์แสดงลำดับการส่งข้อความ (Message sending sequences) และการเรียกในระดับเมท็อด

2) การตัดแบ่งคลาสในกรณีที่คลาสแม่ของความสัมพันธ์เป็นแบบอินเฮอร์ริแทนซ์นั้น เราไม่สามารถใช้การตัดส่วนแบบสถิต (Static slicing) ได้เพราะคุณสมบัติของคลาสแม่สามารถสืบทอดไปถึงคลาสลูกต่อไปอีกหลายทอดได้ ดังนั้นการตัดส่วนในกรณีนี้อาจจะต้องใช้การตัด

ส่วนแบบพลวัต (Dynamic slicing) โดยเรียกโปรแกรมให้ทำงานก่อนแล้วสังเกตว่าส่วนของโปรแกรมใดบ้างที่เกี่ยวข้องกัน จากนั้นจึงทำการตัดส่วน

3) เนื่องจากงานวิจัยนี้ใช้การตัดส่วนกับคลาสบางคลาสเพื่อขจัดการเรียกที่ขึ้นต่อกันแบบมีวงเท่านั้น ทำให้เกิดการทดสอบแบบขนาน (Parallel testing) เฉพาะคลาสที่ถูกตัดส่วน ซึ่งถ้านำการตัดส่วนเชิงวัตถุกับทุกคลาสเพื่อเรียงลำดับคลาสสำหรับการทดสอบแบบขนาน จำนวนขั้นการทดสอบที่ได้มีความน่าจะเป็นที่จะลดลง

4) หลังจากได้ลำดับการทดสอบคลาสแล้วสามารถพัฒนาในส่วนของการสร้างกรณีทดสอบการบูรณาการต่อได้

6.4 บทความวิชาการที่ตีพิมพ์

ในการวิจัยนี้ ผู้วิจัยผลงานวิชาการร่วมกับคณะผู้วิจัย เป็นบทความวิชาระดับนานาชาติ รวมเป็น 2 บทความ (แสดงใน ภาคผนวก ง.) ได้แก่

1) บทความวิชาการเรื่อง"Finding a Test Order using Object-Oriented Slicing Technique" ซึ่งได้รับการคัดเลือกเพื่อนำเสนอและตีพิมพ์ในงาน "The 14th Asia-Pacific Software Engineering Conference: APSEC 2007" ระหว่างวันที่ 5 – 7 ธันวาคม 2550 ณ Midland Hall เมืองนาโกย่า ประเทศญี่ปุ่น

2) บทความวิชาการเรื่อง "Class Ordering Tool – A Tool for Class Ordering in Integration Testing" ซึ่งได้รับการคัดเลือกเพื่อนำเสนอและตีพิมพ์ในงาน "The 2008 International Conference on Advanced Computer Theory and Engineering ICACTE 2008" ระหว่างวันที่ 20 – 22 ธันวาคม 2551 ณ โรงแรมป่าตอง บีช จังหวัดภูเก็ต ประเทศไทย

รายการอ้างอิง

[1] R. V. Binder. Testing Object-Oriented Systems—Models, Pattern, and Tools. Addison-Wesley, 1999.

[2] K.-C. Tai, F. J. Daniels. Test Order for Inter-Class Integration Testing of Object-Oriented Software. Proceedings of the 21th Annual International Computer Software and Applications Conference (COMPSAC'97), 1997.

[3] Y. Le Traon, T. Jéron. J.-M. Jézéquel, and P. Morel. Efficient Object-Oriented Integration and Regression Testing. IEEE Transactions on Reliability, 2000.

[4] L.C Briand, Y. Labiche and Y. Wang. Revisiting strategies for ordering class integration testing in the presence of dependency cycle. Proceedings of the International Symposium on Software Reliability and Engineering (ISSRE01)., 2001.

[5] Douglas B. West. Introduction to Graph Theory. 2nd Edition, Prentice Hall, 2001.

[6] Paul C. Jorgensen. Software Testing A Craftsman's Approach. 2nd Edition, CRC Press, 2002.

[7] R. Tarjan. Depth-first search and linear graph algorithms. SIAM J. Computing, 1972.

[8] Z. Chen, B. Xu. Slicing Object-Oriented Java Programs, ACM SIGPLAN, 2001.

[9] D. Liang, M. J. Harrold. Slicing Objects Using System Dependence Graph. Proceedings of the international Conference on Software Maintenance, 1998.

[10] A. Abdurazik and J. Offutt. Using Coupling-Based Weights for the Class Integration and Test Order Problem. The Computer Journal Advance Access., 2007.

[11] L.C Briand, J. Feng and Y. Labiche. Using Genetic Algorithms and Coupling Measures to Devise Optimal Integration Test Orders. Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering., 2002.

[12] W. McLendon III, B. Hendrickson, S. J. Plimpton, L. Rauchwerger. Finding strongly connected components in distributed graphs. Journal of Parallel and Distributed Computing., Volume 65, Issue 8, pp. 901-910, 2005.

[13] Denis Erchoff. A handwritten CSharp parser. Available from: http://www.codeplex.com/csparser, 2007.

[14] Standard ECMA-334, C# Language Specification. Available from: http://www.ecma-international.org/ publications/standards/ecma-334.htm, December 2001.

[15] L. Nachmanson, G. Robertson, B. Lee. Drawing graphs with GLEE. Lecture Notes

in Computer Science, Springer-Verlag Berlin Heidelberg, LNCS 4875, pp. 389–394, 2007.

[16] Microsoft Research, GLEE, version 1.0. Available from: http://research.microsoft.com/research/downloads/Details/c927728f-8872-4826-80ee-ecb842d10371/ Details.aspx, 2006.

[17] L.C Briand, Y. Labiche and Y. Wang. An Investigation of Graph-Based Class Integration Test Order Strategies. IEEE Transactions on Software Engineering. vol. 29, no. 7, pp. 594-607, July 2003.

[18] Uggi3D - A .Net 3D Engine. Available from: http://sourceforge.net/projects/uggi3d/., 2006.

[19] Chris J. Friesen. Triangulated Surface Drawing Utility. Available from: http://sourceforge.net/projects/triangles/., 2003.

[20] Y. Labiche, P. Thévenod-Fosse, H. Waeselynck and M.-H. Durand. Testing Levels for Object-Oriented Software. Proceedings of the International Conference on Software Engineering (ICSE00)., 2000.

[21] Q. Chen and X. Li. An Order-Assigned Strategy of Classes Integration Testing Based on Test Level. Proceedings of the 8th International Conference on Computer Supported Cooperative Work in Design., 2003.

ภาคผนวก

ภาคผนวก ก
คลาสของเครื่องมือเรียงลำดับการบูรณาการคลาส

จากแผนภาพคลาสของเครื่องมือเรียงลำดับการบูรณาการคลาส ในบทที่ 4 รูปที่ 4.5 คลาสซึ่งแสดงในแผนภาพคลาสมีรายละเอียดแสดงดังตารางต่อไปนี้

ตารางที่ ก.1 คลาส COTool

Class	COTool
Description	เป็นคลาสสำหรับควบคุมส่วนต่อประสานผู้ใช้ทั้งหมดของเครื่องมือ
Attribute	
- ColumnDataType	เป็น string[] สำหรับเก็บไทป์ของสดมภ์ข้อมูลในการสร้าง DataTable allNodeGlobal
- ColumnDataName	เป็น string[] สำหรับเก็บชื่อของสดมภ์ข้อมูลในการสร้าง DataTable allNodeGlobal
- allNodeGlobal	เป็น DataTable สำหรับเก็บคลาสและความสัมพันธ์ระหว่างคลาสในรูปแบบของตาราง
- levelIDListGlobal	เป็น ArrayList ที่เก็บหมายเลขคลาสของลำดับขั้นในการเรียงคลาส
- sliceListGen	เป็น List สำหรับเก็บคลาสหลังการตัดส่วน เพื่อสร้างไฟล์
- isOpenFile	เป็น bool สำหรับเช็คเงื่อนไขการเปิดไฟล์
Method	
- viewCycle	ทำหน้าที่วาดกราฟแสดงคลาสสัมพันธ์ระหว่างคลาสก่อนทำการตัดส่วนคลาส
- viewgraph	ทำหน้าที่วาดกราฟแสดงคลาสสัมพันธ์ระหว่างคลาสหลังทำการตัดส่วนคลาส
- folderToolStripMenuItem_Click	เป็น window action หลังผู้ใช้กดเมนู Open Folder ทำหน้าที่เลือกโฟลเดอร์ที่บรรจุรหัสต้นฉบับเพื่อเรียงคลาส
- saveToolStripMenuItem_Click	เป็น window action หลังผู้ใช้กดเมนู Generate Slice ทำหน้าที่สร้างคลาสหลังการตัดส่วน
- exitToolStripMenuItem_Click	เป็น window action หลังผู้ใช้กดเมนู Exit ทำหน้าที่ปิดเครื่องมือ
External Relation	มีความสัมพันธ์ aggregation เรียกไปยังคลาส ClassHandler, FileGenerator, TOVisualizer
StereoType	Boundary Class

ตารางที่ ก.2 คลาส ClassHandler

Class	ClassHandler
Description	เป็นคลาสหลักที่ควบคุมการทำงานในการเรียงลำดับคลาสของเครื่องมือนี้
Attribute	
- sliceList	เป็น List ที่เก็บคลาสหลังการตัดส่วนสำหรับคลาส COTool เรียกใช้
- testOrderOutLine	เป็น ArrayList ที่เก็บกลุ่มคลาสหลังการตรวจจับเอสซีซี
Method	
- getTestOrder	ทำหน้าที่เรียงลำดับคลาสโดยมีการทำงานหลักๆ คือ 1. ตรวจหาและวิเคราะห์ความสัมพันธ์ระหว่างคลาส 2. ตรวจจับการเรียกที่ขึ้นต่อกันแบบมีวง 3. ขจัดการเรียกที่ขึ้นต่อกันแบบมีวง 4. เรียงลำดับคลาส
- isCycle	ทำหน้าที่ตรวจสอบกลุ่ม ArrayList ใน testOrderOutLine ว่ามีการเรียกที่ขึ้นต่อกันแบบมีวงหรือไม่ มีรีเทิร์นไทป์เป็นบูลีน
- breakCycle	ทำหน้าที่แบ่งประเภทของกลุ่มคลาสที่ได้รับว่าเป็นไซเคิลประเภทไหนโดยเรียกคลาส SCCIdentifier และทำการตัดส่วนคลาสโดยเรียกคลาส ClassSlicer
External Relation	มีความสัมพันธ์association เรียกไปยังคลาส RelationshipDetector , SCCDetector, SCCIdentifier, ClassSlicer, ClassConnector, ClassSorter
StereoType	Control Class

ตารางที่ ก.3 คลาส RelationshipDetector

Class	RelationshipDetector
Description	เป็นคลาสสำหรับตรวจสอบความสัมพันธ์ระหว่างคลาส โดยรับรหัสต้นฉบับเข้ามาและทำการตรวจสอบความสัมพันธ์ของคลาสทุกคลาส
Attribute	
-	-
Method	
- findEdge	ทำหน้าที่ตรวจสอบความสัมพันธ์และมีรีเทิร์นไทป์เป็น DataTable ซึ่งเก็บความสัมพันธ์ของคลาสทุกคลาส
- checkContainInFile	ทำหน้าที่ตรวจสอบหาคลาสปลายทางในรหัสต้นฉบับของคลาสต้นทางโดยการแจงส่วน ซึ่งมีเรียกเมท็อดภายในต่างๆ ดังนี้ 1. checkInherit 2. checkProperties 3. checkFields

	4. checkConstructor 5. checkMethod
- checkInherit	ทำหน้าที่หาความสัมพันธ์แบบอินเฮอร์ริแทนซ์ระหว่างคลาสต้นทางและคลาสปลายทางในรหัสต้นฉบับ
- checkProperties	ทำหน้าที่ตรวจสอบหาไทป์ของคลาสปลายทางในการประกาศ Properties
- checkFields	ทำหน้าที่ตรวจสอบหาไทป์ของคลาสปลายทางในการประกาศ Fields
- checkConstructor	ทำหน้าที่ตรวจสอบการเรียกคลาสปลายทางในข้อความสั่ง (Statement) ซึ่งอยู่ภายใน Constructor
- checkMethod	ทำหน้าที่ตรวจสอบการเรียกคลาสปลายทางในข้อความสั่งซึ่งอยู่ภายในเมท็อด
External Relation	มีความสัมพันธ์ association เรียกไปยังคลาส ClassChecker
StereoType	Control Class

ตารางที่ ก.4 คลาส ClassChecker

Class	ClassChecker
Description	เป็นคลาสสำหรับแจงส่วนรหัสต้นฉบับที่รับเข้ามา และทำการตรวจสอบความสัมพันธ์ระหว่างคลาสของคลาสต้นทางและคลาสปลายทาง คลาสนี้ทำการเรียกคลังโปรแกรม (Library) ของตัวแจงส่วนซีชาร์ป [13] เพื่อช่วยในการแจงส่วน
Attribute	
- destClsType	เป็น string สำหรับเก็บชื่อไทป์ของคลาสปลายทาง
- allAMStrDest	เป็นคลาสสำหรับเก็บ List ของลักษณะประจำและเมท็อดของคลาสปลายทางที่ถูกคลาสต้นทางเรียก
- methodCallList	เป็น List ของเมท็อดของคลาสปลายทางที่ถูกคลาสต้นทางเรียก โดยรวมถึงเมท็อดภายในที่เกี่ยวข้องกับเมท็อดดังกล่าว
- checkList	เป็น List ของเมท็อดสำหรับตรวจสอบเงื่อนไขในการตัดส่วนคลาส
Method	
- checkSrcClass	ทำหน้าที่ตรวจสอบหาลักษณะประจำและเมท็อดของคลาสปลายทางที่ถูกเรียกในคลาสต้นทาง
- canSliceDestClass	ทำหน้าที่ตรวจสอบเมท็อดของคลาสปลายทาง โดยตรวจสอบเฉพาะเมท็อดที่ถูกคลาสต้นทางเรียก ว่ายังมีการเรียกคลาสอื่นๆ ในไซเคิลหรือไม่ ซึ่งถ้าไม่มีการเรียกคลาสอื่นๆ จะสามารถทำการตัดส่วนได้

	โดยมีรีเทิร์นไทป์เป็นบูลีน
- sliceClass	ทำหน้าที่ตัดส่วนคลาสเฉพาะลักษณะประจำและเมท็อดที่เกี่ยวข้องกับการเรียกของคลาสต้นทาง
- ParseFile	ทำหน้าที่แจงส่วนรหัสต้นฉบับที่รับเข้ามา โดยแปลงให้อยู่ในรูป ClassNode
- getPubAttrMethod	ทำหน้าที่เก็บลักษณะประจำและเมท็อดของคลาสที่ประกาศเป็น Public จาก ClassNode
- getAllAttrMethod	ทำหน้าที่เก็บทุกๆ ลักษณะประจำและเมท็อดของคลาส จาก ClassNode
- findRelatedMethod	ทำหน้าที่หาเมท็อดภายในที่เกี่ยวข้องกับเมท็อดใน allAMStrDest โดยเพิ่มเมท็อดที่เกี่ยวข้องทั้งหมดใน methodCallList
- checkMethod	ทำหน้าที่ตรวจสอบการประกาศไทป์ของคลาสปลายทางในเมท็อด โดยตรวจสอบใน StatementBlock
- checkAccExp	ทำหน้าที่ตรวจสอบการเข้าถึงนิพจน์ (Expression) ว่าในนิพจน์นั้นๆ มีการเข้าถึงลักษณะประจำและเมท็อดใดบ้าง
- checkExp	ทำหน้าที่ตรวจสอบนิพจน์จากข้อความสั่งที่ได้รับว่าเป็นนิพจน์ประเภทใด
- checkBlock	ทำหน้าที่ตรวจสอบกลุ่มของแต่ละข้อความสั่งใน StatementBlock และทำการตรวจสอบขอบเขตของข้อความสั่งในกลุ่ม
- checkStatement	ทำหน้าที่แบ่งประเภทข้อความสั่งว่าเป็นชนิดไหนและทำการตรวจสอบแต่ละประเภทของข้อความสั่งนั้น
- clearFieldValue	ทำหน้าที่ลบทุกค่าในลักษณะประจำภายในคลาส ก่อนการเปลี่ยนคลาสต้นทางและคลาสปลายทางใหม่
External Relation	มีความสัมพันธ์ association เรียกไปยังคลังโปรแกรม CMicroParser
StereoType	Control Class

ตารางที่ ก.5 คลาส SCCDetector

Class	SCCDetector
Description	เป็นคลาสสำหรับตรวจหาการเรียกที่ขึ้นต่อกันแบบมีวง โดยใช้ทาร์จันอัลกอริทึมในการตรวจหา
Attribute	
- unVistedList	เป็น ArrayList ที่เก็บคลาสทุกคลาสที่ยังไม่ได้ตรวจสอบ ซึ่งคลาสที่ถูกเดินผ่านโดยอัลกอริทึมแล้วจะถูกลบออก

- max_dfs	เป็น int สำหรับนับความลึกในการค้นหาตามแนวลึก
- s_stack	เป็น Stack ในการเก็บคลาสที่ถูกเดินผ่านโดยอัลกอริทึม โดยเก็บค่าหมายเลขของคลาส
- testOrderOutLine	เป็น ArrayList ที่เก็บกลุ่มคลาสที่มีการเรียกที่ขึ้นต่อกันแบบมีวง
Method	
- detectSCC	ทำหน้าที่หากลุ่มเอสซีซีจากกลุ่มคลาสทั้งหมด โดยจะทำการตัดคลาสที่ไม่เกี่ยวข้องกับเอสซีซีออกก่อน และเรียกทาร์จันเมท็อดในการค้นหากลุ่มเอสซีซี จากนั้นจึงตรวจสอบความสัมพันธ์ของคลาสแบบกึ่งวิถี เมท็อดนี้มีรีเทิร์นไทป์เป็น ArrayList
- trimClass	ทำหน้าที่ตัดคลาสที่ไม่เกี่ยวข้องกับเอสซีซีออก
- tarjan	ทำหน้าที่หากลุ่มคลาสที่มีการเรียกที่ขึ้นต่อกันแบบมีวง โดยใช้การค้นหาตามแนวลึก
- checkSemiPath	ทำหน้าที่ตรวจสอบหาความสัมพันธ์ของคลาสแบบกึ่งวิถี
- getAllStartNode	ทำหน้าที่หาคลาสที่มีการเรียกคลาสอื่นและไม่ถูกคลาสใดๆเรียกเพื่อเป็นคลาสเริ่มต้นในการค้นหาโดยทาร์จันเมท็อด
External Relation	-
StereoType	Control Class

ตารางที่ ก.6 คลาส SCCIdentifier

Class	SCCIdentifier
Description	เป็นคลาสสำหรับแบ่งประเภทของกลุ่มเอสซีซี ซึ่งการพิจารณาประเภทของเอสซีซีนั้น จะพิจารณาเฉพาะกลุ่มคลาสที่ได้รับ โดยดูความสัมพันธ์ระหว่างคลาสในกลุ่มนั้น ซึ่งประเภทของเอสซีซีมี 3 แบบคือ แบบหนึ่งไซเคิล แบบไซเคิลที่มีคลาสซ้อนทับกัน และแบบไซเคิลที่มีเส้นทางซ้อนทับกัน
Attribute	
-	-
Method	
- classifyCyle	ทำหน้าที่แบ่งประเภทกลุ่มเอสซีซี ซึ่งเป็น ArrayList ที่รับเข้ามา โดยตรวจสอบความสัมพันธ์ระหว่างคลาสจาก DataTable ที่รับเข้ามาเช่นกัน มีรีเทิร์นไทป์เป็น SCCStruct ซึ่งเก็บค่าประเภทของไซเคิล
External Relation	-

StereoType	Control Class

ตารางที่ ก.7 คลาส ClassSlicer

Class	ClassSlicer
Description	เป็นคลาสสำหรับตัดส่วนคลาสตามประเภทของไซเคิล โดยรับประเภทของเอสซีซีและ DataTable ที่เก็บความสัมพันธ์ระหว่างคลาสเป็นข้อมูลนำเข้า
Attribute	
- visitedList	เป็น List ของคลาส สำหรับตรวจสอบคลาสที่ถูกเดินผ่านแล้ว ซึ่งมีการเรียกใช้ในเมท็อด splitToOneCycle
- dtPO	เป็น DataTable สำหรับเก็บไซเคิลเดียวจากการแบ่งไซเคิลที่มีการซ้อนทับในเมท็อด splitToOneCycle
- startID	เป็น string ซึ่งเก็บคลาสเริ่มต้นในการแบ่งไซเคิลที่มีการซ้อนทับ
- r_id	เป็น int สำหรับเป็นตัวนับเพื่อเพิ่มแถวใน DataTable
- dtList	เป็น List ซึ่งเก็บ DataTable ที่บรรจุคลาสประเภทไซเคิลเดียว ซึ่งได้จากการแบ่งไซเคิลที่มีการซ้อนทับ
Method	
- sliceOneCycle	ทำหน้าที่ตัดส่วนคลาสประเภทไซเคิลเดียว โดยทำการเรียกคลาส ClassChecker เพื่อช่วยในการแจงส่วนคลาสในการหาความสัมพันธ์ระหว่างคลาสที่สามารถตัดส่วนได้
- splitToOneCycle	ทำหน้าที่แบ่งไซเคิลที่มีการซ้อนทับกันออกเป็นไซเคิลเดียว โดยใช้การค้นหาตามแนวลึก
- sliceClassOverlap	ทำหน้าที่ตัดส่วนคลาสประเภทไซเคิลที่มีคลาสซ้อนทับกัน โดยทำการเรียกเมท็อด splitToOneCycle ก่อน เพื่อแยกออกมาเป็นไซเคิลเดียว จากนั้นจึงเรียกเมท็อด sliceOneCycle
- slicePathOverlap	ทำหน้าที่ตัดส่วนคลาสประเภทไซเคิลที่มีเส้นทางซ้อนทับกัน โดยทำการเรียกเมท็อด splitToOneCycle ก่อน เพื่อแยกออกมาเป็นไซเคิลเดียว และทำการหาไซเคิลที่มีจำนวนคลาสน้อยที่สุดก่อน จากนั้นจึงเรียกเมท็อด sliceOneCycle
- addDTList	ทำหน้าที่ตรวจสอบว่าไซเคิลที่ตรวจจับได้นั้น ได้ถูกเพิ่มใน dtList แล้วหรือไม่ ถ้าไม่ปรากฏใน dtList จึงทำการเพิ่มไซเคิลนั้น แต่ถ้ามีอยู่ใน dtList แล้ว ก็ไม่จำเป็นต้องเพิ่มลงไป
External Relation	มีความสัมพันธ์ association เรียกไปยังคลาส ClassChecker
StereoType	Control Class

ตารางที่ ก.8 คลาส ClassConnector

Class	ClassConnector
Description	เป็นคลาสสำหรับสร้างความสัมพันธ์ระหว่างคลาสใหม่หลังจากทำการตัดส่วน โดยความสัมพันธ์ระหว่างคลาสใหม่นี้จะเป็นความสัมพันธ์ที่ปราศจากกลุ่มเอสซีซี
Attribute	
-	-
Method	
- connectCycle	ทำหน้าที่เปลี่ยนโครงสร้างความสัมพันธ์ใน DataTable ใหม่หลังจากตัดส่วน โดยเพิ่มคลาสที่ถูกทำการตัดส่วนและเปลี่ยนการเรียกคลาสโดยให้คลาสต้นทางที่เรียกไปยังคลาสที่ถูกตัดส่วนนั้น เปลี่ยนให้เรียกไปที่คลาสใหม่ที่เพิ่มขึ้นมา
External Relation	-
StereoType	Control Class

ตารางที่ ก.9 คลาส ClassSorter

Class	ClassSorter
Description	เป็นคลาสสำหรับเรียงลำดับคลาสให้อยู่ในรูปของขั้นการทดสอบ โดยใช้การเรียงลำดับแบบทอพอโลยีย้อนหลัง
Attribute	
- levelList	เป็น ArrayList ที่เก็บชื่อคลาสในแต่ละขั้น
- levelIDList	เป็น ArrayList ที่เก็บหมายเลขคลาสในแต่ละขั้น
- sortedList	เป็น ArrayList ที่เก็บคลาสหลังการเรียงลำดับคลาสแล้วในแต่ละขั้น
Method	
- orderClass	ทำหน้าที่รวมกลุ่มคลาสในขั้นเดียวกัน โดยเริ่มต้นจากกลุ่มคลาสที่ไม่มีการเรียกไปยังคลาสอื่นๆ ก่อน จากนั้นจึงหาคลาสที่เรียกคลาสในกลุ่มแรกนั้นต่อไปเป็นลำดับขั้น
- findUpperClass	ทำหน้าที่หาคลาสในขั้นแรกสุดที่ควรถูกทดสอบก่อน ซึ่งคลาสในขั้นนี้คือกลุ่มคลาสที่ไม่มีการเรียกไปยังคลาสอื่นๆ
- addClass	ทำหน้าที่เพิ่มคลาสในขั้นถัดไปหลังจากเรียงคลาสในขั้นแรกเสร็จ
External Relation	-

StereoType	Control Class

ตารางที่ ก.10 คลาส TOVisualizer

Class	TOVisualizer
Description	เป็นคลาสสำหรับสร้างมโนภาพของลำดับการทดสอบ คลาสนี้ทำการเรียกคลังโปรแกรมของจีแอลอีอี [16] เพื่อช่วยในการวาดกราฟ
Attribute	
-	-
Method	
- shiftPos	ทำหน้าที่เลื่อนตำแหน่ง x ของโหนดถัดไป ในกรณีตำแหน่ง x ของโหนดในขั้นเดียวกันมีค่าที่ใกล้กันมากจนเกิดการซ้อนทับกัน
- drawTestOrder	ทำหน้าที่กำหนดตำแหน่ง x, y ในการวาดโหนดและเส้นเชื่อมของกราฟ
External Relation	มีความสัมพันธ์ association เรียกไปยังคลังโปรแกรม GLEE
StereoType	Control Class

ตารางที่ ก.11 คลาส FileGenerator

Class	FileGenerator
Description	เป็นคลาสสำหรับสร้างรหัสต้นฉบับของคลาสหลังจากการตัดส่วน
Attribute	
-	-
Method	
- writeFile	ทำหน้าที่สร้างไฟล์รหัสต้นฉบับของคลาส โดยสร้างไว้ในโฟลเดอร์ที่ผู้ใช้เลือกจากเมนู Generate Slice
External Relation	-
StereoType	Control Class

ตารางที่ ก.12 คลังโปรแกรม CMicroParser

Library	CMicroParser
Description	เป็นคลังโปรแกรมสำหรับแจงส่วนคลาสจากรหัสต้นฉบับ เพื่อช่วยในการวิเคราะห์ความสัมพันธ์ระหว่างคลาสและการตัดส่วนคลาส
Class	

- Lexer	เป็นคลาสสำหรับแปลงตัวอักษรในรหัสต้นฉบับเป็นโทเค็น (Token)
- TokenCollection	เป็นคลาสสำหรับเก็บกลุ่มโทเค็นของรหัสต้นฉบับหลังการแปลง
- Parser	เป็นคลาสสำหรับแจงส่วนกลุ่มของโทเค็นให้อยู่ในรูปของ CompilationUnitNode ตามไวยากรณ์ภาษาซีชาร์ป
- CompilationUnitNode	เป็นคลาสซึ่งเก็บรหัสต้นฉบับหลังการแจงส่วนในรูปของโหนด
- ClassNode	เป็นคลาสซึ่งเก็บโครงสร้างและเนื้อหาในคลาสจากรหัสต้นฉบับ
- FieldNode	เป็นคลาสซึ่งเก็บโครงสร้างและเนื้อหาในฟิลด์จากรหัสต้นฉบับ
- PropertyNode	เป็นคลาสซึ่งเก็บโครงสร้างและเนื้อหาที่เป็น Property ของคลาสจากรหัสต้นฉบับ
- ConstructorNode	เป็นคลาสซึ่งเก็บ BlockStatement ในตัวสร้างของคลาส
- MethodNode	เป็นคลาสซึ่งเก็บ BlockStatement ในเมท็อดของคลาส
- BlockStatement	เป็นคลาสซึ่งเก็บข้อความสั่งในรูปของ StatementNode
- StatementNode	เป็นคลาสซึ่งเก็บข้อความสั่งและนิพจน์ ซึ่งชนิดของข้อความสั่งนั้นสามารถมีค่าเป็นข้อความสั่งเฉพาะย่อยออกไปได้อีก เช่น IfStatement, ForStatement เป็นต้น
- ExpressionStatement	เป็นคลาสซึ่งเก็บนิพจน์ ซึ่งชนิดของนิพจน์นั้นสามารถมีค่าเป็นนิพจน์เฉพาะย่อยออกไปได้อีก เช่น MemberAccessExpression, InvocationExpression เป็นต้น
StereoType	Library File

ตารางที่ ก.13 คลังโปรแกรม GLEE

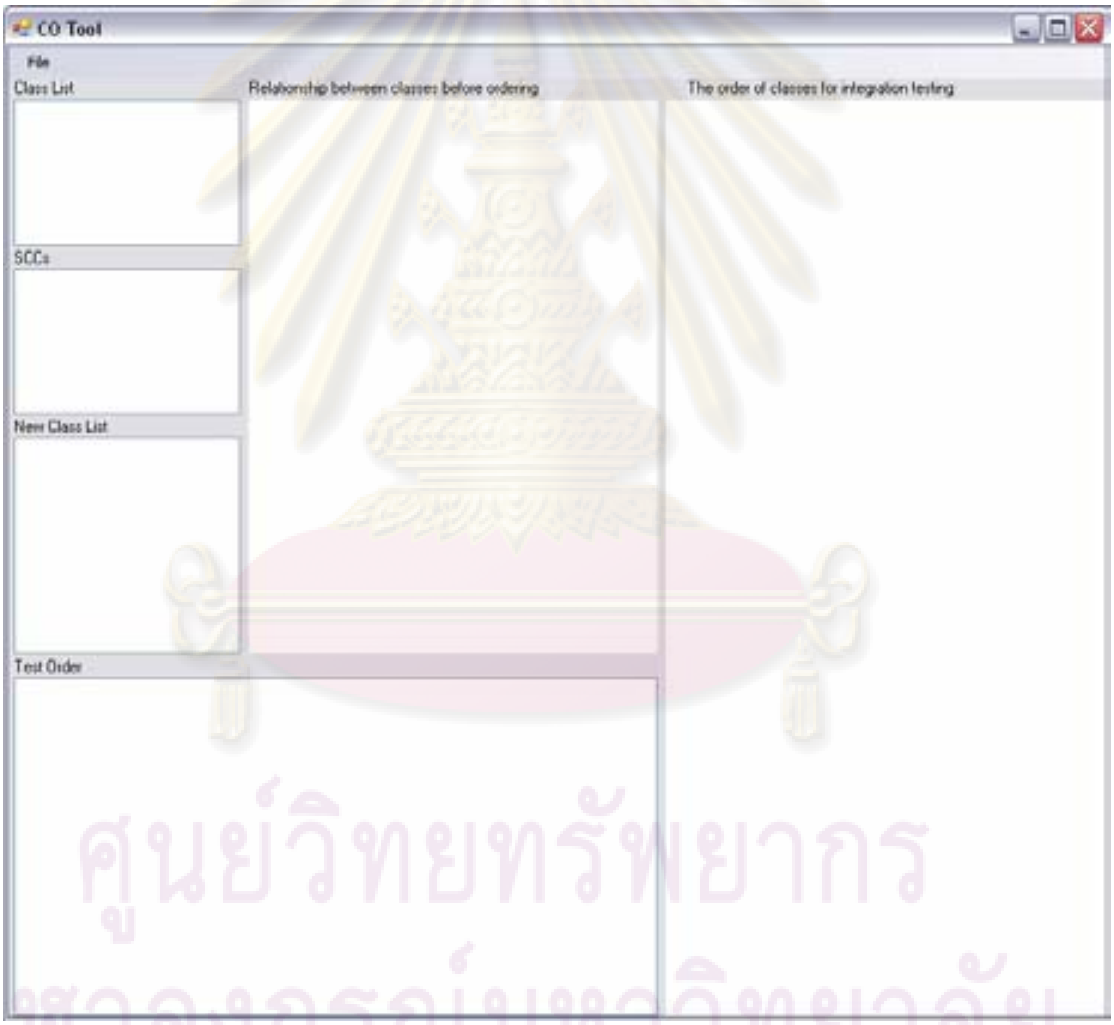
Library	GLEE
Description	เป็นคลังโปรแกรมสำหรับการวาดกราฟและกำหนดตำแหน่งที่เหมาะสมของโหนดและเส้นเชื่อม
Class	
- GViewer	เป็นคลาสสำหรับแสดงภาพของโหนดและเส้นเชื่อมในกราฟ
- Graph	เป็นคลาสสำหรับสร้างกราฟ โดยกำหนดตำแหน่ง x, y ที่เหมาะสมของโหนดและเส้นเชื่อมจากความสัมพันธ์ของโหนดต้นทางและโหนดปลายทาง
- Node	เป็นคลาสซึ่งเก็บตำแหน่ง x, y ของโหนดหลังจากกำหนดตำแหน่งในกราฟ
- Edge	เป็นคลาสซึ่งเก็บตำแหน่ง x, y ของเส้นหลังจากกำหนดตำแหน่งในกราฟ
StereoType	Library File

ภาคผนวก ข
คู่มือการใช้งานเครื่องมือ

เครื่องมือเรียงลำดับการบูรณาการคลาสโดยใช้เทคนิคการตัดส่วนเชิงวัตถุ เป็นเครื่องมือเรียงลำดับคลาสสำหรับการทดสอบแบบบูรณาการสำหรับโปรแกรมภาษาซีชาร์ป โดยมีวิธีการใช้งานเครื่องมือดังนี้

การเริ่มต้นการใช้งานเครื่องมือ

เรียกใช้เครื่องมือเรียงลำดับการบูรณาการคลาส เมื่อโปรแกรมพร้อมใช้งาน จะปรากฏหน้าจอดังรูปที่ ข.1



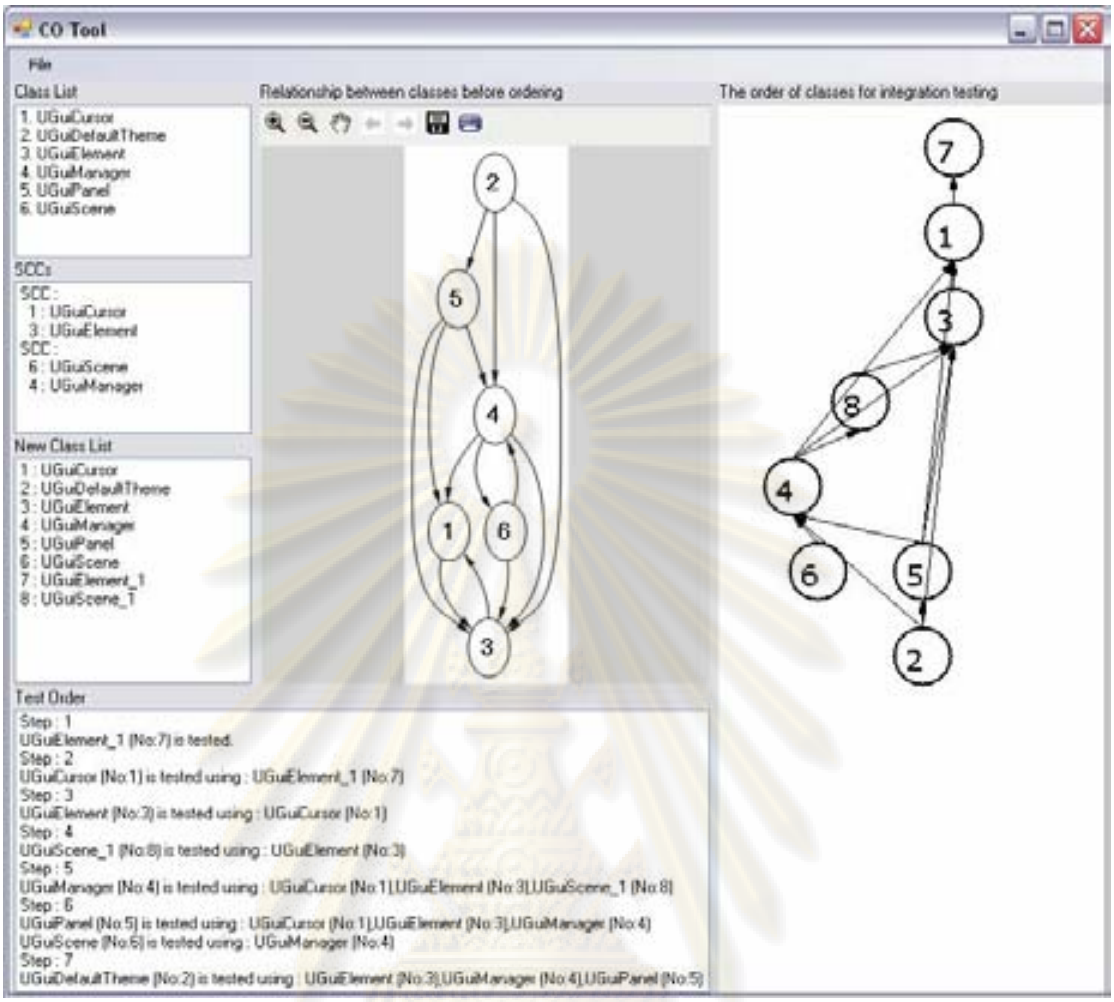
รูปที่ ข.1 หน้าจอเริ่มต้นการทำงานของเครื่องมือ

การเรียงลำดับการบูรณาการคลาส

ในการเรียงลำดับคลาสต้องเลือกโฟลเดอร์ที่บรรจุไฟล์รหัสต้นฉบับที่ต้องการเรียงลำดับการทดสอบ โดยเลือกเมนู File -> Open -> Folder จากนั้นจึงปรากฏหน้าจอดังรูปที่ ข.2

รูปที่ ข.2 หน้าจอแสดงการเลือกโฟลเดอร์ซึ่งบรรจุรหัสต้นฉบับ

เมื่อเลือกโฟลเดอร์ซึ่งบรรจุรหัสต้นฉบับที่ต้องการเรียงลำดับการทดสอบแล้ว เครื่องมือจะเรียงลำดับโดยทำการหาคลาสในไซเคิลที่สามารถทำการตัดส่วนได้มาทำการตัดส่วนคลาส จากนั้นจึงเรียงลำดับคลาสสำหรับการทดสอบแบบบูรณาการ โดยผลลัพธ์แสดงดังรูปที่ ข.3

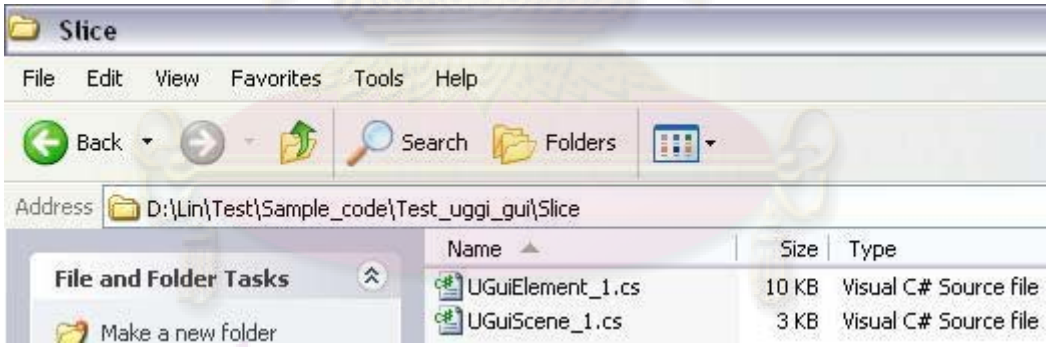


รูปที่ ข.3 หน้าจอแสดงผลการเรียงลำดับการทดสอบ

ลำดับคลาสสำหรับการทดสอบแบบบูรณาการแสดงในหน้าต่างลำดับการทดสอบของหน้าจอในรูปที่ ข.3 หลังจากได้ลำดับการทดสอบแล้ว การสร้างรหัสต้นฉบับหลังจากตัดส่วนทำได้โดยเลือกเมนู File -> Generate Slice จากนั้นจึงปรากฏหน้าจอดังรูปที่ ข.4

รูปที่ ข.4 หน้าจอแสดงการเลือกโฟลเดอร์สำหรับรหัสต้นฉบับของคลาสหลังการตัดส่วน

จากนั้นจึงทำการเลือกโฟลเดอร์สำหรับรหัสต้นฉบับของคลาสหลังการตัดส่วน ซึ่งรหัสต้นฉบับเหล่านี้ถูกสร้างสำหรับคลาสที่ต้องการเรียกในการทดสอบแบบบูรณาการ โดยรหัสต้นฉบับของคลาสหลังการตัดส่วนในโฟลเดอร์แสดงดังรูปที่ ข.5



รูปที่ ข.5 หน้าต่างแสดงโฟลเดอร์ที่เก็บรหัสต้นฉบับที่เป็นข้อมูลนำออก

ภาคผนวก ค
การคำนวณความซับซ้อนของลำดับการทดสอบ

จากตัววัดค่าความซับซ้อนที่ได้อธิบายไว้ในบทที่ 2 หัวข้อ 2.2.6 เนื่องจากค่าถ่วงน้ำหนักมีคุณสมบัติ $W_V + W_M + W_R + W_P = 1$ ดังนั้นจึงกำหนดให้ค่าถ่วงน้ำหนักของ W_V, W_M, W_R, W_P แต่ละค่ามีค่าเท่ากับ 0.25

การคำนวณค่าความซับซ้อนของลำดับการทดสอบของระบบ Uggi3D

จากแผนภาพคลาสของระบบ Uggi3D ในบทที่ 5 รูปที่ 5.3 มีความสัมพันธ์ระหว่างคลาสอยู่ 13 ความสัมพันธ์ เมื่อนำมาสร้างสตับจะมีค่าคัปปลิง แสดงดังตารางที่ ค.1

ตารางที่ ค.1 ค่าคัปปลิงระหว่างคลาสในการสร้างสตับของระบบ Uggi3D

No	Stub	V_d	M_d	R_d	P_d
1	stub(UGuiElement, UGuiCursor)	1	6	0	0
2	stub(UGuiCursor, UGuiElement)	0	0	0	0
3	stub(UGuiManager, UGuiScene)	0	0	0	0
4	stub(UGuiScene, UGuiManager)	1	2	0	3
5	stub(UGuiElement, UGuiDefaultTheme)	4	1	0	0
6	stub(UGuiPanel, UGuiDefaultTheme)	4	0	0	0
7	stub(UGuiManager, UGuiDefaultTheme)	4	4	0	12
8	stub(UGuiElement, UGuiPanel)	4	2	1	2
9	stub(UGuiManager, UGuiPanel)	0	0	0	0
10	stub(UGuiCursor, UGuiPanel)	0	0	0	0
11	stub(UGuiCursor, UGuiManager)	0	1	0	0
12	stub(UGuiElement, UGuiManager)	8	3	1	2
13	stub(UGuiElement, UGuiScene)	5	5	0	4

จากตารางดังกล่าว ค่าที่น้อยที่สุดและค่าที่มากที่สุดของคัปปลิงแต่ละชนิด แสดงดังตารางที่ ค.2

ตารางที่ ค.2 ค่าที่น้อยที่สุดและค่าที่มากที่สุดของคัปปลิงแต่ละชนิดของระบบ Uggi3D

	V_d	M_d	R_d	P_d
Min	0	0	0	0
Max	8	6	1	12

เนื่องจากสตับสำหรับการเรียงลำดับการทดสอบของระบบ Uggi3D จากงานวิจัยของ Tai et al., Traon et al., Briand et al. ต้องการใช้สตับเพียงแค่สตับหมายเลขที่ 1 – 4 เท่านั้น ดังนั้นการหาค่าบรรทัดฐานของสตับจึงมีแค่ 4 สตับ แสดงดังตารางที่ ค.3

ตารางที่ ค.3 ค่าบรรทัดฐานของคัปปลิงแต่ละชนิดของระบบ Uggi3D

No	Stub	$\overline{V}$	$\overline{M}$	$\overline{R}$	$\overline{P}$
1	stub(UGuiElement,UGuiCursor)	0.125	1	0	0
2	stub(UGuiCursor,UGuiElement)	0	0	0	0
3	stub(UGuiManager,UGuiScene)	0	0	0	0
4	stub(UGuiScene , UGuiManager)	0.125	0.333	0	0.25

เมื่อนำค่าจากตารางที่ ค.3 มาคำนวณตามสมการที่ 4 ในบทที่ 2 หัวข้อ 2.2.6 ค่าความซับซ้อนของแต่ละสตับ แสดงดังตารางที่ ค.4

ตารางที่ ค.4 ค่าความซับซ้อนของสตับของระบบ Uggi3D

No	Stub	Complexity
1	stub(UGuiElement, UGuiCursor)	1.503
2	stub(UGuiCursor, UGuiElement)	1
3	stub(UGuiManager, UGuiScene)	1
4	stub(UGuiScene, UGuiManager)	1.217

สตับที่ได้จากการหาลำดับการทดสอบของ Tai et al. และ Briand et al. คือ stub(UGuiElement, UGuiCursor) และ stub(UGuiScene, UGuiManager) ดังนั้นค่าความซับซ้อนของลำดับการทดสอบ แสดงดังตารางที่ ค.5

ตารางที่ ค.5 ค่าความซับซ้อนของลำดับการทดสอบของ Tai et al. และ Briand et al.

Stub	Complexity
stub(UGuiElement, UGuiCursor)	1.503
stub(UGuiScene, UGuiManager)	1.217
Order Complexity	2.72

สตับที่ได้จากการหาลำดับการทดสอบของ Traon et al. คือ stub(UGuiCursor,UGuiElement) และ stub(UGuiManager, UGuiScene) ดังนั้นค่าความซับซ้อนของลำดับการทดสอบ แสดงดังตารางที่ ค.6

ตารางที่ ค.6 ค่าความซับซ้อนของลำดับการทดสอบของ Traon et al.

Stub	Complexity
stub(UGuiCursor, UGuiElement)	1
stub(UGuiManager, UGuiScene)	1
Order Complexity	2

การคำนวณค่าความซับซ้อนของลำดับการทดสอบของระบบ TSDU

จากแผนภาพคลาสของระบบ TSDU ในบทที่ 5 รูปที่ 5.5 มีความสัมพันธ์ระหว่างคลาสอยู่ 14 ความสัมพันธ์ เมื่อนำมาสร้างสตับจะมีค่าคัปปลิง แสดงดังตารางที่ ค.7

ตารางที่ ค.7 ค่าคัปปลิงระหว่างคลาสในการสร้างสตับของระบบ TSDU

No	Stub	V_d	M_d	R_d	P_d
1	stub(MouseListener, GLView)	0	4	0	2
2	stub(FPSCounter, GLView)	1	3	0	1
3	stub(KeyListener, GLView)	0	3	0	2
4	stub(Controller , MainClass)	2	2	0	0
5	stub(Model , MainClass)	1	2	0	0
6	stub(Controller , KeyListener)	0	0	0	0
7	stub(Controller , FPSCounter)	0	0	0	0
8	stub(GLView, Model)	0	2	0	0
9	stub(GLView, MouseListener)	9	5	0	6
10	stub(GLView , Controller)	4	2	1	2
11	stub(GLView , FPSCounter)	1	1	0	0
12	stub(GLView , KeyListener)	5	6	0	4
13	stub(Model , Controller)	0	4	0	0
14	stub(GLView , MainClass)	0	1	0	0

จากตารางดังกล่าว ค่าที่น้อยที่สุดและค่าที่มากที่สุดของคัปปลิงแต่ละชนิด แสดงดังตารางที่ ค.8

ตารางที่ ค.8 ค่าที่น้อยที่สุดและค่าที่มากที่สุดของคัปปลิงแต่ละชนิดของระบบ TSDU

	V_d	M_d	R_d	P_d
Min	0	0	0	0
Max	9	6	3	6

เนื่องจากสตับสำหรับการเรียงลำดับการทดสอบของระบบ TSDU จากงานวิจัยของ Tai et al., Traon et al., Briand et al. ต้องการใช้สตับเพียงแค่สตับหมายเลขที่ 1 – 12 เท่านั้น ดังนั้นการหาค่าบรรทัดฐานของสตับจึงมี 12 สตับ แสดงดังตารางที่ ค.9

ตารางที่ ค.9 ค่าบรรทัดฐานของคัปปลิงแต่ละชนิดของระบบ TSDU

No	Stub	$\overline{V}$	$\overline{M}$	$\overline{R}$	$\overline{P}$
1	stub(MouseListener, GLView)	0	0.667	0	0.333
2	stub(FPSCounter, GLView)	0.111	0.5	0	0.167
3	stub(KeyListener, GLView)	0	0.5	0	0.333
4	stub(Controller , MainClass)	0.222	0.333	0	0
5	stub(Model , MainClass)	0.111	0.333	0	0
6	stub(Controller , KeyListener)	0	0	0	0
7	stub(Controller , FPSCounter)	0	0	0	0
8	stub(GLView, Model)	0	0.333	0	0
9	stub(GLView, MouseListener)	1	0.833	0	1
10	stub(GLView , Controller)	0.444	0.333	0.333	0.333
11	stub(GLView , FPSCounter)	0.111	0.167	0	0
12	stub(GLView , KeyListener)	0.556	1	0	0.667

เมื่อนำค่าจากตารางที่ ค.9 มาคำนวณตามสมการที่ 4 ในบทที่ 2 หัวข้อ 2.2.6 ค่าความซับซ้อนของแต่ละสตับ แสดงดังตารางที่ ค.10

ตารางที่ ค.10 ค่าความซับซ้อนของสตับของระบบ TSDU

No	Stub	Complexity
1	stub(MouseListener, GLView)	1.372678
2	stub(FPSCounter, GLView)	1.269316
3	stub(KeyListener, GLView)	1.300463
4	stub(Controller , MainClass)	1.200308
5	stub(Model , MainClass)	1.175682
6	stub(Controller , KeyListener)	1
7	stub(Controller , FPSCounter)	1
8	stub(GLView, Model)	1.166667
9	stub(GLView, MouseListener)	1.820738
10	stub(GLView , Controller)	1.364302
11	stub(GLView , FPSCounter)	1.100154
12	stub(GLView , KeyListener)	1.662021

สตับที่ได้จากการหาลำดับการทดสอบของ Tai et al. มี 7 สตับ ซึ่งผลรวมค่าความซับซ้อนของสตับแต่ละตัวคือค่าความซับซ้อนของลำดับการทดสอบ แสดงดังตารางที่ ค.11

ตารางที่ ค.11 ค่าความซับซ้อนของลำดับการทดสอบของ Tai et al. ของระบบ TSDU

Stub	Complexity
stub(MouseListener, GLView)	1.372678
stub(FPSCounter, GLView)	1.269316
stub(KeyListener, GLView)	1.300463
stub(Controller , MainClass)	1.200308
stub(Model , MainClass)	1.175682
stub(Controller , KeyListener)	1
stub(Controller , FPSCounter)	1
Order Complexity	8.32

สตับที่ได้จากการหาลำดับการทดสอบของ Traon et al. มี 5 สตับ ซึ่งผลรวมค่าความซับซ้อนของสตับแต่ละตัวคือค่าความซับซ้อนของลำดับการทดสอบ แสดงดังตารางที่ ค.12

ตารางที่ ค.12 ค่าความซับซ้อนของลำดับการทดสอบของ Traon et al. ของระบบ TSDU

Stub	Complexity
stub(GLView , Model)	1.166667
stub(GLView , MouseListener)	1.820738
stub(GLView , Controller)	1.364302
stub(GLView , FPSCounter)	1.100154
stub(GLView , KeyListener)	1.662021
Order Complexity	7.11

สตับที่ได้จากการหาลำดับการทดสอบของ Briand et al. มี 3 สตับ ซึ่งผลรวมค่าความซับซ้อนของสตับแต่ละตัวคือค่าความซับซ้อนของลำดับการทดสอบ แสดงดังตารางที่ ค.13

ตารางที่ ค.13 ค่าความซับซ้อนของลำดับการทดสอบของ Briand et al. ของระบบ TSDU

Stub	Complexity
stub(MouseListener , GLView)	1.372678
stub(FPSCounter , GLView)	1.269316
stub(KeyListener , GLView)	1.300463
Order Complexity	3.94

ภาคผนวก ง
บทความวิชาการที่ตีพิมพ์

ในการวิจัยนี้ ผู้วิจัยมีผลงานวิชาการร่วมกับคณะผู้วิจัย เป็นบทความวิชาระดับนานาชาติ รวมเป็น 2 บทความ ได้แก่

ง.1 บทความวิชาการเรื่อง"Finding a Test Order using Object-Oriented Slicing Technique" ซึ่งได้รับการคัดเลือกเพื่อนำเสนอและตีพิมพ์ในงาน "The 14th Asia-Pacific Software Engineering Conference: APSEC 2007" ระหว่างวันที่ 5 – 7 ธันวาคม 2550 ณ Midland Hall เมืองนาโกย่า ประเทศญี่ปุ่น

ง.2 บทความวิชาการเรื่อง "Class Ordering Tool – A Tool for Class Ordering in Integration Testing" ซึ่งได้รับการคัดเลือกเพื่อนำเสนอและตีพิมพ์ในงาน "The 2008 International Conference on Advanced Computer Theory and Engineering ICACTE 2008" ระหว่างวันที่ 20 – 22 ธันวาคม 2551 ณ โรงแรมป่าตอง บีช จังหวัดภูเก็ต ประเทศไทย

Finding a Test Order using Object-Oriented Slicing Technique

Jutarat Jaroenpiboonkit and Taratip Suwannasart
Software Engineering Laboratory
Department of Computer Engineering, Faculty of Engineering
Chulalongkorn University, Bangkok 10330 Thailand
Jutarat.Ja@student.chula.ac.th, Taratip.S@chula.ac.th

Abstract

For integration testing processes, testers need to find test sequences of classes in order to execute interactions. One major problem of test order is the presence of cyclic dependency calls. Many researchers have proposed techniques to solve this problem by removing relationships to break cycles and then create test stubs. Therefore, one major purpose of integration testing is minimization of test stubs. This paper presents an approach to find test sequences by using object-oriented slicing technique to alleviate using test stubs. The main objective of using object-oriented slicing is to break cycles by slicing classes for partial testing instead of removing relationships for test stub construction. With this approach, the cost of implementing test stubs is decreased.

Keywords: Integration Testing, Test Orders, Object-Oriented Program Slicing

1. Introduction

Integration testing is a search for component faults that cause inter-component failures [1]. One major problem in integrating and testing object-oriented program is to determine the order of classes to be tested, which is important for several reasons [3]. Furthermore, the difficulty of test order is the presence of cyclic dependency calls; that is, when a class calls other classes, there exists a path from a called class is back to the calling class. To solve this problem, many researchers have proposed techniques to remove relationships by performing no cycle graph which is called Directed Acyclic Graph (DAG). Deleted relationship implies that stub construction in order to test source class. Therefore, a main purpose of integration testing is minimization of the effort to create stubs; that is, NP-complete problem [2, 5]. A number of test stubs are a cost factor for integration testing [2].

Although existing test order strategies reduce a number of stubs but in case of stubs are complex to design, it may take long time to create and need knowledge of testers about interaction between stubs and classes, since different return values of stubs affect different states of classes. If stub creation is more complicated, more errors will be occurred [5]. Thus this paper uses object-oriented slicing to find classes that can be separated to be tested instead of stubs. It reduces cost and difficulties of stub creation.

In this paper we propose an approach to find a test order in order to use less stubs by using Test Dependency Graph (TDG) [9] and object-oriented slicing technique [17]. We use TDG to represent relationships among classes because it shows details in level of method calls which can be used to display portion of partial testing, while object-oriented slicing technique is used to break cyclic dependency of classes and to find a test order.

The remainder of this paper is organized as follows: Section 2, we briefly discuss existing integration test order strategies. Section 3 presents the object-oriented slicing technique. Section 4, we explain our proposed strategy and demonstrate the use of object-oriented slicing to find a test order. Section 5 compares our technique to using stubs in other strategies. Finally, conclusion and future work are drawn in Section 6.

2. Related Work

Kung et al. [10] were the first researchers to address the class test order problem. They showed that, when no dependency cycles are present among classes, deriving an integration order is equivalent to performing a topological sorting of classes based on their dependency graph - a well known graph theory

problem. In the presence of dependency cycles, they proposed a strategy of identifying strongly connected components (SCCs) and removing associations until no cycles remain. Kung et al.'s approach removes edge randomly when there is more than one candidate association for cycle breaking. They mentioned that a possible solution would involve the use of the complexity of associations involved in cycles [8].

Tai and Daniels [3] proposed a number of desirable properties for inter-class test order and defined a major and minor level number to sort classes. The assignment of level number has two steps. First, they used inheritance and aggregation relations assigned to each class in major level number. Then, in the same major level number, separating each class to a minor level number based on association relationships only. After that, if cycle occurs in ORD, breaking dependency will be needed to apply before topological sorting. Tai and Daniels chose association relationship to break cyclic dependency. To select dependencies for removing, *weight* function is defined for each association edges in major level. Weight computation of this strategy is summation of the number of incoming of dependencies of source class and the number of outgoing dependencies of target class. Edge which has higher weight was selected to break cycles

Le Traon et al. [9] strategy used Tarjan's algorithm [7] that is based on depth first search (DFS) to identify strongly connected components (SCCs). To break cycle for this method used *frond* edge as a decision. A frond edge is a relation which is defined as going from one class to another that has been traversed in depth first search, or a class that depended on another directly or indirectly. Weight computation to break cycle is derived from the summation of the number of incoming and outgoing frond edges for a given class, within the SCC under consideration. Incoming edges of maximum weight class in SCC will be selected to remove for breaking cycle. This algorithm finds SCC and removes edges until no cycle remains.

Briand et al. proposed a strategy [4] which is based on techniques of Le Traon et al. [9] and Tai and Daniels [3]. First, Tarjan's algorithm was used to identify SCCs. Next, breaking cycle was performed by removing association edges with the highest weight. These steps were repeated until no cycle remains. But weight computation was modified Tai and Daniels's formula, which association's weight equals the number of incoming edges of the source class multiply the number of outgoing edges of the target class.

Malloy et al. [6] presented a Class Ordering System that is driven by a parameterized cost model. They extended the ORD to include six edge types, *association*, *composition*, *dependency*, *inheritance*, *ownedElement and polymorphic* edge. The first five types of edges are used in UML class diagrams [11]. The *polymorphic* edge is presented in [12, 13] as a dynamic edge. This strategy does not use aggregation edge in ORD because they defined aggregation relation as association relation. Static and dynamic relationships were considered in this approach.

Abdurazik and Offut [8] strategy was different from other graph-based approaches in three respects. Firstly, they modeled classes and their relationships with weights on both nodes and edges. Secondly, weights of node and edges were based on a quantitative measure of coupling. Lastly, they used an algorithm that incorporates edge and node weights as well as number of cycles in cycle breaking.

3. Object-Oriented Slicing

Object-oriented slicing concept is based on program slicing; that is, to isolate the interested part from total program [14, 15, 16, 17]. Object-oriented slicing techniques are constructed for software development with OOP properties such as encapsulation, inheritance, and polymorphism. These slicing techniques are utilized, especially in program understanding and debugging. Although there are many object-oriented slicing techniques, we use only class slicing for searching test order. Because this technique slices class independent. It slices at a destination class so it is suitable to separate the class for cycle breaking.

3.1. Class Slicing

Class slicing [17] is a kind of partial testing in object-oriented slicing techniques. It combines object slicing [19] within it. At first, considering a statement at source class. Data members (attributes) and methods (operations) of a destination class that might influence to the statement of a source class will be cut off. Class slicing does not distinguish data members for different objects instantiated from the same class.

3.2. Representation Slicing with Graphs

There are many graphs most frequently use to represent relation of OO program such as Object-oriented Program Dependency Graph (OPDG) [18] and Object-oriented Dependency Graph (ODG) [16]. However, finding test order by class slicing does not need these graphs because they consist of more details and are complicated to understand. Therefore, we use TDG because class slicing focuses on relationships of

classes and methods only. Transformation of composition and aggregation relations from UML class diagram to TDG in [9] is reciprocal calling, but we define these relationships as one-way calling. In definition of aggregation and composition relations of our TDG, a class which bounds with diamond means a source class whereas another class which bounds with solid line refers to a destination class.

TDG [9] is a model which represents the main structural dependencies between components (classes or methods) in an object-oriented system. General definitions of test dependencies in TDG are test dependency (TD), contractual dependency (CD), implementation dependency (ID) and client contractual dependency (CCD). Types of dependencies are classified to three types of TD: class-to-class, method-to-class and method-to-method. For cycle breaking, we need method call representation to distinguish data members and methods which are called by source class. Thus TDG is suitable for displaying relationships among classes in level of method call.

4. Finding Test Order Strategy

Our strategy uses part of the modified DCSC algorithm [20] with Tarjan's algorithm [7] for finding SCCs. Our strategy is separated into four steps. Firstly, we trim irrelevant classes in SCCs and order these classes. Secondly, we detect all dependency cycles and identify characteristic of cycles. Next, classes that can be separated are selected and they are sliced. Lastly, we use topological sorting to find a test order. The following subsections describe details of each step.

4.1. Trimming classes

In this step, trimming classes is the same as trim in modified DCSC algorithm [20]. It requires reading all classes to search vertices with no ancestors and descendants. The forward trim starts with all nodes with no ancestors and removes them and all their edges. After removal, nodes with no ancestors may appear so this trimming will continue until no more nodes can be removed. Reverse topological sorting can apply for this removal to find test orders. The backward trim performs the same operation but starts with classes with no descendants and uses topological sorting to find test orders. The objective of this trimming is separating classes with no concerning cycles and pre-ordering these classes. If all classes have no SCCs, finding a test order will be completed at this step.

Figure 1 shows graph of trimming classes. Figure 1(a) shows sample classes before trimming. Figure 1(b) shows an order after trimming. Ordering starts in level 3 which is backward trimming. We use a topological sorting so the first class of testing is F and the next one is E. Level 2 is SCCs level. Level 1 is forward trimming. We use a reverse topological sorting so the former is B and the latter is A.

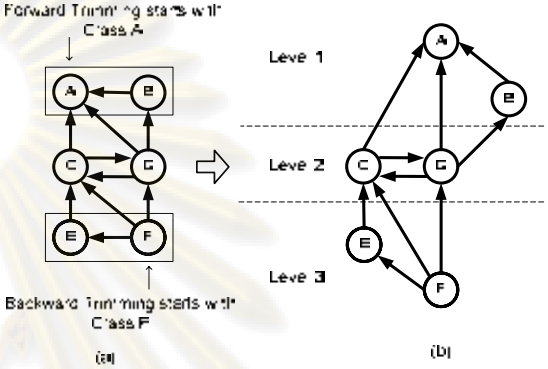


Figure 1. (a) Before trimming. (b) After trimming.

4.2. Detecting cycles

The rest classes from trimming involve to SCCs. In this SCCs, Tarjan's algorithm is used to detect all cycles. After that, we classify cycles into three types. This process assists finding the appropriated class to slice. Topological sorting can be applied to each SCC subdivision to find outline order of a test order.

4.2.1. Classifying SCCs. The main purpose is classifying characteristics of each cycle to select appropriated method for class slicing. (The slicing method will be explained in 4.3) Subdivided cycle will be classified into 1) one cycle or cycles with no overlap part (two cycles have no overlap) 2) cycles with class overlap 3) cycles with path overlap. Figure 2 shows the categories of cycles. Classes which do not belong to any cycles will be defined as remainder classes.

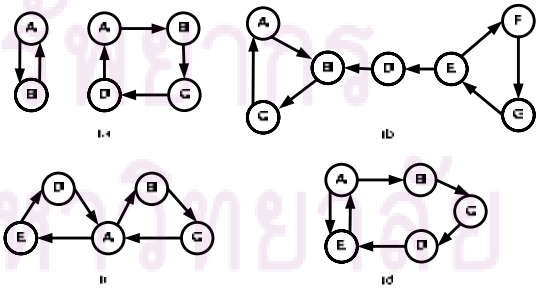


Figure 2. (a) One cycle. (b) No overlap part in two cycles and D is remainder class. (c) Class overlaps in cycle. (d) Path overlaps in cycle.

4.2.2. Ordering each SCC. After classifying, we consider an SCC to be a node. These nodes perform an acyclic graph so they can be ordered by using reverse topological sorting. The objective of this ordering is to find a test order in outline scope. In figure 2(c), we group class A, B and C to a node; class E, F and G to be another node shown in figure 3(a). The SCC with ABC node needs to be tested first, then D. After that, EFG node will be tested. Figure 3(b) shows the result of outline SCC ordering.

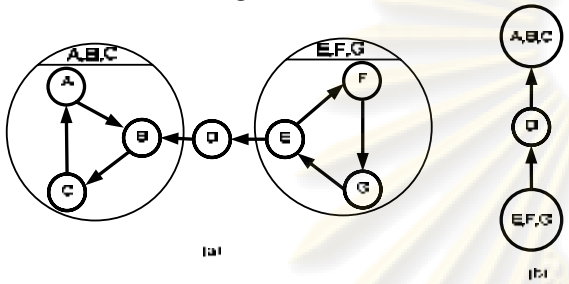


Figure 3. (a) Grouping SCC to node. (b) Outline order

4.3. Slicing classes

The main purpose of this step is to search a class that can be isolated. We separate this class to break cycles from the SCC. First of all, we must select one cycle to be considered at a time. But the first priority type is the smallest cycle of all SCCs from cycle detection. If desired class is found, we will use class slicing to separate and order them. After that the adjacency cycle is connected and class slicing is repeated until no cycle remains. Slicing each cycle types are explained as follows:

4.3.1. Slicing one cycle. Figure 4(a) is a sample cycle for explanation. Operations of slicing one cycle are explained as follows:

1) Select any class in cycle to be a source class, and the descendant of that class is a destination class.

2) Analyze the source class to find methods that call the destination class.

3) Methods and data members of the destination class are sliced if all methods do not call any methods in its descendant node. If this condition is true, then the destination class is sliced. Thus, this cycle is broken, and then we can order these classes. Figure 4(b) displays slicing at class B. The part which can be separated is defined as B'. The remainder is defined as B''. The test order is shown in figure 4(c).

4) If the destination node can not be sliced, the descendant of this node will be considered as the destination class. The former destination class is changed to be the source class. Then, step 3 is repeated. If the new destination class becomes the first destination class that we defined, this cycle can not be sliced. Figure 4(d) shows the slicing in this case. We assume that class C can be separated. Figure 4(e) displays the test order of this case.

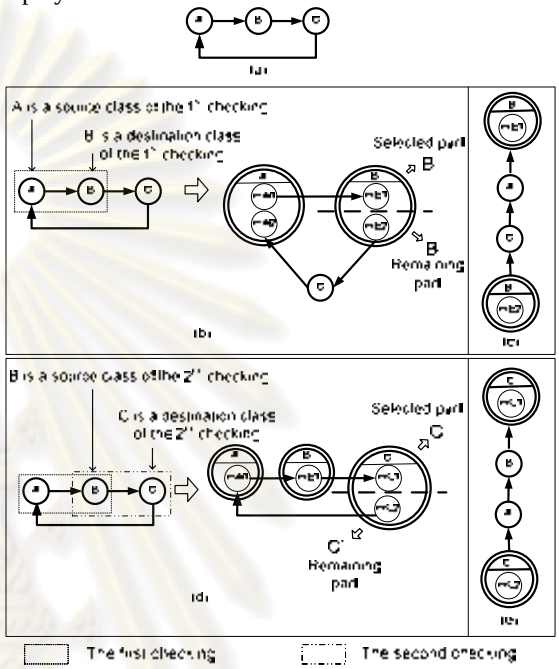


Figure 4. *Slicing one cycle*. (a) Sample cycle. (b) The first check. In this case, class B can be separated. (c) A test order of (b). (d) In case of class B can not be separated, then check next path. (e) A test order of (d).

4.3.2. Slicing cycles with class overlap type. We use the figures in appendix A to explain steps of slicing this cycle type. Operations of this slicing are explained as follows:

1) Select one cycle to be considered first. From (a), we choose cycle with classes D, A, and E.

2) Define the overlap class to be the destination class. The ancestor of this class will be a source class. In (b), the source class is D whereas the destination class is A.

3) Call step 2 in section 4.3.1 to break this cycle. There are two types of breaking cycle. The first type is breaking at the overlap class. The result from this breaking is shown in (c). The second type is breaking at the other class that is not overlap class. The result from this breaking is shown in (k).

4) Select another cycle and connect it to the result from breaking the cycle in the previous step.

4.1) In case of breaking at the overlap class, connecting the result from breaking the cycle at the overlap class to the other cycle could result in either of four different graphs as follows:

- The first type of result has the remainder slice calling a class in the other cycle, and has a class in the other cycle calling the separated slice.
- The second type of result has the separated slice calling a class in the other cycle, and has a class in the other cycle calling the remainder slice.
- The third type of result has a class in the other cycle calling the remainder slice, which calls a class in the other cycle.
- The last type of result has a class in the other cycle calling the separated slice, which calls a class in the other cycle.

The four results from this step for the example in appendix A are shown in (d), (e), (f), and (g) respectively. Only the first type (d) of result is a graph with no cycle. Other types result in a graph with cycle.

4.2) For the other case, connecting the result from breaking the cycle to the other cycle results in only one graph which still contains a cycle. If the example in appendix A is not sliced at class A, the result of connecting would be as shown in (l).

5) Go back to step 3 and continue until there is no more cycle in the graph.

- (m) and (n) show the selected part calls a descendant node of the previous cycle:

  If it is called by the ancestor node of the previous cycle, then the join section is sliced to be a separated part and the rest is the remainder slice. Otherwise, the selected part which is not called by this ancestor will be the separated part.
- (o) and (p) show the selected part does not call a descendant node of the previous cycle:

  If it is called by the ancestor node of the previous cycle, then this section is sliced to be a separated part and the rest is the remainder slice. Otherwise, the selected part which is not called by this ancestor will be the separated part.
- (q) shows the result of the overlap class does not be sliced.

6) Order classes in the graph using topological sorting. The final results from the example are shown in (h), (i), (j), (q), (r), (s), (t), and (u).

4.3.3. Slicing cycles with path overlap type. The figures in appendix B are used as the example for this type of slicing. Steps for slicing are explained below.

1) Select the smallest cycle from the graph.

2) Call step 2 in section 4.3.1 to break this cycle. There are three possible types of sliced node.

- Joint node: a node with more than one of either indegree or outdegree (node A and E in the example)
- Overlap path node: a node on overlap path which is not a joint node (node F in the example).
- Single cycle node: a node which is only in the cycle (node G in the example).

3) Select another cycle and connect it to the result from breaking the cycle in the previous step.

- If the sliced node is a joint node, the result can be a graph either with or without cycle.
- If the sliced node is an overlap path node, the result is always a graph without cycle.
- If the sliced node is a single cycle node, the result is always a graph with cycle.

The three results from this step for the example in appendix B are shown in (c), (d), (f), (g), (m), and (o) respectively.

4) Go back to step 1 and continue until there is no more cycle in the graph.

5) Order classes in the graph using topological sorting. The final results from the example are in (h), (i), (j), (k), (p), and (q).

5. Comparison with using test stubs

We use an example from [2] to compare between test orders found from our strategy and Briand et al. strategy [4]. The graph of the example is shown in Figure 5.

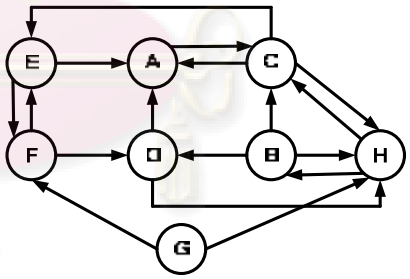


Figure 5. The example graph from [2].

From figure 5, we use our strategy and Briand et al. strategy to find test order. The result of test order is shown in Figure 6. Our strategy does not need to test slice B and C because they pass unit testing. At level 1, Briand et al. strategy starts with stub creation whereas our strategy can start at level 2. At level 2-4, our strategy can parallel test so it finishes testing at level 6 while Briand et al. strategy finishes at level 8. From this comparison, our strategy uses less time and does not need to create stubs.

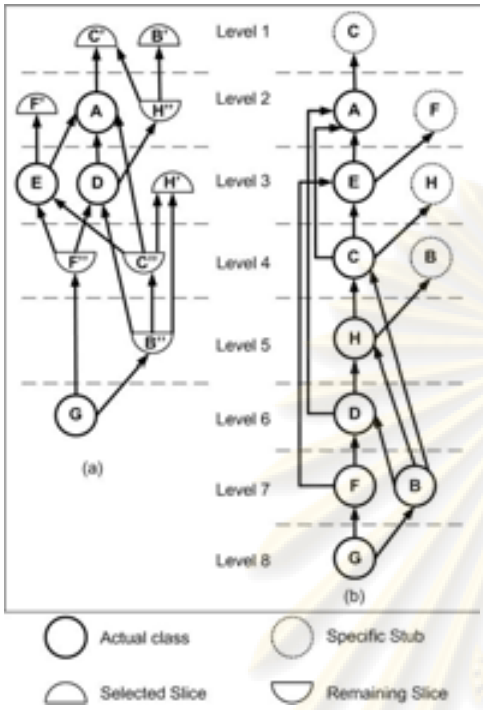


Figure 6. (a) Class test order from this strategy. (b) Class test order from Briand et al. strategy.

6. Conclusion and Future Work

In this paper, we have presented a new approach to find a test order. To order classes for testing, we isolate classes for partial testing so as to break dependency cycles. Our strategy decreases test stubs construction and testing with actual classes can detect faults more than using stubs. Moreover, it reduces time in testing phase because in some cases, testing partial classes can be run test parallel.

For our future work, we plan to improve our algorithm to separate mutually related composite classes such as wrapper class, and content class containing each other. According to characteristics of object-oriented programming, polymorphism and dynamic binding make a priori of complexity in relationships between classes. Thus, finding a test order needs some auxiliary, such as sequence diagrams. The sequence diagrams are typically used to model usage scenarios. It shows message sending sequences and method calls. These features may help determine a test order.

7. References

[1] R. V. Binder, Testing Object-Oriented Systems—Models, Pattern, and Tools, Addison-Wesley, 1999.

[2] L.C Briand, Y. Labiche and Y. Wang, "An Investigation of Graph-Based Class Integration Test Order Strategies", IEEE Transactions on Software Engineering, vol. 29, no. 7, pp. 594-607, July 2003.

[3] K.-C. Tai and F. J. Daniels, "Test Order for Inter-Class Integration Testing of Object-Oriented Software", *COMSAC '97*, 1997.

[4] L.C Briand, Y. Labiche and Y. Wang, "Revisiting strategies for ordering class integration testing in the presence of dependency cycle", *ISSRE'01*, 2001.

[5] V. Le Hahn, K. Akif, Y. Le Traon and J. M. Jézéquel, "Selecting an Efficient OO Integration Testing Strategy: An Experimental Comparison of Actual Strategies", *ECOOP'01*, pp. 381-401, 2001.

[6] B. A. Malloy, P. J. Clarke and E. L. Lloyd, "A Parameterized Cost Model to Order Classes for Class-based Testing of C++ Applications", *Proc. of the ISSRE' 03*, 2003.

[7] R. Tarjan, "Depth-first search and linear graph algorithms", *SIAM J. Computing*, vol. 1, pp. 146-160, June, 1972.

[8] A. Abdurazik and Jeff Offutt, "Coupling-based Class Integration and Test Order", *AST' 06*, May 2006.

[9] Y. Le Traon, T. Jéron. J.-M. Jézéquel, and P. Morel, "Efficient Object-Oriented Integration and Regression Testing", *IEEE Trans. Reliability*, vol. 49, no. 1, pp. 12-25, 2000.

[10] D. Kung , J. Gao, P. Hsia, J. Lin and Y. Toyoshima, "Class Firewall, Test Order, and Regression Testing of Object-Oriented Programs", *Journal of Object-Oriented Programming*, vol. 8, no. 2, pp. 51-65, 1995.

[11] OMG Unified Modeling Language Specification, http://www.omg.org/cgi-bin/doc?formal/03-03-01.pdf, March 2003.

[12] Y. Labiche, P. Thévenod-Fosse, H. Waeselynck and M.-H. Durand, "Testing Levels for Object-Oriented Software", *ICSE* 2000.

[13] Q. Chen and X. Li, "An Order-Assigned Strategy of Classes Integration Testing Based on Test Level", *IEEE* 2003.

[14] Mark Weiser, "Program Slicing", *IEEE Trans. Software Engineering*, 16(5), pp. 498-509, 1984.

[15] F. Tip, "A survey of program slicing techniques" *Journal of Programming Languages*, 3(3), 1995.

[16] J.-L. Chen, F.-J. Wang, Y.-L. Chen, "Slicing Object-Oriented Programs", In *Proc. of the APSEC'97, IEEE*, 1997.

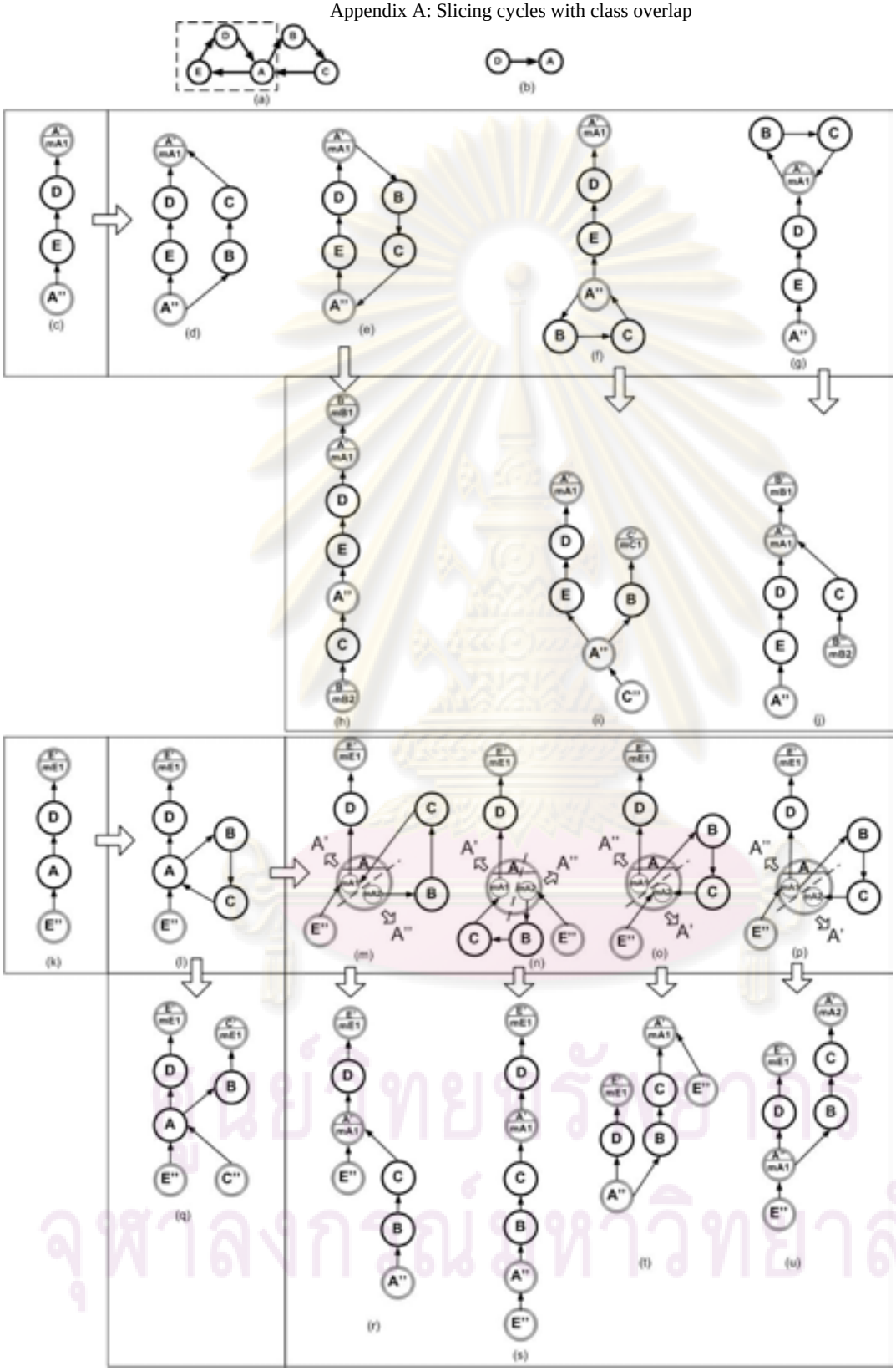
[17] Z. Chen, B. Xu, "Slicing Object-Oriented Java Programs", *ACM SIGPLAN*, vol. 36(4), 2001.

[18] Brian A. Malloy, John D. McGregor, A. Krishnaswamy, Murali Medikonda, "An Extensible Program Representation for Object-Oriented Software", *ACM SIGPLAN*, vol. 29(12), 1994.

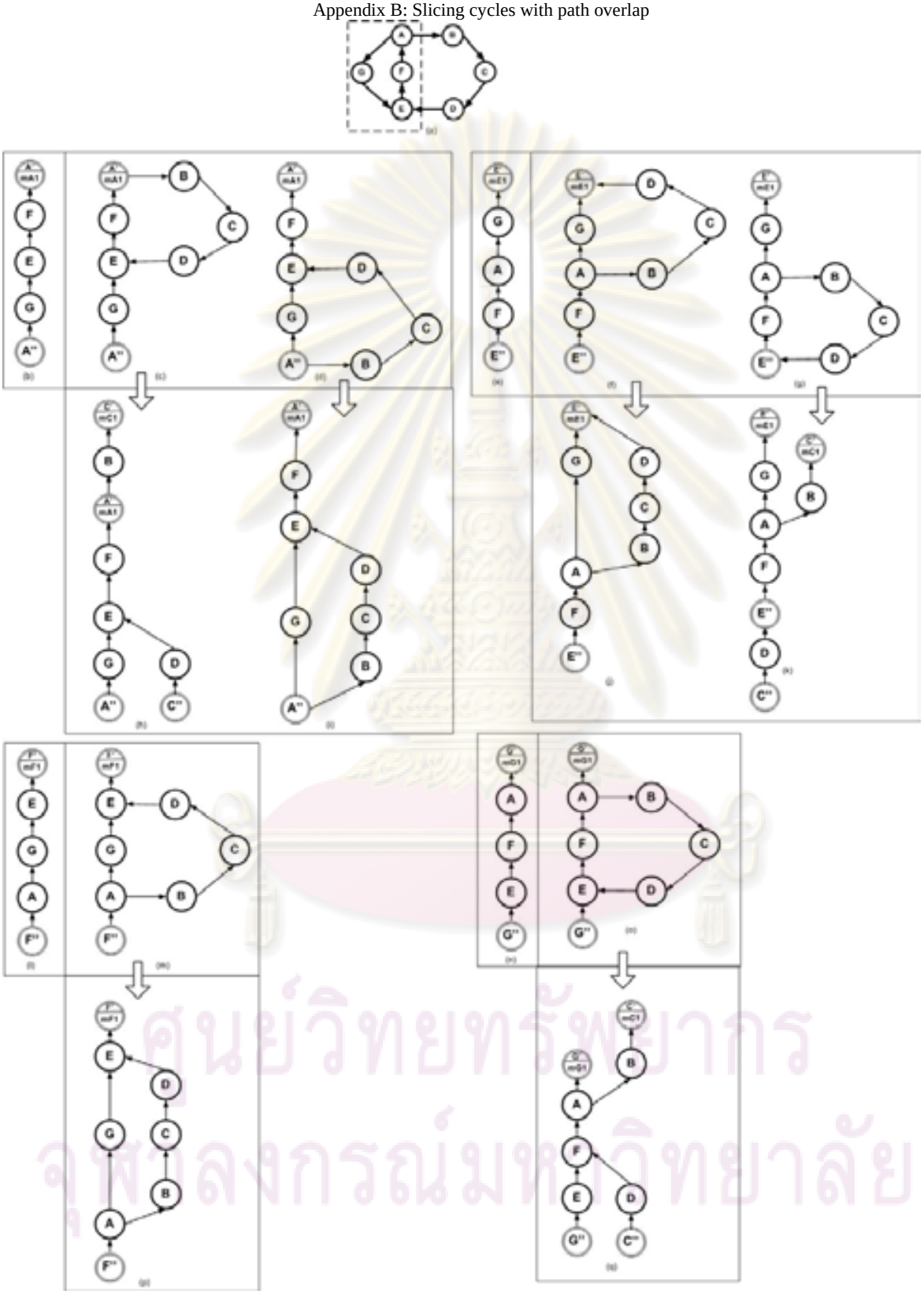
[19] D. Liang, M. J. Harrold, "Slicing Objects Using System Dependence Graph", *Proc. of the international Conference on Software Maintenance*, 1998.

[20] W. McLendon III, B. Hendrickson, S. J. Plimpton, L. Rauchwerger, "Finding strongly connected components in distributed graphs", *Journal of Parallel and Distributed Computing*, pp. 901-910, 2005.

Appendix A: Slicing cycles with class overlap



Appendix B: Slicing cycles with path overlap



Class Ordering Tool – A Tool for Class Ordering in Integration Testing

Jutarat Jaroenpiboonkit and Taratip Suwannasart
Center of Excellence in Software Engineering,
Department of Computer Engineering, Faculty of Engineering
Chulalongkorn University, Bangkok 10330 Thailand
Jutarat.Ja@student.chula.ac.th, Taratip.S@chula.ac.th

Abstract

The presence of cyclic dependency calls in integration testing is a major problem to determine an order of classes to be tested. We proposed an approach [1] to solve this problem. Hence, this paper proposes a Class Ordering Tool (CO Tool) for integration testing based on our approach for finding a test order using object-oriented slicing technique. Our approach breaks cycles by slicing classes for partial testing, whereas many researchers delete relationships to break cycles and create stub to represent interactions of testing between classes. Therefore, the benefit of our approach is to decrease the cost of implementing test stubs.

1. Introduction

Object-oriented programs typically have many small components and more interfaces than traditional programs [2]. As a result of that, it is difficult to detect inter-component faults in integration testing. However, one major problem in integration testing is the existing of cyclic dependency calls or cycle. A cycle is a loop in a directed graph. For example, suppose that class A calls class B, class B calls class C, and class C calls back to class A. If relationship between classes performs as this example, determining test order of classes in order to execute interactions can not be performed. A cluster of classes with cycle is called Strongly Connected Component (SCC).

Many researchers find out solutions of the problem by proposing techniques to remove relationships of classes in order to break cyclic dependencies and produce graphs with no cycle [3, 4, 5, and 6]. These graphs are called Directed Acyclic Graph (DAG). Deleting a relationship means construction of a test stub. A stub is a dummy component used to simulate the behavior of a real component [7]. Therefore, a main purpose of integration testing is minimization of the effort to create stubs. That is an NP-complete problem [3, 4]. A number of test stubs are a cost factor for integration testing [4].

In this paper, we propose a Class Ordering Tool – A tool for Class Ordering in integration testing. It is an automated tool implemented in C# for generating a test order. CO Tool is composed of three main subsystems: 1) analyzing relationships, 2) breaking cycles and generating sliced classes, 3) sorting and visualizing.

The remainder of this paper is organized as follows: section 2 provides our approach for finding a Test Order using Object-Oriented Slicing Technique. Section 3, we describe the system architecture of CO Tool and details in each subsystem. CO Tool Application is shown in section 4. Section 5 compares results from our approach with other strategies. Finally, conclusion is drawn in section 6.

2. An Approach for Finding a Test Order using Object-Oriented Slicing Technique

Our approach [1] uses slicing class technique to break cycle instead of removing relationship between classes. The main purpose of using object-oriented slicing in this approach is to take out the isolated part from a class for finding a test order. An object-oriented slicing [8] that used in this approach is the class slicing [9]. Slicing classes is continued until all cycles are broken. If there is no cycle remains, ordering class is applied by using topological sorting. Our approach is composed of four steps which are explained as follows:

1) Trimming irrelevant classes in SCCs and order these classes.
2) Detecting all SCCs and classifying each SCC to cycle type.
3) Slicing classes in each SCC by cycle type.
4) Sorting classes to find a test order.

3. CO Tool System Architecture

Class Ordering Tool, CO Tool, which is an automated tool to generate an order of classes to be tested, is developed based on our approach. CO Tool displays a graphical representation of the test class order and class interactions. Besides the visualization, CO Tool can generate a source code file for each part of the sliced class in order to provide interaction testing by a test order output.

The tool architecture is composed of three main subsystems, namely the analyzing relationship subsystem, the breaking cycle subsystem, as well as the sorting and visualizing subsystem. The input of CO Tool is source code of all classes under test (CUT). The analyzing relationship subsystem produces the class relationship detection to obtain the Class Relationship Table (CRT) and SCCs List. After finishing the analyzing relationship subsystem, the breaking cycle subsystem analyses CRT and SCCs list with source code to break cyclic dependencies. In addition, the breaking cycle subsystem generates source code of sliced classes and produces CRT without SCCs. Finally, the sorting and visualizing subsystem sorts classes from CRT without SCCs is to find the class order list and allows a tester to visualize class interactions. An overview of the tool architecture is shown in figure 1.

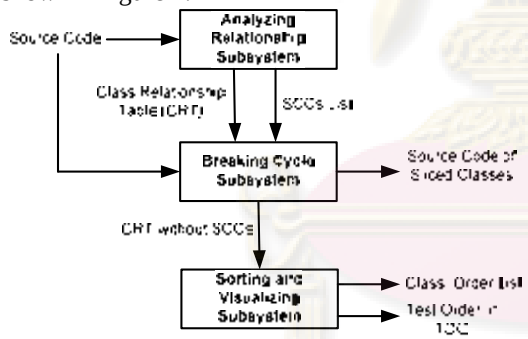


Figure 1. CO Tool Architecture.

3.1. Analyzing Relationship Subsystem

Figure 2 depicts the components of the analyzing relationship subsystem. This subsystem is composed of four components. The main function of this subsystem is to provide the necessary information for the breaking cycle subsystem. The output of this subsystem is CRT and SCCs list. Each component is explained as follows:

3.1.1. Relationship Detector. Source codes of all classes are loaded by relationship detector. After that, relationship detector calls C# parser to parse source code and gets class structure to detect all relationships between classes. If detection process finishes, CRT will be produced.

3.1.2. C# Parser. The function of this parser [10] is parsing source code to the class structure format based on ECMA-334 C# Grammar [11].

3.1.3. SCC Detector. CRT produced from relationship detector is the input of the SCC detector. The SCC detector translates CRT into node and edge format based on G(V, E) [12], and then parses using tarjan's algorithm to construct SCCs list from the node list.

3.1.4. Tarjan's Algorithm. This algorithm [13] based on depth first search is used to identify SCC.

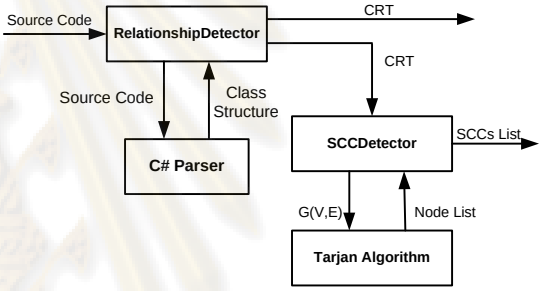


Figure 2. Analyzing Relationship Subsystem.

3.2. Breaking Cycle Subsystem

The breaking cycle subsystem is drawn in figure 3. The objective of this subsystem is to find the separated class. After the desired class is found, breaking cycle will be operated until no cycle remains. In addition, CRT without SCCs and source code files of sliced classes are generated to be the output of this subsystem. The following subsections describe the components of the breaking cycle subsystem.

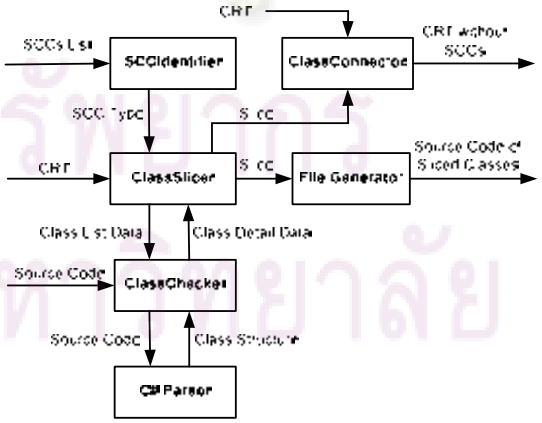


Figure 3. Breaking Cycle Subsystem.

3.2.1. SCC Identifier. SCCs list, the input of SCC identifier, is used to identify SCC type which is defined based on our approach [1]. It is a necessary function for selecting the appropriated slicing type of slicing class.

3.2.2. Class Slicer. Slicing type is chosen by SCC type. In case of one cycle type, slicing one cycle process is invoked directly. On the other hand, if SCC type is class over type or path overlap type, then slicing processes of both types will split overlap cycles to one cycle first and call slicing one cycle afterward. Details of each slicing type are in [1]. Slicing process is repeated until no cycle remains. After this process is completed, the slice is produced.

3.2.3. Class Checker. The class list derived from the class slicer is used to select source files for consideration. The class checker calls the C# parser [13] to parse the source code to obtain a class structure format. After that, Analyzing process is driven to produce the output class details.

3.2.4. C# Parser. This parser is same as 3.1.2.

3.2.5. Class Connector. The information of slice is used to reconstruct the relationships between classes. The objective of the class connector is to produce the new CRT which is CRT without SCCs.

3.2.6. File Generator. The slice produced from the class slicer is used to generate the source code of sliced classes by using the file generator. The purpose of this component is to produce sliced class files which are called by the caller class during interaction testing.

3.3. Sorting and Visualizing Subsystem

Figure 4 shows two components of the sorting and visualizing subsystem. The main propose of this subsystem is to order classes after slicing. In addition, transformation of test class order list to TDG can help a tester to depict an order obviously. The class sorter and the test order visualizer are explained as follows:

3.3.1. Class Sorter. After the breaking cycle subsystem is processed, it returns the CRT which contains class relationship list without cyclic dependencies. The class sorter processes CRT with the ordering function by using the backward topological sorting. It starts with classes with no ancestor or the root node. The root node means the ancestor class; that is, the caller class which is not called by any classes. Therefore, the root class is the final order to be tested. The descendants of the root class are sorted into the next order consequently.

3.3.2. Test Order Visualizer. The primary function of the test order visualizer is to visualize the TDG representation of classes, which graphically displays class interactions. The class order list is transformed into TDG aspect to illustrate the test class order. Drawing graphs is based on Nachmanson et al. strategy [14]. This visualizer is composed of Graph Layout Execution Engine (GLEE), a .NET library for graph layout and viewing developed at Microsoft Research [14, 15].

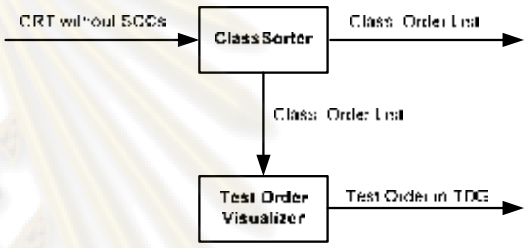


Figure 4. Sorting and Visualizing Subsystem.

4. CO Tool Application

The CO Tool is implemented to automatically find the test order for integration testing. There are six main windows and two menu items which are the main components of CO Tool user interface.

4.1. The main components of CO tool user interface

After CO Tool loads source code files for consideration, the information of these source files are shown in three windows:

- Class List window: It is located on the top left position and used to display all class names within consideration. This window is shown in figure 7(a).
- SCCs window: This window is below window of class list window and shows the lists of class names in each SCC. This window is shown in figure 7(b).
- SCC Graph window: The graph which is shown on the middle of the application is the SCC Graph. It depicts the aspect of relationships between classes before cycle breaking. A tester can use the class list to identify node id in order to see what the class name is. This window is shown in figure 7(c).

The remaining windows, which are used to display the test class order output, include text format and graphical view.

- New Class List window: This window position is located below the SCCs window. It shows a list of new id and class name. This list is not a test order because it is used to display the id and class name of sliced classes only. Classes, which are sliced, become new classes and will be named the same as old class names but extended with _1. The remainder classes still use the old name. This window is shown in figure 7(d).

- Test Order window: It located on the bottom left position and used to display the test class order. This window displays the test class order. Classes in the same level mean that they can be parallel tested run. This window is shown in figure 7(e).

- Test Order Graph window: This window is shown on the right of the application. Showing the test class order in the graphical view is useful for the user to portray the test class clearly. This window is shown in figure 7(f).

Two menu items which manipulate file input/output stream are shown in figure 5.

- Open Folder menu item: The folder which contains source code files is selected in order that the source code files will be loaded to find a test order.

- Generate Slice menu item: It is used to generate source code of the separated part that will become a new class. The file name of new class is the name with extension. Nevertheless, the actual class name in the content of file is still the same after generating the new class.

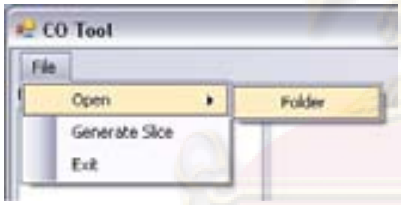


Figure 5. Menu items of CO Tool.

4.2. An example result of using CO Tool

TSDU (Triangulated Surface Drawing Utility) system [16] is the case study for CO Tool. Actually, there are 44 classes, but merely 7 classes related with cyclic dependencies. Therefore, 7 classes are selected for testing. Class diagram of this system is shown in figure 6. The result after finding a test class order is shown in figure 7 consequently. In TSDU system, there is one SCC which is path overlap type containing 7 cycles overlap. MouseListener, KeyListener, and FPSCounter are sliced to break all cyclic dependencies. After no cycle remains, a test order is displayed on the Test Order window at the bottom of CO Tool window in figure 7.

Figure 6. The example class diagram from [16].

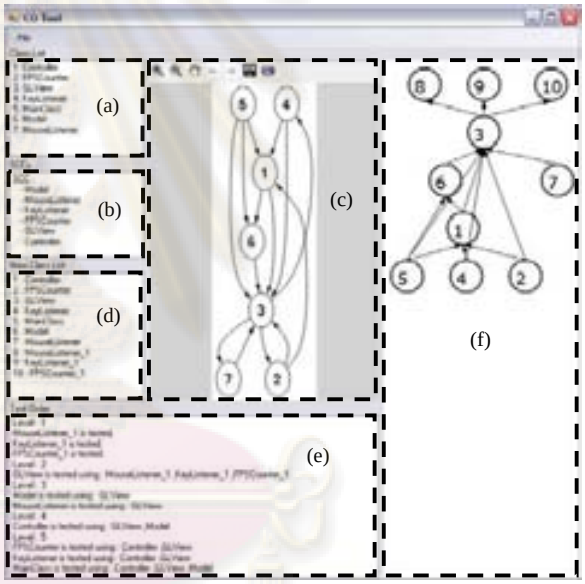


Figure 7. The CO Tool User Interface.

5. Comparison with other works

Table 1 shows the results of four strategies for the TSDU system. In terms of number of stubs for comparison, our approach does not need to create stub so it takes time less than the other approaches. For comparison with the number of test steps (the length of longest path), other approaches start with stub creation whereas our approach starts at step 2 because classes at step 1 pass unit testing. If unit testing at step 1 is not counted as a path length, the path length of our approach is four that equals the minimum path length from the approach of Tai and Daniels [5].

Table 1. Comparison a test order of four strategies in integration testing for TSDU System

Step	Tested classes in each step			
	Tai's approach	Traon's approach	Briand's approach	Our approach
1	Construct: stub(MouseListener,GLView) stub(FPSCounter,GLView) stub(KeyListener,GLView) stub(Controller,MainClass) stub(Model,MainClass) stub(Controller,KeyListener) stub(Controller,FPSCounter)	Construct: stub(GLView,Model) stub(GLView,MouseListener) stub(GLView,Controller) stub(GLView,FPSCounter) stub(GLView,KeyListener)	Construct: stub(KeyListener,GLView) stub(FPSCounter,GLView) stub(MouseListener,GLView)	Unit test: - slice(MouseListener_1,GLView) - slice(KeyListener_1,GLView) - slice(FPSCounter_1,GLView)
2	GLView is tested using : stub(MouseListener,GLView) , stub(FPSCounter,GLView) , stub(KeyListener,GLView)	- Model is tested using : stub(GLView,Model) - MouseListener is tested using : stub(GLView,MouseListener)	- GLView is tested using : stub(KeyListener,GLView) , stub(FPSCounter,GLView) , stub(MouseListener,GLView)	- GLView is tested using : slice(MouseListener_1,GLView) , slice(KeyListener_1,GLView) , slice(FPSCounter_1,GLView)
3	- MainClass is tested using : GLView , stub(Controller,MainClass) , stub(Model,MainClass) - Model is tested using : GLView - MouseListener is tested using : GLView - KeyListener is tested using : GLView , stub(Controller,KeyListener) - FPSCounter is tested using : GLView , stub(Controller,FPSCounter)	Controller is tested using : Model , stub(GLView,Controller)	- Model is tested using : GLView - MouseListener is tested using : GLView	- Model is tested using : GLView - MouseListener is tested using : GLView
4	Controller is tested using : Model ,GLView	- FPSCounter is tested using : Controller , stub(GLView,FPSCounter) - KeyListener is tested using : Controller , stub(GLView,KeyListener)	Controller is tested using : Model ,GLView	Controller is tested using : Model ,GLView
5		GLView is tested using : MouseListener , KeyListener , FPSCounter	- MainClass is tested using : Controller ,Model ,GLView - FPSCounter is tested using : Controller ,GLView - KeyListener is tested using : Controller ,GLView	- MainClass is tested using : Controller ,Model ,GLView - FPSCounter is tested using : Controller ,GLView - KeyListener is tested using : Controller ,GLView
6		MainClass is tested using : Controller , Model , GLView		
#Stubs	7	5	3	0

6. Conclusion

We have presented CO Tool, a tool for class ordering in integration testing based on our approach for finding a Test Order using Object-Oriented Slicing Technique [1]. The major benefits of this tool are its ability to automatically generate the test order of classes and to visually display class interactions by using GLEE. In addition, a tester can provide interaction testing follow a test order with the generated slice file from a sliced class.

7. References

[1] J. Jaroenpiboonkit, T. Suwannasart, "Finding a Test Order using Object-Oriented Slicing Technique", *APSEC'07*, 2007.

[2] R. V. Binder, *Testing Object-Oriented Systems—Models, Pattern, and Tools*, Addison-Wesley, 1999.

[3] V. Le Hahn, K. Akif, Y. Le Traon and J. M. Jézéquel, "Selecting an Efficient OO Integration Testing Strategy: An Experimental Comparison of Actual Strategies", *ECOOP'01*, pp. 381-401, 2001.

[4] L.C Briand, Y. Labiche and Y. Wang, "An Investigation of Graph-Based Class Integration Test Order Strategies", *IEEE Transactions on Software Engineering*, vol. 29, no. 7, pp. 594-607, July 2003.

[5] K.-C. Tai and F. J. Daniels, "Test Order for Inter-Class Integration Testing of Object-Oriented Software", *COMSAC '97*, 1997.

[6] Y. Le Traon, T. Jéron. J.-M. Jézéquel, and P. Morel, "Efficient Object-Oriented Integration and Regression Testing", *IEEE Trans. Reliability*, vol. 49, no. 1, pp. 12-25, 2000.

[7] B. Beizer, *Software Testing Techniques*, Van Nostrand Reinhold Inc., 2nd edition, 1990.

[8] D. Liang, M. J. Harrold, "Slicing Objects Using System Dependence Graph", *International Conference on Software Maintenance*, pp.358-367, 1998.

[9] Z. Chen, B. Xu, "Slicing Object-Oriented Java Programs", *ACM SIGPLAN*, vol. 36(4), 2001.

[10] Denis Erchoff, A handwritten CSharp parser, http://www.codeplex.com/csparser, 2007.

[11] Standard ECMA-334, C# Language Specification, December 2001, http://www.ecma-international.org/publications/standards/ecma-334.htm

[12] Douglas B. West, *Introduction to Graph Theory*, Second Edition, Prentice Hall, 2001.

[13] R. Tarjan, "Depth-first search and linear graph algorithms", *SIAM J. Computing*, vol. 1, pp. 146-160, June, 1972.

[14] L. Nachmanson, G. Robertson, B. Lee, "Drawing graphs with GLEE", *Lecture Notes in Computer Science*, Springer-Verlag Berlin Heidelberg, LNCS 4875, pp. 389–394, 2007.

[15] Microsoft Research, GLEE, version 1.0, http://research.microsoft.com/research/downloads/Details/c927728f-8872-4826-80ee-ecb842d10371/Details.aspx, 2006.

[16] Chris J. Friesen, Triangulated Surface Drawing Utility, http://sourceforge.net/projects/triangles/, 2003.

ประวัติผู้เขียนวิทยานิพนธ์

นางสาวจุฑารัตน์ เจริญไพบูลย์กิจ เกิดวันที่ 10 กุมภาพันธ์ พ.ศ. 2523 สำเร็จการศึกษาระดับปริญญาวิศวกรรมศาสตรบัณฑิต จากภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหาร ลาดกระบัง ในปีการศึกษา 2545 และเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2549