การจัดลำดับการสร้างกรณีทดสอบมิวเทชันด้วยหน่วยวัดค่าความเป็นศูนย์กลาง

นายศุภชัย ทรัพย์มาก

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมซอฟต์แวร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2564

Prioritization of Mutation Test Case Generation with Centrality Measures

Mr. Supachai Supmak

A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science in Software Engineering

Department of Computer Engineering

FACULTY OF ENGINEERING

Chulalongkorn University

Academic Year 2021

หัวข้อวิทยานิพนธ์	การจัดลำดับการสร้างกรณีทดสอบมิวเทชันด้วยหน่วยวัดค่าความเป็นศูนย์กลาง
โดย	นายศุภชัย ทรัพย์มาก
สาขาวิชา	วิศวกรรมซอฟต์แวร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร.ญาใจ ลิ่มปิยะกรณ์

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(ศาสตราจารย์ ดร.สุพจน์ เตชวรสินสกุล)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.สุกรี สินธุภิญโญ)

..... อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(รองศาสตราจารย์ ดร.ญาใจ ลิ่มปิยะกรณ์)

..... กรรมการ
(อาจารย์ ดร.พิตติพล คันธวัฒน์)

..... กรรมการภายนอกมหาวิทยาลัย
(อาจารย์ ดร.ภาสกร อภิรักษ์วรพินิต)

ศุภชัย ทรัพย์มาก : การจัดลำดับการสร้างกรณีทดสอบมิวเทชันด้วยหน่วยวัดค่าความเป็นศูนย์กลาง. (Prioritization of Mutation Test Case Generation with Centrality Measures) อ.ที่ปรึกษาหลัก : รศ. ดร.ญาใจ ลิ่มปิยะกรณ์

การทดสอบการกลายพันธุ์สามารถนำไปใช้กับการประเมินคุณภาพของกรณีทดสอบได้ การจัดลำดับความสำคัญของการสร้างการทดสอบการกลายพันธุ์เป็นองค์ประกอบที่สำคัญของแนวปฏิบัติทางอุตสาหกรรมที่จะมีส่วนช่วยในการประเมินกรณีทดสอบ โดยทั่วไปแล้ว อุตสาหกรรมจะส่งมอบผลิตภัณฑ์ภายใต้เงื่อนไขของเวลาสู่ตลาด ดังนั้นจึงต้องเสียสละงานทดสอบซอฟต์แวร์อย่างหลีกเลี่ยงไม่ได้ แม้ว่าจะต้องใช้กรณีทดสอบจำนวนมากสำหรับการตรวจสอบซอฟต์แวร์ การใช้การวัดศูนย์กลางเครือข่ายสังคม เพื่อจัดลำดับความสำคัญของการสร้างการทดสอบการกลายพันธุ์ ซอร์สโค้ดที่มีค่าเพจแรงก์สูงสุด จะถูกเน้นก่อนเมื่อพัฒนากรณีทดสอบ เนื่องจากโมดูลเหล่านี้เสี่ยงต่อข้อบกพร่องหรือความผิดปกติซึ่งอาจทำให้เกิดข้อบกพร่องที่ตามมาในโมดูลที่เกี่ยวข้องอื่นๆ นอกจากนี้ แนวทางดังกล่าวจะช่วยระบุกรณีทดสอบที่ลดได้ในชุดทดสอบ โดยยังคงรักษาเกณฑ์เดียวกันกับจำนวนกรณีทดสอบเดิม

สาขาวิชา วิศวกรรมซอฟต์แวร์ ลายมือชื่อนิสิต .....

ปีการศึกษา 2564 ลายมือชื่อ อ.ที่ปรึกษาหลัก .....

6370281021 : MAJOR SOFTWARE ENGINEERING

KEYWORD: mutation testing, test case prioritization, centrality measures

Supachai Supmak : Prioritization of Mutation Test Case Generation with Centrality Measures. Advisor: Assoc. Prof. Yachai Limpiyakorn, Ph.D.

Mutation testing can be applied for quality assessment of test cases. Prioritization of mutation test generation has been a critical element of the industry practice that would contribute for the evaluation of test cases. The industry generally delivers the product under the condition of time to the market and thus, inevitably to sacrifice software testing tasks, even though many test cases are required for software verification. Using a social network centrality measure to prioritize mutation test generation. The source code with the highest values of PageRank, will be focused first when developing their test cases as these modules are vulnerable for defects or anomalies which may cause the consequent defects in many other associated modules. Moreover, the approach would help identify the reducible test cases in the test suite, still maintaining the same criteria as the original number of test cases.

Field of Study: Software Engineering Student's Signature .....

Academic Year: 2021 Advisor's Signature .....

กิตติกรรมประกาศ

วิทยานิพนธ์เล่มนี้สำเร็จสมบูรณ์ได้ด้วยดีเพราะได้รับความกรุณาอย่างดียิ่งจาก รองศาสตราจารย์ ดร.ญาใจ ลิ่มปิยะกรณ์ อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ได้ประสิทธิ์ประสาทความรู้ สละเวลาอันมีค่าให้คำแนะนำและคำปรึกษา คอยผลักดัน ติดตามความก้าวหน้า และตรวจสอบแก้ไขข้อบกพร่องของงานวิจัยเป็นอย่างดียิ่งมาโดยตลอด ทำให้งานวิจัยนี้สำเร็จลุล่วงไปได้ด้วยดี ผู้วิจัยขอกราบขอบพระคุณด้วยความเคารพอย่างสูงมา ณ โอกาสนี้

ขอขอบพระคุณ ผู้ช่วยศาสตราจารย์ดร.สุกรี สินธุภิญโญ ประธานกรรมการสอบวิทยานิพนธ์ อาจารย์ดร.พิตติพล คันธวัฒน์ และ อาจารย์ดร.ภาสกร อภิรักษ์วรพินิต กรรมการสอบวิทยานิพนธ์ที่กรุณาเสียสละเวลาอันมีค่า ในการตรวจสอบและให้คำแนะนำที่เป็นประโยชน์ในการทำวิทยานิพนธ์ในครั้งนี้

ขอขอบพระคุณบิดามารดา และญาติพี่น้องที่ได้ให้การสนับสนุน คอยเป็นห่วงและเป็นกำลังใจที่ดีเสมอมา และขอขอบคุณเพื่อน ๆ พี่ ๆ น้อง ๆ ที่ภาควิชาวิศวกรรมคอมพิวเตอร์ทุกคน ที่คอยสนับสนุน ติดตาม และให้กำลังใจมาโดยตลอด คอยให้คำแนะนำ ความช่วยเหลือ และคำปรึกษาเป็นอย่างดี รวมถึงขอขอบพระคุณนาย จักรพงศ์ ภูบาล และนาย นันทวัฒน์ หล้าชิว ที่กรุณาได้ให้คำปรึกษาและช่วยเหลือในการทำการทดสอบหน่วยย่อย รวมถึงขอขอบพระคุณท่านผู้เกี่ยวข้องทุกท่านซึ่งมิได้กล่าวลงนามไว้ ณ ที่นี้ด้วย และท้ายที่สุดขอบคุณตัวเองที่ไม่ยอมแพ้และไม่ท้อถอย ซึ่งทำให้วิทยานิพนธ์สำเร็จลุล่วงไปได้ด้วยดี ผู้วิจัยหวังเป็นอย่างยิ่งว่าวิทยานิพนธ์ฉบับนี้จะเป็นประโยชน์บ้างไม่มากก็น้อยต่อผู้ที่สนใจที่จะศึกษาต่อไปในภายภาคหน้า

ศุภชัย ทรัพย์มาก

สารบัญ

หน้า

สารบัญตาราง

หน้า

สารบัญรูปภาพ

หน้า

บทที่ 1
บทนำ

1.1.ที่มาและความสำคัญของปัญหา

ปัจจุบัน ระบบซอฟต์แวร์ได้ถูกพัฒนาขึ้นเพื่อสนับสนุนการทำงานให้มีประสิทธิภาพและพร้อมใช้งานให้ได้เร็วที่สุด อย่างไรก็ตาม กระบวนการพัฒนาซอฟต์แวร์ที่รวดเร็ว อาจส่งผลให้เกิดข้อผิดพลาดต่อระบบและส่งผลให้ความพึงพอใจต่อการใช้งานของลูกค้าลดลง การทดสอบซอฟต์แวร์ที่มีประสิทธิผลและประสิทธิภาพจึงมีความสำคัญเพื่อค้นหาข้อผิดพลาดและดำเนินการแก้ไข ก่อนส่งมอบผลิตภัณฑ์ให้กับลูกค้า ดังนั้น องค์กรจึงเล็งเห็นถึงความสำคัญในการออกแบบเทคนิคสำหรับการทดสอบซอฟต์แวร์ เพราะการทดสอบซอฟต์แวร์เป็นส่วนหนึ่งของการพัฒนาซอฟต์แวร์ก่อนส่งมอบซอฟต์แวร์ให้กับลูกค้า นอกจากนี้ การทดสอบในภาวะจำกัดในเรื่องของเวลาอาจส่งผลกระทบต่อซอฟต์แวร์ที่จะส่งมอบ เช่น การทดสอบการทำงานซ้ำก่อนการปรับใหม่อาจทำให้เสียเวลาในการทดสอบเป็นอย่างมากและอาจส่งผลให้ไม่ทันตามเวลาที่กำหนดส่งมอบ นักทดสอบจึงจำเป็นต้องหาวิธีการมาสนับสนุนการทดสอบ โดยวิธีการที่ผู้วิจัยนำเสนอ คือ การทดสอบมิวเทชัน (Mutation Testing) ทั้งนี้ การสร้างกรณีทดสอบด้วยวิธีการทดสอบมิวเทชันจำเป็นต้องมีขั้นตอนการดำเนินงานอย่างชัดเจน กอปรกับเทคนิคและเครื่องมือที่นำมาใช้ในการทดสอบต้องประยุกต์ใช้ร่วมกับการสร้างกรณีทดสอบได้ เพื่อประโยชน์ในการลดระยะเวลาในการดำเนินงาน เพิ่มประสิทธิภาพของการทดสอบเป็นสำคัญ และสามารถนำข้อมูลมาวิเคราะห์และประยุกต์ใช้ในการพัฒนาระบบต่อไป

งานวิจัยนี้ได้นำเสนอเทคนิคการประเมินกรณีทดสอบด้วยวิธีการมิวเทชัน โดยมุ่งเน้นการเลือกตัวดำเนินการมิวเทชันด้วยเครื่องมือที่ช่วยสแกนมิวเทชัน และการจัดลำดับกรณีทดสอบด้วยหน่วยวัดค่าความเป็นศูนย์กลาง การประเมินผลงานวิจัยประกอบด้วยการวิเคราะห์เปรียบเทียบค่าความเป็นศูนย์กลางต่าง ๆ ที่นำเสนอในงานวิจัย

1.2.วัตถุประสงค์

ศึกษา วิเคราะห์ เปรียบเทียบความสามารถในการจัดลำดับกรณีทดสอบมิวเทชันด้วยหน่วยวัดค่าความเป็นศูนย์กลาง รวมทั้งศึกษา

1.3.ขอบเขตการดำเนินงาน

1.3.1. ศึกษาและประเมินการทดสอบมิวเทชันบนระบบซอฟต์แวร์ Omise

1.3.2. เปรียบเทียบความสามารถในการจัดลำดับกรณีทดสอบมิวเทชันโดยใช้ค่าระดับความเป็นศูนย์กลาง ค่าระดับความเป็นศูนย์กลางโดยวัดจากค่าคั่นกลาง ค่าระดับความเป็นศูนย์กลางโดยวัดจากความใกล้ชิด และเพจแรงก์

1.3.3. รายงานผลการดำเนินงานทดสอบมิวเทชัน โดยแสดงคะแนนมิวเทชัน จำนวนมิวแทนต์ที่ถูกฆ่า จำนวนที่หมดเวลารอ จำนวนมิวแทนต์ที่มีชีวิต จำนวนมิวแทนต์ที่ไม่ครอบคลุม และจำนวนข้อผิดพลาด

1.4.ขั้นตอนการดำเนินงาน

1.4.1. ศึกษาค้นคว้าทฤษฎีและงานวิจัยที่เกี่ยวข้อง

1.4.2. ออกแบบวิธีการวิจัย

1.4.3. จัดลำดับความสำคัญการสร้างกรณีทดสอบ

1.4.4. สร้างกรณีทดสอบ

1.4.5. ดำเนินการทดสอบกรณีทดสอบ

1.4.6. ประเมินผลงานวิจัย วัดประสิทธิภาพของวิธีการเลือกตัวดำเนินการมิวเทชัน

1.4.7. เผยแพร่ผลงานวิชาการ

1.4.8. สรุปผลและเรียบเรียงวิทยานิพนธ์

1.5.ประโยชน์ที่คาดว่าจะได้รับ

1.5.1. ช่วยลดเวลาการทำงานของนักทดสอบในการสร้างมิวเทชัน

1.5.2. ช่วยจัดลำดับกรณีทดสอบเพื่อทดสอบมิวเทชัน

1.5.3. ช่วยปรับปรุงการเขียนโค้ดจากการใช้เทคนิคการเลือกมิวเทชัน

1.6.ผลงานที่ได้รับการตีพิมพ์

Supmak, S. & Limpiyakorn, Y. (2022). Prioritization of Mutation Test Generation with Centrality Measure. ICSTS 2022: International Conference on Software Testing Strategies, Amsterdam, Netherlands.

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

2.1.ทฤษฎีที่เกี่ยวข้อง

2.1.1. การทำเหมืองสื่อสังคม (Social Media Mining)

เป็นศาสตร์ที่ว่าด้วยแนวคิดและทฤษฎีที่อาศัยองค์ความรู้มากกว่าหนึ่ง (Interdisciplinary) หลักการพื้นฐานอัลกอริทึมล้ำสมัยมาทำการแสดง การวิเคราะห์และการสกัดรูปแบบและแนวโน้มที่สามารถดำเนินการได้จากข้อมูลที่มีอยู่ในสื่อสังคม [1] โดยทั่วไป ในโลกของสื่อสังคมจะมีองค์ประกอบดังต่อไปนี้ ปัจเจกบุคคล (Individuals) เอนทิตี (Entities) เช่น เนื้อหา (Content) ไซต์ (Sites) และเครือข่าย (Networks) เป็นต้น และการกระทำที่เกิดขึ้นระหว่างปัจเจกบุคคลกับเอนทิตี เพื่อการทำเหมืองเครือข่ายสังคมให้มีประสิทธิภาพจะต้องรวบรวมข้อมูลเกี่ยวกับปัจเจกบุคคลและเอนทิตี แล้วนำมาวัดการโต้ตอบของสองสิ่งและค้นหาแบบจำลองเพื่อทำความเข้าใจพฤติกรรมของมนุษย์

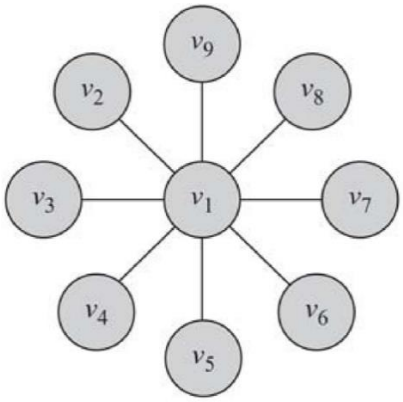
หน่วยวัดเครือข่าย (Network Measure) เป็นการวัดเพื่อหาว่าจุดต่อ (node) ที่มีความสำคัญในเครือข่ายนั้น ๆ มีการใช้รูปแบบใดในการแสดงการโต้ตอบของจุดต่อและจุดต่อใดมีส่วนที่เหมือนกันหรือมีความคล้ายกันของจุดต่อ ในการแสดงวิธีการจะต้องกำหนดการวัดความเป็นจุดศูนย์กลาง (Centrality) ซึ่งการหาค่าความเป็นศูนย์กลาง มีวิธีการดังต่อไปนี้

- ค่าระดับความเป็นศูนย์กลาง (Degree Centrality) จะเรียงลำดับความสำคัญจากจำนวนเส้นเชื่อม (Degree) ของจุดต่อในการวัดค่าระดับความเป็นศูนย์กลาง หากจุดต่อใดมีจำนวนเส้นเชื่อมมากที่สุดภายในเครือข่าย จะหมายถึงจุดต่อนั้นมีอิทธิพลมาก การคำนวณค่าความเป็นศูนย์กลาง ดังสมการ (1)

$$C_d(v_i) = d_i \quad (1)$$

โดย d_i หมายถึง จำนวนเส้นเชื่อมของจุดต่อ v_i

ภาพที่ 1 แสดงตัวอย่างกราฟค่าระดับความเป็นศูนย์กลางของจุดต่อ v_1 ที่มีค่าความเป็นศูนย์กลาง วัดจากเส้นเชื่อมเท่ากับ 8



ภาพที่ 1 ตัวอย่างกราฟค่าระดับความเป็นศูนย์กลาง [1]

- ค่าความเป็นศูนย์กลางโดยวัดจากการคั่นกลาง (Betweenness Centrality) เป็นการพิจารณา ถึงความสำคัญของจุดต่อหนึ่งที่เชื่อมโยงกับอีกจุดต่อหนึ่งในเครือข่าย เพื่อวัดจำนวนระยะทางที่สั้นที่สุดของคู่จุดต่อที่พิจารณาเทียบกับเส้นทางที่สั้นที่สุดทั้งหมดระหว่างจุดต่อทั้งสอง หากค่าคั่นกลางสูง หมายถึงจุดต่อนั้นมีความสำคัญในการเชื่อมต่อกับกลุ่มอื่นภายในเครือข่ายเดียวกัน ดังสมการ (2)

$$C_b(v_i) = \sum_{s \neq t \neq v_i} \frac{\sigma_{st}(v_i)}{\sigma_{st}} \qquad (2)$$

σ_{st} หมายถึง จำนวนของระยะทางที่สั้นที่สุด (Shortest Paths) ที่เป็นไปได้จากจุดต่อ s ไป t

$\sigma_{st}(v_i)$ หมายถึง จำนวนของระยะทางที่สั้นที่สุดที่เป็นไปได้จากจุดต่อ s ไป t ที่ลากผ่าน v_i

- ค่าความเป็นศูนย์กลางโดยวัดจากความใกล้ชิด (Closeness Centrality) เป็นการหาจุดต่อหนึ่งจุดต่อที่มีความสัมพันธ์ใกล้ชิดกับจุดต่อต่าง ๆ จากระยะทางที่สั้นที่สุดที่ใช้ในการเดินทางที่อยู่ภายในเครือข่ายเดียวกัน หากจุดต่อใดมีค่าความใกล้ชิดมากที่สุด หมายถึงจุดต่อนั้นมีการเชื่อมโยงกันอย่างทั่วถึงกับจุดต่ออื่น ๆ ดังสมการ (3)

$$C_c(v_i) = \frac{1}{\bar{l}_{v_i}} \qquad (3)$$

โดย $\bar{l}_{v_i} = \frac{1}{n-1} \sum_{v_j \neq v_i} l_{i,j}$ หมายถึง จุดต่อ v_i ของค่าเฉลี่ยของระยะทางที่สั้นที่สุดในการเข้าถึง (Shortest Path Length) ที่ไปยังจุดต่ออื่น

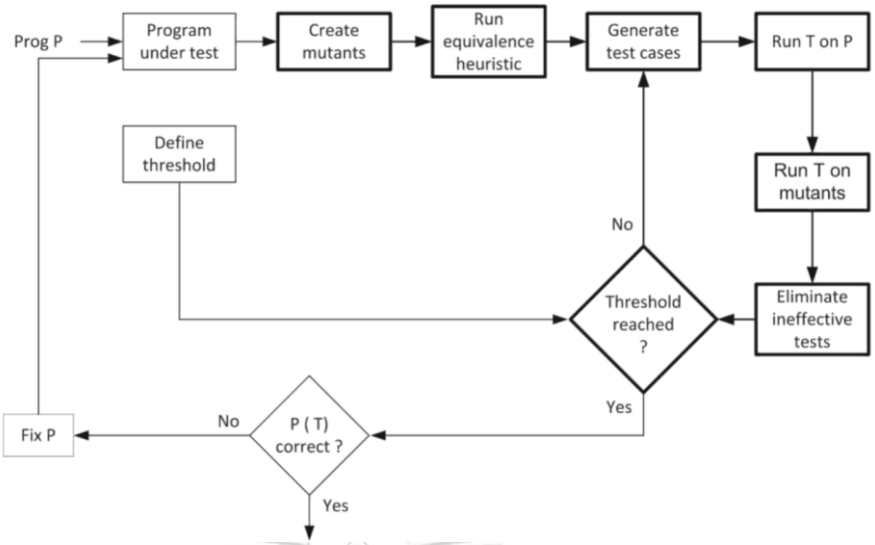
- เพจแรงก์ (PageRank) นำเสนอโดย Brin and Page [2] ในปี 1998 โดยอัลกอริทึมที่ถูกพัฒนา ถูกนำมาใช้ในการจัดอันดับการค้นหาใน Google Search เพื่อตัดสินใจว่าจะแสดงผลลัพธ์ใดที่ด้านบนสุดของรายการการค้นหา อย่างไรก็ตาม โหนดที่มีค่าความเป็นศูนย์กลางสูง จะแบ่งค่าความเป็นศูนย์กลาง โดยส่งผ่านจำนวนลิงก์ที่ออกจากโหนดทั้งหมด เพื่อให้โหนดเพื่อนบ้านที่เชื่อมต่อแต่ละรายได้รับเศษส่วนของค่าศูนย์กลางของโหนดต้นทาง ดังสมการ (4)

$$C_p(v_i) = \alpha \sum_{j=1}^{n} A_{i,j} \frac{C_p(v_i)}{d_j^{out}} + \beta \quad (4)$$

d_j^{out} หมายถึง มีค่าไม่เท่ากับศูนย์

2.1.2. การทดสอบมิวเทชัน (Mutation Testing)

การทดสอบมิวเทชัน เป็นวิธีการอย่างหนึ่งในการประเมินประโยชน์ที่ต้องการจากชุดกรณีทดสอบ [3] การทดสอบมิวเทชัน เป็นการประเมินซอฟต์แวร์ โดยมีเป้าหมายเพื่อทำให้มิวแทนต์ (Mutant) แต่ละตัวแสดงพฤติกรรมที่แตกต่างกัน เมื่อพบมิวแทนต์ มิวแทนต์จะถือว่าตายแล้วและไม่จำเป็นต้องอยู่ในขั้นตอนการทดสอบอีกต่อไป เนื่องจาก มิวแทนต์ที่แสดงจะถูกตรวจพบโดยการทดสอบเดียวกันกับที่ฆ่า ที่สำคัญไปกว่านั้น มิวแทนต์มีคุณสมบัติตรงตามข้อกำหนดในการระบุกรณีทดสอบที่มีประโยชน์ วิธีการที่จะนำไปสู่ความสำเร็จในการทดสอบมิวเทชัน คือตัวดำเนินการของมิวเทชัน ซึ่งถูกออกแบบมาสำหรับการเขียนโปรแกรม ข้อกำหนด หรือภาษาที่ใช้ในการเขียนโปรแกรม มิวเทชันในโปรแกรม อักขระที่ไม่ถูกต้อง คือผิดจากวากยสัมพันธ์และคอมไพเลอร์จะตรวจจับได้ สิ่งเหล่านี้ เรียกว่ามิวเทชันที่พบก่อนกำหนดและไม่ควรถูกสร้างขึ้นหรือควรทิ้งทันที บางครั้ง ที่พบมิวเทชันเล็กน้อยสามารถฆ่าได้เกือบทุกกรณีทดสอบ มิวแทนต์บางตัวมีฟังก์ชันเทียบเท่ากับโปรแกรมดั้งเดิม กล่าวคือ มิวแทนต์จะให้ผลลัพธ์เหมือนกับโปรแกรมดั้งเดิมเสมอ ดังนั้น จึงไม่มีกรณีทดสอบใดที่จะสามารถฆ่ามิวแทนต์ได้ มิวเทชันสมมูล แสดงถึงข้อกำหนดในการทดสอบที่เป็นไปไม่ได้ ขั้นตอนการทดสอบมิวเทชันอธิบายได้ ดังภาพที่ 2 แสดงถึงกระบวนการการทดสอบมิวเทชัน [4] โดยกล่องกรอบสีเข้มแสดงขั้นตอนที่เป็นอัตโนมัติ (automation) ส่วนกล่องที่เหลือแสดงขั้นตอนที่ทำมือ (manual)



ภาพที่ 2 กระบวนการการทดสอบมิวเทชัน [4]

องค์ประกอบหนึ่งที่สำคัญสำหรับการทดสอบมิวเทชัน คือ ตัวดำเนินการมิวเทชัน (Mutation Operator) เป็นกฎที่ระบุรูปแบบวากยสัมพันธ์ของอักขระที่สร้างจากไวยากรณ์ ตัวดำเนินการมิวเทชันมักใช้กับอักขระ แต่ยังสามารถนำไปประยุกต์ใช้กับไวยากรณ์ ดังนั้น ส่วนที่สำคัญที่สุดของการนำมิวเทชันไปประยุกต์ใช้ คือการออกแบบตัวดำเนินการมิวเทชันที่เหมาะสม ชุดของตัวดำเนินการที่ออกแบบมาอย่างดีอาจส่งผลให้มีการทดสอบที่ทรงพลังมาก แต่หากออกแบบมาไม่ดีอาจส่งผลให้การทดสอบไม่มีประสิทธิภาพ ตัวอย่างเช่น เครื่องมือเชิงพาณิชย์ ในการทำให้เกิดผลมิวเทชัน (Implements Mutation) แต่เปลี่ยนแปลงเพียงเพรดิเคต (Predicates) เป็นจริงและเท็จเท่านั้นจะเป็นวิธีการที่มีราคาแพงในการทำให้ครอบคลุมทุกสาขา นอกจากนี้ ตัวอย่างเช่น ข้อมูลเข้าที่ถูกต้อง (Invalid Input) อาจเป็นคำขอธุรกรรมจากผู้ใช้งานที่เข้าสู่ระบบอย่างถูกต้อง กรณีที่ไม่ถูกต้องอาจเป็นคำขอธุรกรรมเดียวกันจากผู้ใช้งานที่ไม่ได้เข้าสู่ระบบ หากอักขระที่แสดง G 25 08.01.90 เป็นอักขระที่ถูกใช้ตามไวยากรณ์ มิวเทชันที่ถูกต้องสองรายการอาจเป็นไปดังภาพที่ 3 แสดงถึงอักขระแรกที่เป็นอักขระที่ถูกใช้ตามไวยากรณ์ ส่วนภาพที่ 4 แสดงมิวเทชันที่ไม่ถูกต้อง

B 25 08.01.90
G 43 08.01.90

ภาพที่ 3 มิวเทชันที่ถูกต้อง [4]

12 25 08.01.90
G 25 08.01

ภาพที่ 4 มิวเทชันที่ไม่ถูกต้อง [4]

สำหรับการทดสอบมิวเทชันโดยวิธีคัดเลือก (Selective Mutation) เป็นการอธิบายกลยุทธ์ของการใช้ตัวดำเนินการมิวเทชันที่มีประสิทธิภาพ ส่วนประสิทธิผลได้รับการประเมินดังนี้ หากการทดสอบที่สร้างขึ้นโดยเฉพาะ เพื่อฆ่ามิวเทชันที่สร้างขึ้นด้วยตัวดำเนินการมิวเทชัน O_i ยังฆ่ามิวเทชันที่สร้างขึ้นโดยตัวดำเนินการมิวเทชัน O_j ที่มีความน่าจะเป็นสูงมาก ดังนั้น ตัวดำเนินการมิวเทชัน O_i จะมีประสิทธิภาพมากกว่า O_j

ผู้วิจัยได้ข้อสรุปว่าชุดของตัวดำเนินการมิวเทชันด้วย Fortran-77 (ระบบ Mothra) แต่ผลลัพธ์ถูกปรับให้เข้ากับ Java ตัวดำเนินการที่ถูกกำหนดไว้ในตารางที่ 1 คือ ตัวดำเนินการที่ถูกใช้ตลอดในระบบ Mothra [4]

ตารางที่ 1 ตัวดำเนินการที่ถูกใช้ตลอดในระบบ Mothra [4]

ตัวดำเนินการมิวเทชัน	คำอธิบาย
Absolute Value Insertion (ABS)	แก้ไขนิพจน์แต่ละนิพจน์ (และนิพจน์ย่อย) โดยฟังก์ชัน abs() negAbs() และ failOnZero()
Arithmetic Operator Replacement (AOR)	แทนที่ตัวดำเนินตัวใดตัวหนึ่ง (+,-,*,/,**และ%) ด้วยตัวดำเนินการอื่นแต่ละตัว นอกจากนี้ ให้แทนที่ด้วยตัวดำเนินการมิวเทชันพิเศษ leftOp, rightOp และ mod
Relation Operator Replacement (ROR)	แทนที่แต่ละตัวดำเนินการเชิงสัมพันธ์ (<,<=,>,>=,==,!=) โดยตัวดำเนินการอื่น ๆ และโดย falseOp และ trueOp
Condition Operator Replacement (COR)	แทนที่ตัวดำเนินแต่ละตัว (and-&&, or-k และไม่มีการประเมินตามเงื่อนไข-&, หรือไม่มีการประเมินตามเงื่อนไข-j, และไม่เทียบเท่า-^) ด้วยตัวดำเนินการอื่น ๆ และโดย falseOp, trueOp, leftOp และ rightOp
Shift Operator Replacement (SOR)	แทนที่ตัวดำเนินการตัวใดตัวหนึ่ง (<<,>> และ >>>) ด้วยตัวดำเนินการอื่นแต่ละตัว และโดย leftOp
Logical Operator Replacement (LOR)	แทนที่การเกิดขึ้นของตัวดำเนินการระดับบิตแต่ละตัว และโดย leftOp และ rightOp
Assignment Operator Replacement (ASR)	แทนที่ตัวดำเนินการด้วย (=,+=,-=,*=,/=,%=,&=,--=,^=,<<=,>>=,>>>=)
Unary Operator Insertion (UOI)	แทรกตัวดำเนินการแต่ละตัว (+,-,!, และ ~) ก่อนแต่ละนิพจน์ที่ถูกประเภท
Unary Operator Deletion (UOD)	ลบตัวดำเนินการเอกพจน์แต่ละตัว (+,-,!, และ ~)
Scalar Variable Replacement (SVR)	แทนที่แต่ละรายการด้วยตัวแปรอื่น ๆ ทุกประเภทที่เหมาะสมซึ่งประกาศไว้ในขอบเขตงานปัจจุบัน
Bomb Statement Replacement (BSR)	แทนที่แต่ละคำสั่งด้วยฟังก์ชัน Bomb() พิเศษ

2.2.งานวิจัยที่เกี่ยวข้อง

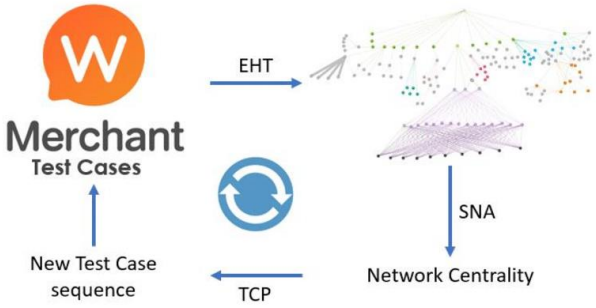
2.2.1. Automatic Test Case Generation and Optimization Based on Mutation Testing

งานวิจัยโดย Y. Du และคณะ [5] ได้นำเสนอวิธีการสร้างชุดกรณีทดสอบคุณภาพสูง โดยมีจุดประสงค์ในการสร้างกรณีทดสอบ จากการเสนอตัวดำเนินการมิวเทชันด้วยวิธีการทดสอบมิวเทชันแบบการเลือก ซึ่งเป็นเทคนิคในการลดจำนวนข้อผิดพลาดของโปรแกรม สำหรับการทดสอบมิวเทชันวิธีการดังกล่าว จะทำการเลือกตัวดำเนินการจาก 19 ตัวดำเนินการให้เหลือเพียง 5 ตัวดำเนินการในการทดสอบ กรณีทดสอบย่อยนี้แสดงค่าการแปรผันเฉลี่ยมากกว่า 95 % ซึ่งคะแนนที่ได้นำไปสร้างกรณีทดสอบ ที่รวมการทดสอบมิวเทชันด้วยขั้นตอนวิธีเชิงพันธุกรรม (Genetic Algorithm) เมื่อเปรียบเทียบขั้นตอนวิธีการและเครื่องมือเสร็จจะได้ชุดกรณีทดสอบที่มีความครอบคลุมกรณีทดสอบสูงและมีคะแนนมิวเทชันที่สูงขึ้น ผู้วิจัยสรุปได้ว่าชุดทดสอบเพียงพอต่อการทดสอบ

2.2.2. GUI Test Case Prioritization using Social Network Analysis

งานวิจัยโดย C.Maitrikul และ Y. Limpiyakorn [6] ได้นำเสนอการวัดค่าความเป็นศูนย์กลาง โดยมีจุดประสงค์เพื่อประเมินประสิทธิภาพการจัดอันดับความสำคัญ โดยเปรียบเทียบความสำคัญของแต่ละฟังก์ชันในเครือข่ายที่ประเมิน โดยการวัดการวิเคราะห์เครือข่ายสังคมและใช้เพื่อจัดอันดับความสำคัญของกรณีทดสอบ ซึ่งการวัดได้แก่ ค่าความเป็นศูนย์กลางโดยวัดจากการคั่นกลาง ค่าความเป็นศูนย์กลางโดยวัดจากความใกล้ชิด เพจแรงก์ ในการทดสอบกรณีทดสอบเพื่อค้นหาวิธีที่ดีที่สุดที่สามารถค้นหาข้อผิดพลาดได้เร็วที่สุดในรอบการทดสอบ จากนั้น บันทึกข้อมูลการทดสอบ 10 รอบตลอดระยะเวลาการทดสอบ สรุปได้ว่าค่าความเป็นศูนย์กลางเป็นหนึ่งในวิธีการที่สำคัญในการวิเคราะห์เครือข่ายสังคมที่ซับซ้อน ในงานวิจัยผู้วิจัยนำการวัดค่าศูนย์กลางมาใช้เพื่อจัดลำดับความสำคัญทั้งกรณีทดสอบที่แก้ไขและกรณีทดสอบใหม่ในระบบขนาดใหญ่ มีกรณีทดสอบมากกว่า 100 กรณีสำหรับทั้งฟังก์ชัน ในระบบ Wongnai RMS ที่ปกติจะมีการทดสอบเป็นรายสัปดาห์ โดยไม่มีการจัดลำดับความสำคัญ การทดลองได้ดำเนินการเพื่อประเมินประสิทธิภาพการจัดลำดับความสำคัญโดยปรียบเทียบความสำคัญของแต่ละหน้าที่ในเครือข่าย โดยคัดเลือกมาตรการดังนี้ ค่าความเป็นศูนย์กลางโดยวัดจากค่าคั่นกลาง ค่าความเป็นศูนย์กลางโดยวัดจากความใกล้ชิด ค่าความเป็นศูนย์กลางโดยวัดจากเวกเตอร์ลักษณะเฉพาะ และอันดับเพจ เมื่อผลลัพธ์ของแต่ละการวัดได้รับการจัดการแล้ว ผู้วิจัยทำการทดสอบจากกรณีทดสอบเพื่อหาวิธีที่ดีที่สุดในการหาข้อผิดพลาดได้เร็วที่สุด จากนั้น บันทึกข้อมูล 10 รอบของการทดสอบ เพื่อค้นหาแนวทางที่ดีที่สุดที่เหมาะสมที่สุดกับ

กรณีทดสอบ ผลงานวิจัยสรุปว่า ค่าความเป็นศูนย์กลางโดยวัดจากค่าคั่นกลางมีประสิทธิภาพมากที่สุด ถัดมาเป็นค่าความเป็นศูนย์กลางโดยวัดจากค่าเวกเตอร์เฉพาะ อย่างไรก็ตาม ไม่พบระดับความเบี่ยงเบนที่มีนัยยะสำคัญระหว่างการวัดทั้งสี่ เนื่องจากขนาดของข้อมูล ภาพรวมของการดำเนินงานวิจัยดังกล่าว แสดงดังภาพที่ 5



ภาพที่ 5 ขั้นตอนการดำเนินงานวิจัย [6]

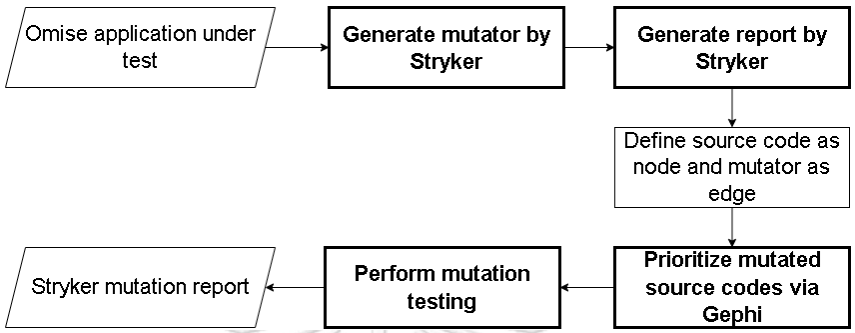
2.2.3. Social Network Analysis in Software Testing to Categorize Unit Test Cases Based on Coverage Information

งานวิจัยโดย N. Koochakzadeh และ R. Alhajj [7] ได้นำเสนอวิธีการประยุกต์ใช้เทคนิคการวิเคราะห์เครือข่ายทางสังคม เพื่อสร้างประเภทของกรณีทดสอบให้ครอบคลุมตามความต้องการ กราฟเครือข่ายที่ถูกสร้าง ประกอบด้วยชุดกรณีทดสอบ หรือกรณีทดสอบที่กำหนดให้เป็นโหนดและเชื่อมความสัมพันธ์ของโหนดไว้ด้วยความต้องการ ผู้วิจัยค้นพบคุณภาพของกรณีทดสอบจากการวัดความเกี่ยวพันธ์กันภายใน (cohesion) แพ็กเกจ วัดความสัมพันธ์ระหว่าง (coupling) แพ็กเกจ และสุดท้ายการเปรียบเทียบกับแพ็กเกจต้นแบบจากการพัฒนาการทดสอบ ซึ่งผลลัพธ์ที่ได้จากการทดลองคือเทคนิคที่สามารถจัดประเภทของกรณีทดสอบได้อัตโนมัติโดยการปรับปรุงคุณภาพของแพ็กเกจ

บทที่ 3

แนวคิดและวิธีการวิจัย

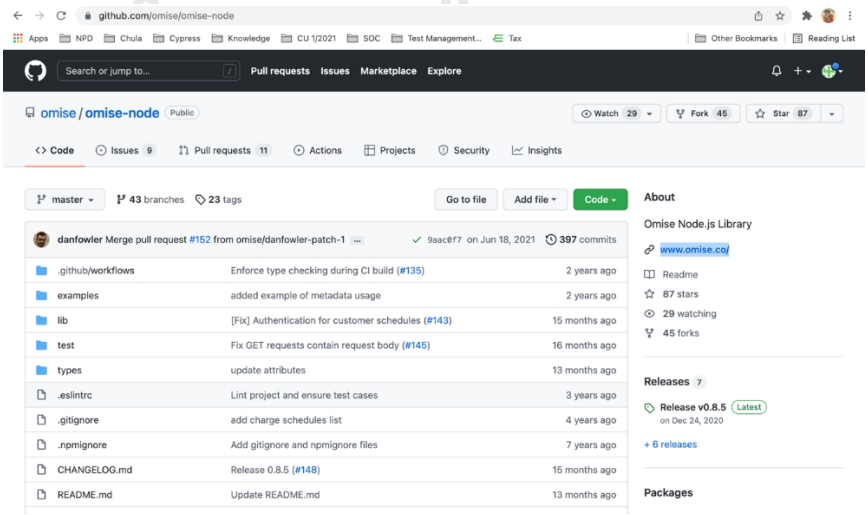
ภาพรวมการดำเนินงานวิจัย ประกอบด้วยขั้นต่าง ๆ ดังแสดงภาพที่ 6 แสดงภาพรวมของการดำเนินงานวิจัย



ภาพที่ 6 ขั้นตอนการดำเนินงานวิจัย

3.1.การเตรียมข้อมูลทดสอบ

โครงการซอฟต์แวร์ Omise [8] (ภาพที่ 7) ถูกพัฒนาขึ้นมาด้วยภาษาจาวาสคริปต์ โดยคุณ Jun Hasegawa และ อิศราดร หะริณสุต รวมถึงทีมงาน เป็นระบบที่ช่วยในการเชื่อมต่อระบบภายในเครือข่ายของเราให้สามารถสื่อสารไปยังเครือข่ายปลายทางที่ให้บริการในส่วนของการชำระเงินต่าง ๆ มีบริการจัดการการทำงานที่สรุปข้อมูลทุกอย่างให้อยู่ในหน้าจอเดียว (dashboard) และ REST API เป็นอินเทอร์เฟซที่ระบบคอมพิวเตอร์สองระบบใช้เพื่อแลกเปลี่ยนข้อมูลผ่านอินเทอร์เน็ตร่วมกันได้อย่างปลอดภัย



ภาพที่ 7 ซอร์ซโค้ดจากระบบโอมิเซะ [8]

3.2.การสร้างมิวแทนต์

เครื่องมือ Stryker Mutator [9] จะสร้างมิวแทนต์ขึ้นอยู่กับไวยากรณ์ในการเขียนซอร์ซโค้ดแต่ละฟังก์ชัน รวมถึงตัวดำเนินการมิวเทชัน (Mutation Operator) [10] ที่ถูกกำหนดไว้ในตารางที่ 2 คือ แต่ละภาษาก็จะมีการสร้างตัวดำเนินการมิวเทชันขึ้นมาเพื่ออธิบายถึงลักษณะเฉพาะในแต่ละมิวแทนต์ โดยเริ่มต้นจะใช้คำสั่ง stryker run ในโครงการ Omise เพื่อให้ Stryker ทำการสแกนซอร์ซโค้ดทั้งหมดหากไม่พบกรณีทดสอบที่สามารถฆ่ามิวแทนต์ได้ จะทำการสร้างมิวแทนต์ขึ้นมาจากซอร์ซโค้ด โดยบรรทัดสีแดงแสดงถึงซอร์ซโค้ดที่นำเข้า Stryker และบรรทัดสีเขียวแสดงถึงซอร์ซโค้ดกลายพันธุ์ ดังตัวอย่างภาพที่ 8

```
#249. [Survived] ObjectLiteral
lib/resources/Account.js:8:5
-       {
-         'key': config['secretKey'],
-         'omiseVersion': config['omiseVersion'],
-       }
+       {}
Ran all tests for this mutant.

#250. [Survived] StringLiteral
lib/resources/Account.js:9:21
-         'key': config['secretKey'],
+         'key': config[""],
Ran all tests for this mutant.

#251. [Survived] StringLiteral
lib/resources/Account.js:10:30
-         'omiseVersion': config['omiseVersion'],
+         'omiseVersion': config[""],
Ran all tests for this mutant.
```

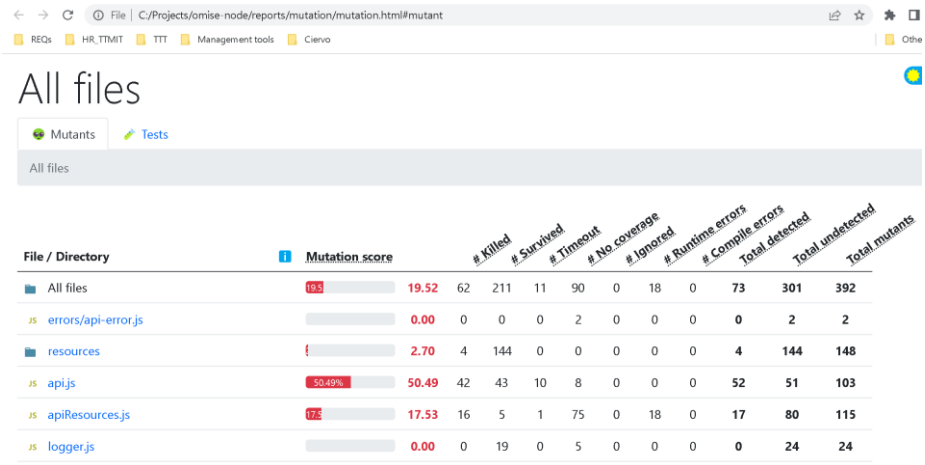
ภาพที่ 8 ตัวดำเนินการมิวเทชันที่พบในซอร์ซโค้ด Account.js

ตารางที่ 2 ตัวดำเนินการที่สนับสนุนจาก Stryker [10]

ตัวดำเนินการ	ซอร์ซโค้ด	ซอร์ซโค้ดกลายพันธุ์
Arithmetic Operator	a + b a - b a * b a / b a % b	a - b a + b a / b a * b a % b
Array Declaration	New Array (1, 2, 3, 4) [1, 2, 3, 4]	new Array () [ ]
Block Statement	Function tryTesting () { Console.log('Test');}	Function tryTesting ()
Boolean Literal	true false ! (a == b)	false true a == b
Conditional Expression	while (a>b)	while (false)
Equality Operator	a <= b, a >= b a == b a! = b	a >= b, a <= b a! = b a == b
Logical Operator	a && b a ?? b	a || b a && b
Method Expression	endWith () startsWith ()	startsWith () endsWith ()
Object Literal	{foo: 'bar'}	{ }
Optional Chaining	foo?.bar foo?.[1] foo?.()	foo.bar foo[1] foo()
Regex	^abc, abc$ [abc] \d (?=abc)	abc [^abc] \D (?!abc)
String Literal	"foo" (non-emprty) s"foo ${bar}" (string interpolation)	" " s""
Unary Operator	+a	-a
Update Operator	a++ ++a	a-- --a

3.3.การสร้างรายงานสรุปผลด้วย Stryker

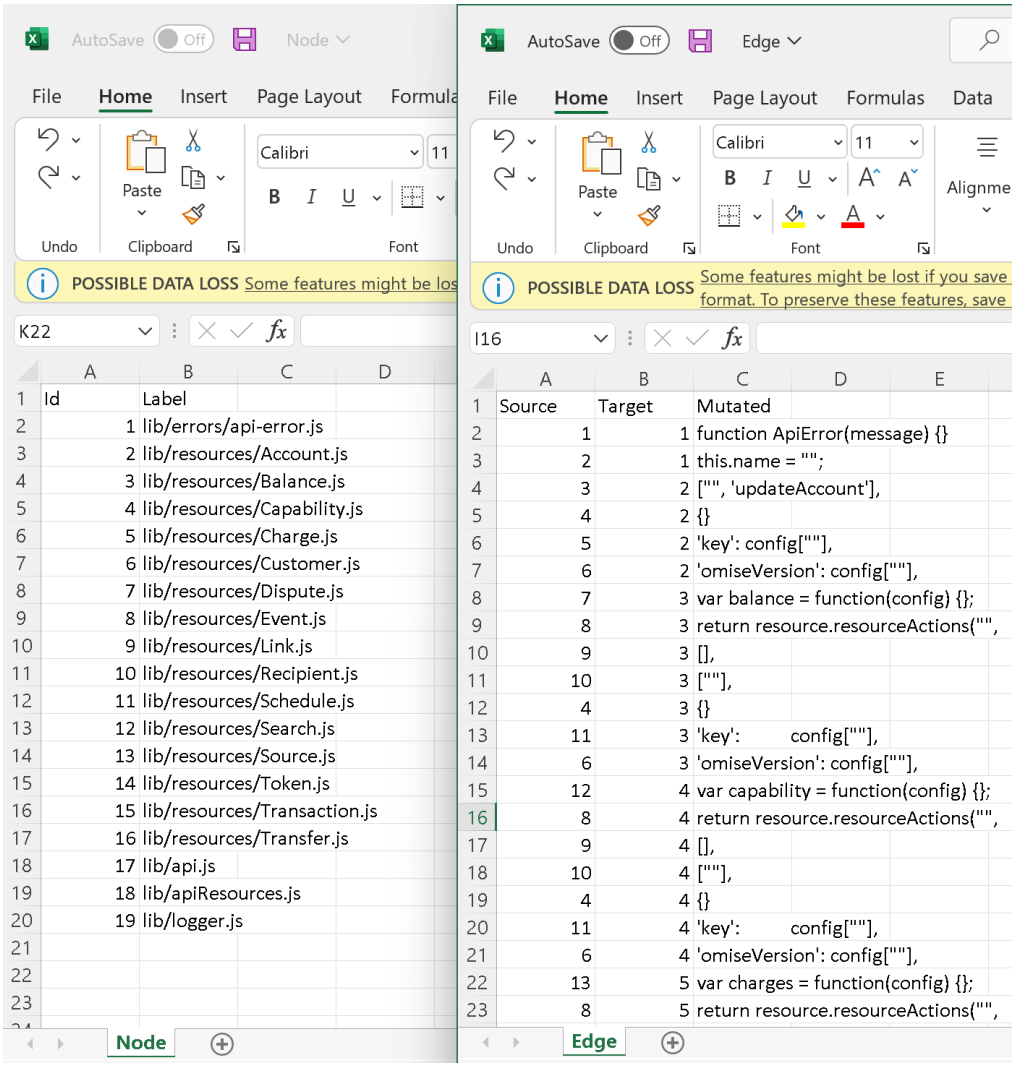
Stryker หลังจากที่ทำการสแกนซอร์ซโค้ดทั้งหมดและสร้างมิวแทนต์เสร็จสิ้น จะทำการสร้างรายงานสรุปผลออกมาเป็นไฟล์ html ภายในโครงการ เพื่อนำข้อมูลไปใช้ในการประมวลผลต่อไป โดยสนใจที่ตัวซอร์ซโค้ด และซอร์ซโค้ดกลายพันธุ์ ดังภาพที่ 9



ภาพที่ 9 รายงานสรุปผลด้วย Stryker

3.4.การกำหนดโหนดและเส้นเชื่อม

จากภาพที่ 9 ผู้วิจัยทำการเก็บข้อมูลซอร์ซโค้ดกลายพันธุ์ทั้งหมดที่อยู่ภายใต้คอมลัมน์ไฟล์/สารบบ (File/Directory) โดยเริ่มเก็บจากไฟล์บนลงล่างตามลำดับ จากนั้นนำข้อมูลที่เก็บวางลงในไฟล์ CSV ทั้งสองไฟล์ โดยที่ ไฟล์ Node.csv ประกอบไปด้วย Id และ Label ส่วนไฟล์ Edge.csv ประกอบไปด้วย Source Target และ Mutated เพื่อนำเข้าสู่โปรแกรม Gephi จำนวนที่นำเข้าทั้งหมด 301 รายการ มาจากจำนวนของซอร์ซโค้ดกลายพันธุ์ทั้งหมดที่เกิดขึ้น ดังภาพที่ 10



Node

Id	Label
1	lib/errors/api-error.js
2	lib/resources/Account.js
3	lib/resources/Balance.js
4	lib/resources/Capability.js
5	lib/resources/Charge.js
6	lib/resources/Customer.js
7	lib/resources/Dispute.js
8	lib/resources/Event.js
9	lib/resources/Link.js
10	lib/resources/Recipient.js
11	lib/resources/Schedule.js
12	lib/resources/Search.js
13	lib/resources/Source.js
14	lib/resources/Token.js
15	lib/resources/Transaction.js
16	lib/resources/Transfer.js
17	lib/api.js
18	lib/apiResources.js
19	lib/logger.js

Edge

Source	Target	Mutated
1	1	function ApiError(message) {}
2	1	this.name = "";
3	2	["", 'updateAccount'],
4	2	{}
5	2	'key': config[""],
6	2	'omiseVersion': config[""],
7	3	var balance = function(config) {};
8	3	return resource.resourceActions("",
9	3	[],
10	3	[""],
11	3	{}
12	3	'key': config[""],
13	3	'omiseVersion': config[""],
14	4	var capability = function(config) {};
15	4	return resource.resourceActions("",
16	4	[],
17	4	[""],
18	4	{}
19	4	'key': config[""],
20	4	'omiseVersion': config[""],
21	5	var charges = function(config) {};
22	5	return resource.resourceActions("",

ภาพที่ 10 การกำหนดโหนดและเส้นเชื่อมของโครงการ Omise

3.5.การจัดลำดับซอร์ซโค้ดกลายพันธุ์โดยใช้เครื่องมือ Gephi

ผู้วิจัยได้นำซอร์ซโค้ดทั้งหมดมาจัดลำดับเพื่อสร้างกรณีทดสอบด้วยหน่วยวัดค่าความเป็นศูนย์กลาง ประกอบด้วย ค่าระดับความเป็นศูนย์กลาง ค่าความเป็นศูนย์กลางโดยวัดจากค่าคั่นกลาง ค่าความเป็นศูนย์กลางโดยวัดจากความใกล้ชิด เพจแรงก์ โดยกำหนดให้ โหนด เป็นซอร์ซโค้ดที่จะนำเข้า Stryker และเส้นเชื่อม เป็นซอร์ซโค้ดกลายพันธุ์

3.6.การทำการทดสอบมิวเทชัน

ผลลัพธ์ที่ได้จากการจัดลำดับกรณีทดสอบ จะถูกนำมาเป็นเกณฑ์ในการเลือกออกแบบกรณีทดสอบ ดังภาพที่ 11 แสดงตัวอย่างกรณีทดสอบของซอร์ซโค้ด Account.js โดยลักษณะการเขียนกรณีทดสอบทั้งหมด จะต้องคำนึงถึงซอร์ซโค้ดกลายพันธุ์ที่เครื่องมือ Stryker สร้างออกมาเพื่อลดการเกิดช่องโหว่ หรือข้อบกพร่องที่จะเกิดขึ้นกับซอร์ซโค้ด

test_omise_account.js 1, M

test > JS test_omise_account.js > ...

```
var chai = require('chai');
var expect = chai.expect;
var config = require('./config');
var omise = require('../index')(config);
var testHelper = require('./testHelper');

describe.skip('Omise', function() {
  describe('#Account', function() {
    ▷ Execute Playwright Test
    it('should be able to retrieve an account', function(done) {
      testHelper.setupMock('account_retrieve');
      omise.account.retrieve(function(err, resp) {
        expect(resp.object, 'account');
        expect(resp.id, 'acct_123');
        expect(resp.email, 'test@omise.co');
        done(err);
      });
    });

    ▷ Execute Playwright Test
    it('should be able to update an account', function(done) {
      testHelper.setupMock('account_update');
      omise.account.updateAccount({
        'webhook_uri': 'https://omise-flask-example.herokuapp.com/webhook',
      }, function(err, resp) {
        expect(resp.object, 'account');
        expect(resp.id, 'acct_123');
        expect(resp.email, 'test@omise.co');
        expect(resp.webhook_uri, 'https://omise-flask-example.herokuapp.com/webhook');
        done(err);
      });
    });
  });
});
```

ภาพที่ 11 แสดงตัวอย่างกรณีทดสอบของซอร์ซโค้ด Account.js

3.7.การรายงานผลการทดสอบมิวเทชันด้วย Stryker

หลังจากได้กรณีทดสอบ ผู้วิจัยทำการทดสอบระบบโดยการใช้คำสั่ง stryker run เพื่อทดสอบมิวเทชัน โดยผลลัพธ์ที่ได้จากเครื่องมือ Stryker มีเมตริกและสถานะที่ถูกกำหนด [11] ไว้ในตารางที่ 3 คือ คำอธิบายผลลัพธ์จากการประเมินชุดกรณีทดสอบที่เกิดขึ้นกับซอร์ซโค้ดกลายพันธุ์ในโครงการ Omise ถ้าชุดกรณีทดสอบที่ออกแบบฆ่ามิวแทนต์ได้จะแสดงจำนวนคะแนนมิวเทชัน และจะแสดงจำนวนมิวเทชันที่ถูกฆ่า (killed) หากไม่สามารถฆ่ามิวแทนต์ได้จะแสดงจำนวนมิวเทชันที่รอดชีวิต (survived) หากกรณีทดสอบที่นำมาทดสอบนั้นได้คะแนนมิวเทชัน 100 % ของทั้งระบบจะสามารถสรุปได้ว่า ชุดกรณีทดสอบทำการฆ่ามิวเทชันได้ บางครั้ง ซอร์ซโค้ดที่ถูกสร้างขึ้นมีฟังก์ชันการทำงานที่เหมือนกัน ทำให้ชุดกรณีทดสอบส่งผลให้มีความสามารถในการฆ่ามิวเทชันที่เกิดขึ้นกับฟังก์ชันที่คล้ายกันได้ ดังภาพที่ 12 แสดงผลลัพธ์จากการสแกนมิวเทชัน โดยสร้างกรณีทดสอบที่อ้างอิงจากตัวดำเนินการที่เครื่องมือตรวจพบ

File / Directory	Mutation score		# Killed	# Survived	# Timeout	# No coverage	# Ignored	# Runtime errors	# Compile errors	Total detected	Total undetected	Total mutants
resources		3.38	5	143	0	0	0	0	0	5	143	148
Account.js	62.50%	62.50	5	3	0	0	0	0	0	5	3	8
Balance.js		0.00	0	7	0	0	0	0	0	0	7	7
Capability.js		0.00	0	7	0	0	0	0	0	0	7	7
Charge.js		0.00	0	17	0	0	0	0	0	0	17	17
Customer.js		0.00	0	16	0	0	0	0	0	0	16	16
Dispute.js		0.00	0	12	0	0	0	0	0	0	12	12
Event.js		0.00	0	8	0	0	0	0	0	0	8	8
Link.js		0.00	0	9	0	0	0	0	0	0	9	9
Recipient.js		0.00	0	12	0	0	0	0	0	0	12	12
Schedule.js		0.00	0	9	0	0	0	0	0	0	9	9
Search.js		0.00	0	7	0	0	0	0	0	0	7	7
Source.js		0.00	0	7	0	0	0	0	0	0	7	7
Token.js		0.00	0	9	0	0	0	0	0	0	9	9
Transaction.js		0.00	0	8	0	0	0	0	0	0	8	8
Transfer.js		0.00	0	12	0	0	0	0	0	0	12	12

ภาพที่ 12 สแกนมิวเทชันจากซอร์ซโค้ดของโครงการ Omise ที่มีกรณีทดสอบมารองรับมิวเทชัน

ตารางที่ 3 สถานะและเมตริกของมิวแทนต์ [11]

สถานะหรือเมตริก	**คำอธิบาย**
Mutation score	อัตราร้อยละของมิวแทนต์ทั้งหมดที่ตรวจพบ
Killed	จำนวนของกรณีทดสอบอย่างน้อยหนึ่งกรณีที่ล้มเหลวในขณะที่มิวแทนต์ถูกสร้าง
Survived	จำนวนของกรณีทดสอบทั้งหมดที่ไม่ล้มเหลวในขณะที่มิวแทนต์ถูกสร้าง
Timeout	จำนวนของการดำเนินการทดสอบที่ส่งผลให้หมดเวลาในขณะที่มิวแทนต์ถูกสร้าง
No coverage	จำนวนของกรณีทดสอบที่ไม่ครอบคลุมมิวแทนต์ที่ถูกสร้างและมิวแทนต์ที่ Survived
Ignored	จำนวนของมิวแทนต์ที่ไม่ได้ถูกทดสอบ
Runtime errors	จำนวนของการทดสอบที่ทำให้เกิดข้อผิดพลาด เช่น การทดสอบส่งค่า OutOfMemoryError หรือการที่ไม่สามารถวิเคราะห์โค้ดได้
Compile errors	จำนวนของมิวแทนต์ที่ทำให้เกิดข้อผิดพลาดในการแปลโปรแกรม (compile)
Total detected	จำนวนของผลรวมของจำนวนที่ฆ่า (killed) และจำนวนที่หมดเวลา (timeout)
Total undetected	จำนวนของผลรวมของจำนวนที่รอดชีวิต (survived) และจำนวนที่ไม่ครอบคลุมการทดสอบ (no coverage)
Total mutants	จำนวนของผลรวมของจำนวนที่สมบูรณ์ (valid) จำนวนที่ไม่สมบูรณ์ (invalid) และจำนวนที่ไม่ได้ถูกทดสอบ (ignored)
Valid	จำนวนของผลรวมของจำนวนที่ตรวจพบ (detected) และจำนวนที่ไม่สามารถตรวจพบ (undetected)
Invalid	จำนวนของผลรวมของจำนวนของการทดสอบที่ทำให้เกิดข้อผิดพลาด (runtime errors) และจำนวนของมิวแทนต์ที่ทำให้เกิดข้อผิดพลาดในการแปลโปรแกรม (compile errors)

บทที่ 4

การทดลองและผลการทดลอง

งานวิจัยนี้นำเสนอแนวทางการวิเคราะห์ค่าความเป็นศูนย์กลางเพื่อจัดลำดับการสร้างกรณีทดสอบมิวเทชัน โดยใช้วิธีการวิเคราะห์เครือข่ายสังคม และทฤษฎีกราฟมาใช้สำหรับวิเคราะห์ตัวดำเนินการมิวเทชันที่รวบรวมจากโครงการซอฟต์แวร์โลจิสติกส์ โดยใช้ Stryker เป็นเครื่องมือในการสร้างการกลายพันธุ์ที่เกิดกับซอร์ซโค้ดต้นฉบับ ภายใต้การพัฒนาในภาษาจาวาสคริปต์ที่สอดคล้องกับโครงการที่นำมาศึกษา และใช้ Gephi เป็นเครื่องมือในการศึกษาทฤษฎีของกราฟ รวมถึงใช้เพื่อการสร้างเครือข่ายสังคมของตัวแสดงที่ถูกระบุเป็นโหนดและเส้นที่เชื่อมโยงระหว่างโหนดเหล่านั้น ซึ่งแนวทางของการวิเคราะห์เครือข่ายภาพนั้นสามารถอธิบายข้อมูลเชิงลึกในการจัดลำดับการสร้างกรณีทดสอบมิวเทชันที่จำเป็นสำหรับนักทดสอบซอฟต์แวร์ ขั้นตอนการทดลองอธิบายในหัวข้อต่อไปนี้

4.1 เครื่องมือที่ใช้ในการทดลอง

ในการทดลองนี้ ผู้วิจัยได้เลือกใช้เฟรมเวิร์ก Stryker [9] ซึ่งเป็นซอฟต์แวร์โอเพนซอร์ซสำหรับการทดสอบมิวเทชัน ใช้เฟรมเวิร์กนี้สร้างตัวดำเนินการมิวเทชัน เพื่อที่จะประเมินชุดกรณีทดสอบที่มีอยู่ว่ามีช่องโหว่ หรือข้อบกพร่องที่นอกเหนือจากการทดสอบทั่วไปที่นักทดสอบได้ทดสอบ โดยสามารถทำการติดตั้งได้ฟรี โดยใช้คำสั่งของเฟรมเวิร์ก ดังตารางที่ 4 คือ ชุดคำสั่งในการติดตั้ง Stryker โดยชุดคำสั่งลำดับที่ 1 เป็นชุดคำสั่งสำหรับการติดตั้ง Stryker ลงบนเครื่อง สามารถรองรับทั้งในระบบปฏิบัติการ Windows และ Mac OS ซึ่งผู้วิจัยได้เลือกระบบปฏิบัติการ Windows เป็นระบบปฏิบัติการหลักในการทำงานวิจัยในครั้งนี้ ชุดคำสั่งลำดับที่ 2 คือชุดคำสั่งสำหรับการกำหนดคุณลักษณะต่าง ๆ ของเฟรมเวิร์ก เช่น รูปแบบของรายงาน เฟรมเวิร์กที่ใช้ในการทดสอบ เป็นต้น ชุดคำสั่งลำดับสุดท้าย ใช้เพื่อดำเนินการสร้างตัวดำเนินการประเภทต่าง ๆ จากซอร์ซโค้ดต้นแบบเพื่อใช้ในการประเมินชุดกรณีทดสอบ

ตารางที่ 4 ชุดคำสั่งของเครื่องมือ Stryker [9]

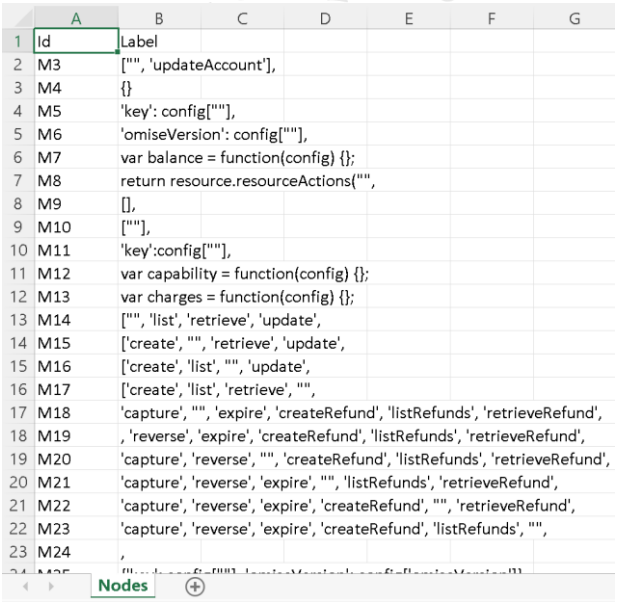
ลำดับ	ชุดคำสั่ง
1.	npm install -g stryker-cli
2.	stryker init
3.	stryker run

นอกจากนี้ ผู้วิจัยได้เลือกใช้โปรแกรม Gephi [12] เป็นซอฟต์แวร์โอเพนซอร์ซ สำหรับการสร้างกราฟ การวิเคราะห์กราฟและเครือข่าย โดยสามารถทำการติดตั้งและดาวน์โหลดได้ฟรี ซึ่งเวอร์ชันที่ผู้วิจัยใช้สำหรับการศึกษาและพัฒนา คือ Gephi version 9.0.6 โดยที่สามารถติดตั้งได้ทั้งในระบบปฏิบัติการ Windows และ Mac OS ซึ่งผู้วิจัยเลือกติดตั้งบนระบบปฏิบัติการ Windows เป็นหลักในการทำวิจัยในครั้งนี้

ในส่วนของการเขียนชุดกรณีทดสอบ ผู้วิจัยได้เขียนกรณีทดสอบที่ทำให้เป็นอัตโนมัติ (automate test case) ด้วยเฟรมเวิร์ก Mocha [13] เป็นเฟรมเวิร์กที่นิยมใช้อย่างมากในการทดสอบระบบที่พัฒนาขึ้นมาด้วยภาษาจาวาสคริปต์ ที่ทำงานบน Node.js และในเบราว์เซอร์ ลักษณะการทดสอบแบบอะซิงโครนัส (asynchronous testing) ทำได้ง่าย เนื่องจากเป็นแบบอะซิงโครนัส การทดสอบก็จะทดสอบเป็นลำดับ

4.2 การเตรียมชุดข้อมูลที่ใช้ในการทดลอง

ใช้คำสั่ง stryker run ภายในโครงการ Omise จากนั้น Stryker จะทำการสแกนซอร์ซโค้ดทั้งหมด เพื่อสร้างซอร์ซโค้ดกลายพันธุ์และสรุปผลที่เกิดขึ้น การเตรียมชุดข้อมูลเพื่อใช้ในการนำเข้าข้อมูลไปที่ Gephi จำเป็นต้องแปลงข้อมูลจากที่เฟรมเวิร์ก Stryker generate ออกมาเป็นไฟล์ CSV ซึ่งเป็นไฟล์ข้อมูลสองเสปรดชีตแยกกัน ประกอบด้วย ไฟล์ที่มีเสปรดชีต “Nodes” และ ไฟล์ที่มีเสปรดชีต “Edges” โดยไฟล์ “Nodes” จะมีลักษณะดังนี้ 1. ไฟล์ต้องประกอบไปด้วยคอลัมน์แรกที่ต้องตั้งชื่อว่า “Id” เพื่อสื่อถึงโหนดต่าง ๆ 2. ในส่วนของคอลัมน์อื่น ๆ นั้นผู้ใช้งานสามารถตั้งชื่อหรือกำหนดได้ตามลักษณะการใช้งาน ดังแสดงในภาพที่ 13

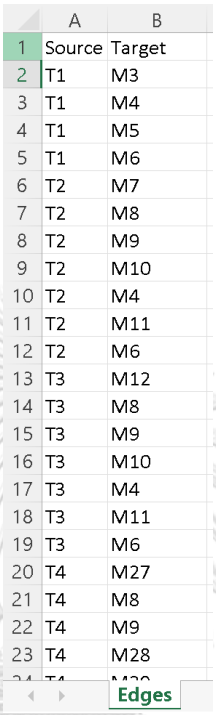


	A	B
1	Id	Label
2	M3	["", 'updateAccount'],
3	M4	{}
4	M5	'key': config[""],
5	M6	'omiseVersion': config[""],
6	M7	var balance = function(config) {};
7	M8	return resource.resourceActions("",
8	M9	[],
9	M10	[""],
10	M11	'key':config[""],
11	M12	var capability = function(config) {};
12	M13	var charges = function(config) {};
13	M14	["", 'list', 'retrieve', 'update',
14	M15	['create', "", 'retrieve', 'update',
15	M16	['create', 'list', "", 'update',
16	M17	['create', 'list', 'retrieve', "",
17	M18	'capture', "", 'expire', 'createRefund', 'listRefunds', 'retrieveRefund',
18	M19	, 'reverse', 'expire', 'createRefund', 'listRefunds', 'retrieveRefund',
19	M20	'capture', 'reverse', "", 'createRefund', 'listRefunds', 'retrieveRefund',
20	M21	'capture', 'reverse', 'expire', "", 'listRefunds', 'retrieveRefund',
21	M22	'capture', 'reverse', 'expire', 'createRefund', "", 'retrieveRefund',
22	M23	'capture', 'reverse', 'expire', 'createRefund', 'listRefunds', "",
23	M24	,

Nodes

ภาพที่ 13 ข้อมูลของโหนดบางส่วนในไฟล์ Nodes.csv

นอกจากนี้ ยังมีอีก 1 ไฟล์สำคัญ คือ ไฟล์ Edges.csv ที่มาเชื่อมต่อโหนดต่าง ๆ เพื่อที่จะสร้างกราฟและวิเคราะห์เครือข่าย ซึ่งไฟล์ Edges.csv ประกอบด้วยคอลัมน์ “Source” “Target” และ “Mutated” ดังภาพที่ 14



	A	B
1	Source	Target
2	T1	M3
3	T1	M4
4	T1	M5
5	T1	M6
6	T2	M7
7	T2	M8
8	T2	M9
9	T2	M10
10	T2	M4
11	T2	M11
12	T2	M6
13	T3	M12
14	T3	M8
15	T3	M9
16	T3	M10
17	T3	M4
18	T3	M11
19	T3	M6
20	T4	M27
21	T4	M8
22	T4	M9
23	T4	M28

Edges

ภาพที่ 14 ข้อมูลของเส้นเชื่อมบางส่วนในไฟล์ Edges.csv

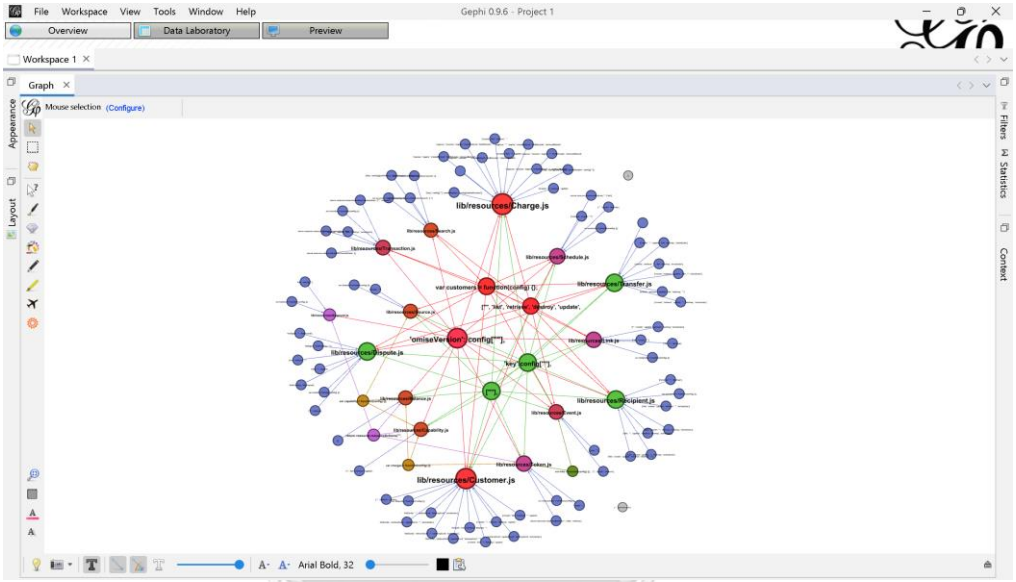
การแปลงข้อมูลประกอบด้วยสองขั้นตอน คือขั้นตอนแรกนำเข้าข้อมูลจากไฟล์ Nodes.csv โดยที่แต่ละโหนดจะมี “Id” ที่ไม่ซ้ำกัน ขั้นตอนที่สอง นำเข้าข้อมูลจากไฟล์ Edges.csv ซึ่งจะบอกถึงความสัมพันธ์ทั้งหมดที่เกิดระหว่างโหนดแต่ละ “Id” ก่อนการนำเข้าข้อมูล ต้องตรวจสอบให้แน่ใจว่าข้อมูลทั้งสองไฟล์ “มีปฏิสัมพันธ์” ซึ่งกันและกัน และ “Id” ที่ไม่ซ้ำกันนี้ถูกเรียกใช้อย่างสม่ำเสมอก่อนนำเข้าข้อมูล

4.3 การดำเนินการสร้างกราฟ

ทำการนำเข้าไฟล์ Nodes.csv และ Edges.csv ที่ได้เตรียมไว้ในข้อ 4.2 เข้าสู่โปรแกรม Gephi ขั้นตอนถัดไป คือ การสร้างกราฟและการวิเคราะห์เครือข่ายสังคม ประกอบด้วย

- คลิกที่แถบ “Overview” แสดงกราฟและแถบเมนูต่าง ๆ เลือกแถบเมนู “Appearance” เลือกสีและเลือกแถบย่อย Ranking ปรับขนาดของโหนดและเลือกแอดทริบิวต์ที่ต้องการนำเสนอเพื่อมาแสดงบนกราฟ SNA

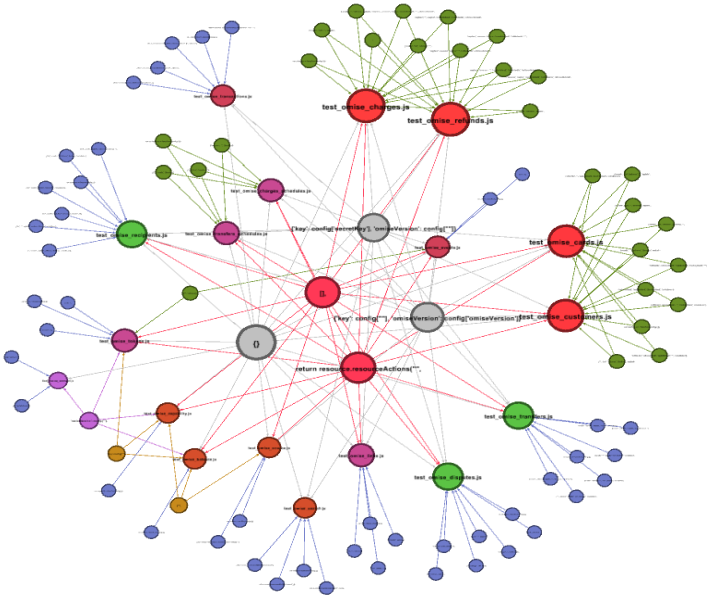
- เลือกแถบเมนู “Layout” เลือกค่าที่ต้องการนำเสนอ โดยผู้วิจัยเลือกใช้ “Fruchterman-Reingold layout” เนื่องจากชุดข้อมูลของโครงการมีขนาดเล็ก เพื่อให้กราฟกระจายตัวของข้อมูลในมุมมองที่สามารถเห็นข้อมูลได้อย่างชัดเจน นอกจากนี้ ผู้ใช้ยังสามารถเลือกใช้อัลกอริทึมอื่น ๆ ตามลักษณะของข้อมูลที่ต้องการอยากให้แสดงในรูปของกราฟได้อีกมากมาย
- เมื่อตั้งค่าต่าง ๆ ในแถบเมนู “Layout” เสร็จสิ้นแล้ว คลิกปุ่ม “Run” เพื่อสร้างกราฟเครือข่าย ดังภาพที่ 15



ภาพที่ 15 ตัวอย่างโปรแกรม Gephi และการแสดงกราฟ

จากการนำเสนอตามวิธีการข้างต้น ได้ทำการวิเคราะห์และปรับพารามิเตอร์ให้ตรงตามสิ่งที่ต้องการนำเสนอ ทำให้ผู้วิจัยได้พยายามทำการทดลองเพื่อสร้างกราฟเครือข่ายโดยใช้ค่าความเป็นศูนย์กลางทั้ง 4 แบบ ได้แก่ ค่าระดับความเป็นศูนย์กลาง ค่าระดับความเป็นศูนย์กลางโดยวัดจากค่าคั่นกลาง ค่าระดับความเป็นศูนย์กลางโดยวัดจากความใกล้ชิด และเพจแรงก์ มาจัดลำดับซอร์ซโค้ดกลายพันธุ์ โดยการวิเคราะห์ผ่านแบบจำลองกราฟ SNA ของค่าความเป็นศูนย์กลางทั้ง 4 ซึ่งได้ผลการทดลองดังต่อไปนี้

4.3.1 กราฟรูปแบบที่ 1 ค่าระดับความเป็นศูนย์กลาง



ภาพที่ 16 แบบจำลองกราฟ SNA ค่าระดับความเป็นศูนย์กลางของโครงการ Omise

ภาพที่ 16 แสดงผลจากการวัดค่าระดับความเป็นศูนย์กลาง โดยโหนดที่มีค่าระดับสูงที่สุดมีชื่อว่า test_omise_charges.js และ test_omise_refunds.js มีค่าเท่ากับ 17 หมายความว่า โหนดดังกล่าวมีการเชื่อมโยงออกจากตัวโหนดของตัวเองเท่ากับ 17 และไม่มีโหนดอื่น ๆ มาเชื่อมโยงหาตัวเอง เมื่อเปรียบเทียบกับโหนดที่มีชื่อว่า test_omise_account.js ที่มีค่าระดับต่ำที่สุดเท่ากับ 4 คือมีการเชื่อมโยงออกจากตัวโหนดของตัวเอง จากค่าที่วัดได้มาข้างต้นของทั้งสองโหนดนั้น นำไปดำเนินการทดสอบมิวเทชัน โดยขั้นแรก นำกรณีทดสอบและซอร์ซโค้ด test_omise_charges.js มาประเมินด้วยเครื่องมือ Stryker ได้ผลดังภาพที่ 17

File / Directory	Mutation score		# Killed	# Survived	# Timeout	# No coverage	# Ignored	# Runtime errors	# Compile errors	Total detected	Total undetected	Total mutants
All files	32.35%	32.35	97	192	24	61	0	18	0	121	253	392
errors/api-error.js		0.00	0	0	0	2	0	0	0	0	2	2
resources	10	10.81	16	132	0	0	0	0	0	16	132	148
api.js	59.22%	59.22	51	37	10	5	0	0	0	61	42	103
apiResources.js	45.36%	45.36	30	4	14	49	0	18	0	44	53	115
logger.js		0.00	0	19	0	5	0	0	0	0	24	24

ภาพที่ 17 คะแนนรวมของซอร์ซโค้ดลำดับที่ 1 ที่ไม่มีกรณีทดสอบมิวเทชัน

หลังจากใช้เครื่องมือ Stryker เพื่อประเมินชุดกรณีทดสอบ test_omise_charges.js ดังภาพที่ 18 สรุปผลคะแนนได้ร้อยละ 32.35 หมายความว่า ยังมีมิวแทนต์รอดชีวิตจากกรณีทดสอบ ดังตารางที่ 5 คือ การสร้างมิวแทนต์ด้วย Stryker ในกรณีทดสอบ test_omise_charges.js

```
it('should be able to create a charge', function (done) {
  testHelper.setupMock('charges_create');
  var charge = {
    'description': 'Charge for order 3947',
    'amount': '100000',
    'currency': 'thb',
    'capture': false,
    'card': tokenId,
  };
  omise.charges.create(charge, function (err, resp) {
    expect(resp.object, 'charge');
    chargeId = resp.id;
    expect(chargeId).to.match(/^chrg_test/);
    expect(resp.capture).be.false;
    expect(resp.paid).be.false;
    done(err);
  });
});

Execute Playwright Test
it('should be able to reverse a charge', function (done) {
  testHelper.setupMock('charges_reverse');
  omise.charges.reverse(chargeId, function (err, resp) {
    expect(resp.object, 'charge');
    var reversed = resp.reversed;
    reversed.should.be.true;
    done(err);
  });
});
```

ภาพที่ 18 กรณีทดสอบบางส่วนของ test_omise_charges.js จากการวัดค่าระดับความเป็นศูนย์กลางที่มีอิทธิพลมากที่สุด

ตารางที่ 5 การสร้างมิวแทนต์ด้วย Stryker ในซอร์ซโค้ด lib/resources/Charge.js จากการจัดลำดับของค่าระดับความเป็นศูนย์กลาง

ซอร์ซโค้ด	ซอร์ซโค้ดกลายพันธุ์
{'key': config['secretKey'], 'omiseVersion': config['omiseVersion']}	{}
{'key': config['secretKey'], 'omiseVersion': config['omiseVersion']}	{'key': config[""], 'omiseVersion': config['omiseVersion']}
{'key': config['secretKey'], 'omiseVersion': config['omiseVersion']}	{'key': config['secretKey'], 'omiseVersion': config[""]}

ผู้วิจัยทำการเพิ่มกรณีทดสอบ โดยให้ความสนใจที่มิวแทนต์ที่ถูกสร้างขึ้น ดังภาพที่ 19 โดยประกาศ “stub” ไว้ในบรรทัดที่ 143 และทำการเรียก “stub” จากซอร์ซโค้ดมาทำงานในกรณีทดสอบ ดังแสดงในบรรทัดที่ 141 ถึง 153

```
141   it('should be able to killed mutated (ObjectLiteral) in charges', function () {
142     testHelper.setupMock('charges_list');
143     const apiGetStub = sandbox.stub(api, 'get');
144     omise.charges.list();
145     expect(apiGetStub.withArgs(
146       {
147         key: '321',
148         omiseVersion: '2019-05-29',
149         path: '/charges',
150         body: undefined
151       }, undefined
152     ).calledOnce).to.be.true;
153   })
```

ภาพที่ 19 กรณีทดสอบที่ฆ่ามิวแทนต์ของซอร์ซโค้ด lib/resources/Charge.js จากการวัดค่าระดับความเป็นศูนย์กลาง

ผลลัพธ์ที่ได้จากการจัดลำดับด้วยการวัดค่าระดับความเป็นศูนย์กลาง ดังภาพที่ 20 ภาพรวมของคะแนนในการเลือกนำกรณีทดสอบลำดับที่ 1 มาทำการทดสอบ จากเดิมที่มีผลรวมของคะแนนเท่ากับ 32.35 เมื่อทำการทดสอบมิวเทชันและสามารถฆ่ามิวแทนต์ได้ในกรณีทดสอบของซอร์ซโค้ด lib/resources/Charge.js ดังภาพที่ 21 มีผลรวมของคะแนนเพิ่มขึ้นเป็น 33.16 มีผลต่างอยู่ที่ 0.81

File / Directory	Mutation score		# Killed	# Survived	# Timeout	# No coverage	# Ignored	# Runtime errors	# Compile errors	Total detected	Total undetected	Total mutants
All files	33.16%	33.16	100	189	24	61	0	18	0	124	250	392
errors/api-error.js		0.00	0	0	0	2	0	0	0	0	2	2
resources	12	12.84	19	129	0	0	0	0	0	19	129	148
api.js	59.22%	59.22	51	37	10	5	0	0	0	61	42	103
apiResources.js	45.36%	45.36	30	4	14	49	0	18	0	44	53	115
logger.js		0.00	0	19	0	5	0	0	0	0	24	24

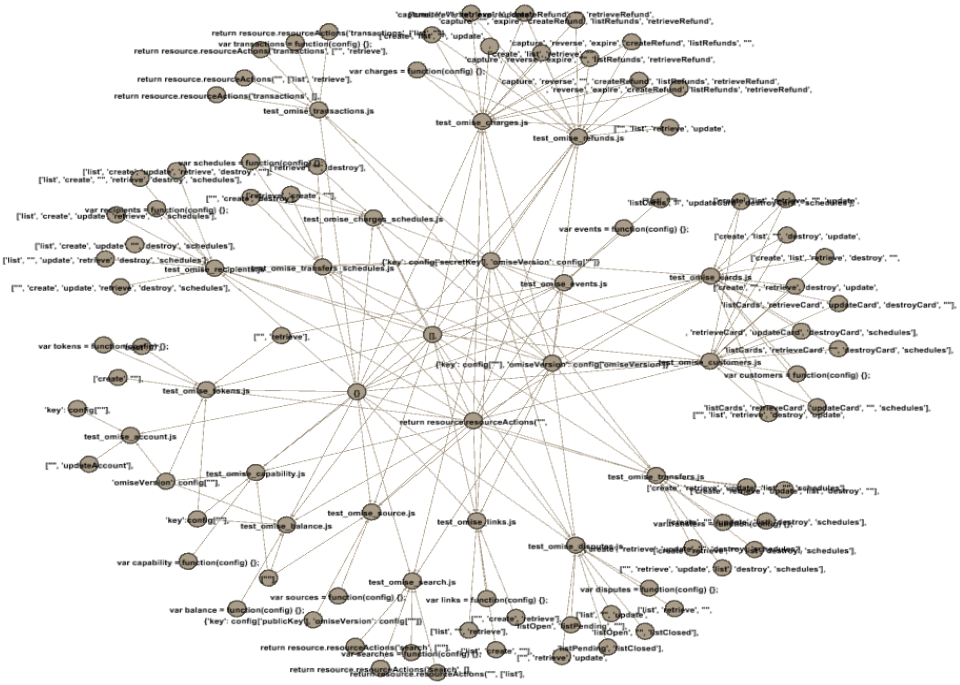
ภาพที่ 20 คะแนนรวมของซอร์ซโค้ดลำดับที่ 1 จากการวัดค่าระดับความเป็นศูนย์กลาง หลังจากทดสอบมิวเทชัน

File / Directory	Mutation score		# Killed	# Survived	# Timeout	# No coverage	# Ignored	# Runtime errors	# Compile errors	Total detected	Total undetected	Total mutants
Charge.js	76.47%	76.47	13	4	0	0	0	0	0	13	4	17

Killed (13) Survived (4)

ภาพที่ 21 คะแนนของซอร์ซโค้ด lib/resources/Charge.js ที่ฆ่ามิวแทนต์

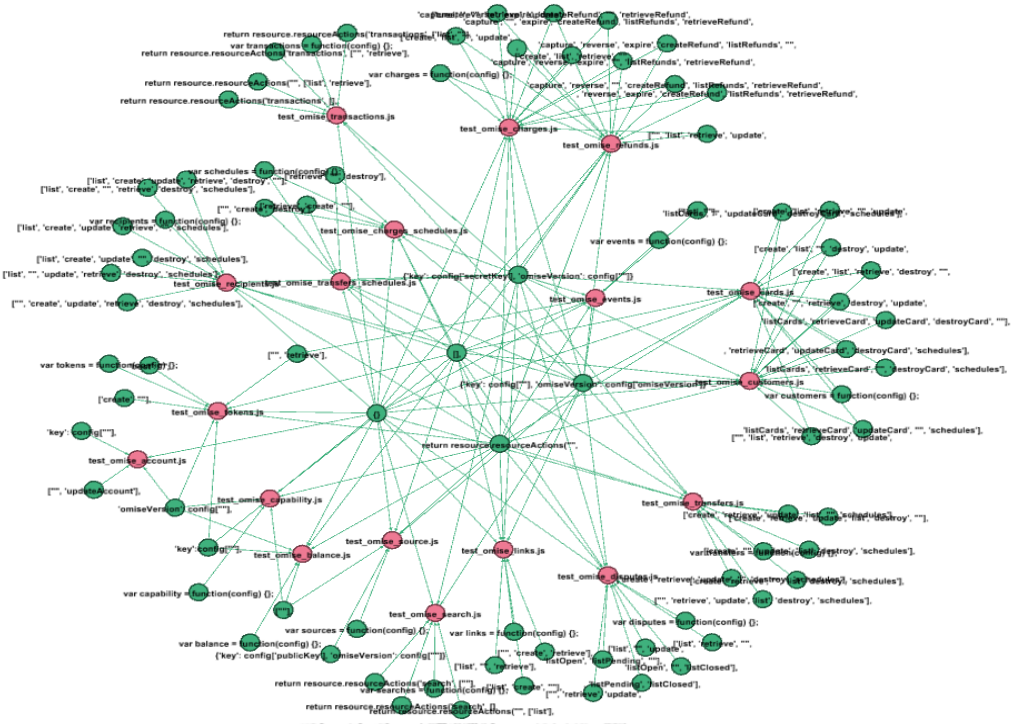
4.3.2 กราฟรูปแบบที่ 2 ค่าความเป็นศูนย์กลางโดยวัดจากค่าคั่นกลาง



ภาพที่ 22 ค่าความเป็นศูนย์กลางโดยวัดจากค่าคั่นกลาง

ภาพที่ 22 แสดงผลจากการวัดค่าความเป็นศูนย์กลางโดยวัดจากค่าคั่นกลาง พบว่า ค่าความเป็นศูนย์กลางโดยวัดจากค่าคั่นกลางที่วัดได้มีค่าเป็น 0.0 ทั้งหมด เนื่องจาก ความสัมพันธ์ในเครือข่ายเป็นความสัมพันธ์ที่อธิบายถึงกรณีทดสอบกับมิวแทนต์ที่เกิดขึ้นในตัวซอร์ซโค้ด ฉะนั้น จะไม่เกิดการลากจากมิวแทนต์หนึ่งไปอีกมิวแทนต์หนึ่งผ่านจุดตัดที่เป็นกรณีทดสอบ เพราะมิวแทนต์ทั้งหมดมีทิศทางเดียวกัน คือ หัวลูกศรจะวิ่งเข้าหากรณีทดสอบ อีกนัยหนึ่ง อธิบายถึงความสัมพันธ์ที่แสดงให้เห็นว่า กรณีทดสอบที่ผ่านการทดสอบมิวเทชันครั้งที่ 1 พบว่ามีมิวแทนต์รอดชีวิตเท่าใดบ้าง และเกิดขึ้นที่ซอร์ซโค้ดใดบ้าง

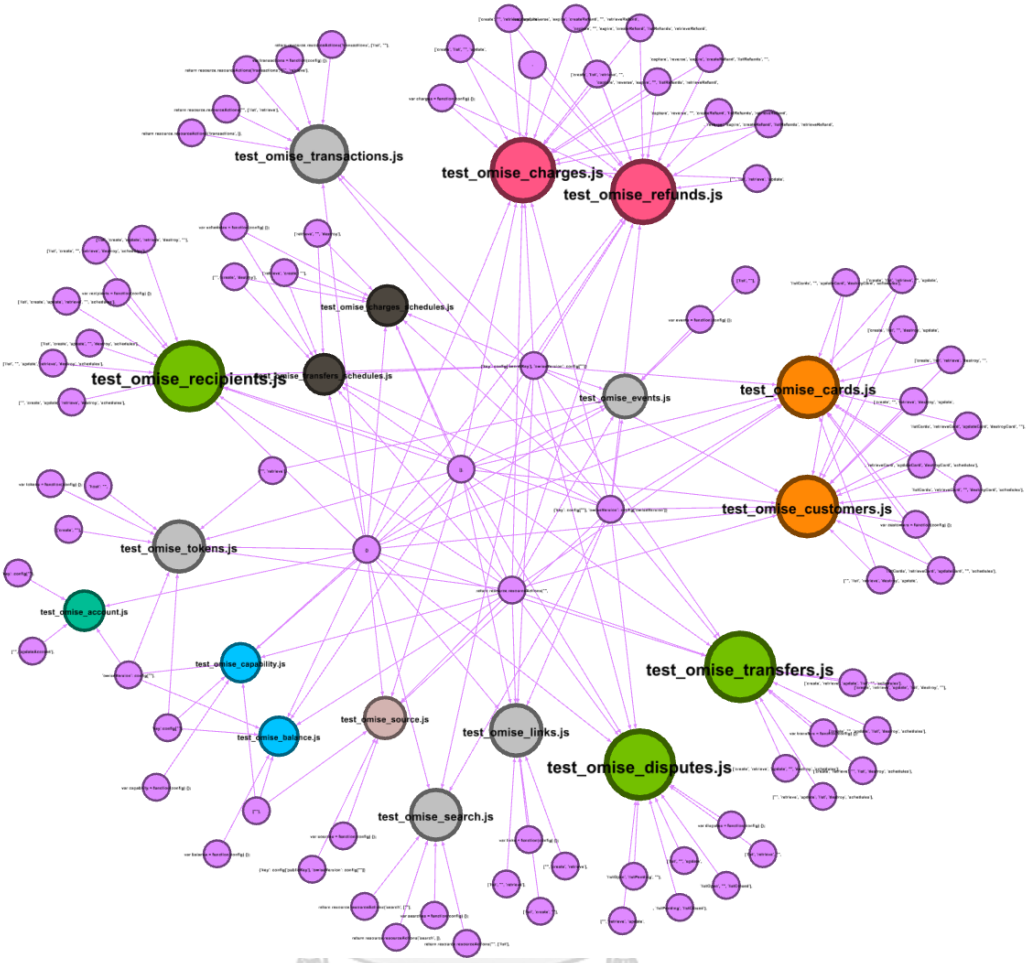
4.3.3 กราฟรูปแบบที่ 3 ค่าความเป็นศูนย์กลางโดยวัดจากความใกล้ชิด



ภาพที่ 23 แบบจำลองกราฟ SNA ค่าความเป็นศูนย์กลางโดยวัดจากความใกล้ชิด

ภาพที่ 26 แสดงผลจากการวัดค่าระดับความเป็นศูนย์กลางโดยวัดจากความใกล้ชิด พบว่าค่าความเป็นศูนย์กลางโดยวัดจากความใกล้ชิด ได้ผลลัพธ์ดังนี้ โหนดสีเขียว คือ มิวแทนต์ที่รอดชีวิตวัดได้เท่ากับ 1.0 ส่วนโหนดสีชมพู คือ กรณีทดสอบวัดได้เท่ากับ 0.0 ความสัมพันธ์ในเครือข่ายดังกล่าวเป็นความสัมพันธ์ที่ประกอบด้วย กรณีทดสอบกับมิวแทนต์ ฉะนั้น จะไม่เกิดการลากจากมิวแทนต์หนึ่งไปอีกมิวแทนต์หนึ่งผ่านจุดตัดที่เป็นกรณีทดสอบ เพราะมิวแทนต์ทั้งหมดมีทิศทางเดียวกัน คือ หัวลูกศรจะวิ่งเข้าหากรณีทดสอบ อีกนัยหนึ่ง อธิบายถึงความสัมพันธ์ที่แสดงให้เห็นว่า กรณีทดสอบที่ผ่านการทดสอบมิวเทชันครั้งที่ 1 พบว่ามีมิวแทนต์รอดชีวิตเท่าใดบ้าง และเกิดขึ้นที่ซอร์ซโค้ดใดบ้าง

4.3.4 กราฟรูปแบบที่ 4 เพจแรงก์



ภาพที่ 24 แบบจำลองกราฟ SNA ที่วัดจากค่าเพจแรงก์

ภาพที่ 31 แสดงผลจากการวัดค่าเพจแรงก์ โดยโหนดที่มีค่าเพจแรงก์ที่สูงที่สุดมี คือ test_omise_disputes.js นอกจากนี้ยังมี test_omise_recipients.jsและtest_omise_transfers.js ที่มีค่าเท่ากับ 0.04313 หมายความว่า โหนดดังกล่าวมีค่าของจุดศูนย์กลางที่ส่งผ่านด้วยจำนวนลิงก์ที่ออกจากโหนดทั้งหมดเพื่อให้โหนดเพื่อนบ้านที่เชื่อมต่อแต่ละรายได้รับเศษส่วนของจุดศูนย์กลางของโหนดต้นทาง เมื่อเปรียบเทียบกับโหนด test_omise_capability.js ที่มีค่าเพจแรงก์น้อยที่สุดเท่ากับ 0.016624 จากค่าที่วัดได้มาข้างต้นของค่าที่มากที่สุดและน้อยที่สุด เมื่อ ดำเนินการทดสอบมิวเทชันโดยนำกรณีทดสอบมาประเมินด้วยเครื่องมือ Stryker ได้ผลดังภาพที่ 32

File / Directory	Mutation score		# Killed	# Survived	# Timeout	# No coverage	# Ignored	# Runtime errors	# Compile errors	Total detected	Total undetected	Total mutants
All files	25.40	25.40	73	208	22	71	0	18	0	95	279	392
errors/api-error.js		0.00	0	0	0	2	0	0	0	0	2	2
resources	6	6.08	9	139	0	0	0	0	0	9	139	148
api.js	52.43%	52.43	44	43	10	6	0	0	0	54	49	103
apiResources.js	32.99%	32.99	20	7	12	58	0	18	0	32	65	115
logger.js		0.00	0	19	0	5	0	0	0	0	24	24

ภาพที่ 25 การประเมินกรณีทดสอบจากการวัดค่าเพจแรงก์ลำดับที่ 1

หลังจากใช้เครื่องมือ Stryker เพื่อประเมินชุดกรณีทดสอบของซอร์ซโค้ด test_omise_disputes.js ดังภาพที่ 32 หลังจาก Stryker สรุปผลคะแนนได้ร้อยละ 6.08 หมายความว่ายังมีมิวแทนต์รอดชีวิตในซอร์ซโค้ด ดังตารางที่ 8 คือ การสร้างมิวแทนต์ด้วย Stryker ในซอร์ซโค้ด resources/Dispute.js

ตารางที่ 6 การสร้างมิวแทนต์ด้วย Stryker ในซอร์ซโค้ด lib/resources/Dispute.js จากการจัดลำดับของค่าเพจแรงก์ที่มีอิทธิพลมากที่สุด

ซอร์ซโค้ด	ซอร์ซโค้ดกลายพันธุ์

{
'key': config['secretKey'],
'omiseVersion': config['omiseVersion'],
}
	{}

'key': config['secretKey'],	'key': config[""],
'omiseVersion': config['omiseVersion'],	'omiseVersion': config[""],

ผู้วิจัยทำการเพิ่มกรณีทดสอบ โดยให้ความสนใจที่มิวแทนต์ที่ถูกสร้างขึ้น ดังภาพที่ 33 โดยประกาศ “stub” ไว้ในบรรทัดที่ 92 และทำการเรียก “stub” จากซอร์ซโค้ดมาทำงานในกรณีทดสอบดังแสดงในบรรทัดที่ 90 ถึง 102

```
90      it.skip('should be able to killed mutated in ', function () {
91        testHelper.setupMock('disputes_retrieve');
92        const apiGetStub = sandbox.stub(api, 'get');
93        omise.disputes.retrieve();
94        expect(apiGetStub.withArgs(
95          {
96            key: '321',
97            omiseVersion: '2019-05-29',
98            path: '/disputes',
99            body: undefined
100         }, undefined
101       ).calledOnce).to.be.true;
102     });
```

ภาพที่ 26 กรณีทดสอบของ test_omise_disputes.js จากการวัดค่าเพจแรงก์ที่มีอิทธิพลมากที่สุด

ผลลัพธ์ที่ได้จากการจัดลำดับด้วยการวัดค่าเพจแรงก์ ดังภาพที่ 34 ภาพรวมของคะแนนในการเลือกนำซอร์ซโค้ดกลายพันธุ์ลำดับที่ 1 มาทำกรณีทดสอบ จากเดิมที่มีผลรวมของคะแนนเท่ากับ 21.66 เมื่อทำการทดสอบมิวเทชันและสามารถฆ่ามิวแทนต์ทั้งหมดได้ในกรณีทดสอบของซอร์ซโค้ด resources/Dispute.js ดังภาพที่ 35 มีผลรวมของคะแนนเท่ากับ 26.47 มีผลต่างอยู่ที่ 1.07 เท่านั้น

File / Directory	Mutation score		# Killed	# Survived	# Timeout	# No coverage	# Ignored	# Runtime errors	# Compile errors	Total detected	Total undetected	Total mutants
All files	26.47%	26.47	77	204	22	71	0	18	0	99	275	392
errors/api-error.js	0.00%	0.00	0	0	0	2	0	0	0	0	2	2
resources	8.11%	8.11	12	136	0	0	0	0	0	12	136	148
api.js	52.43%	52.43	44	43	10	6	0	0	0	54	49	103
apiResources.js	34.02%	34.02	21	6	12	58	0	18	0	33	64	115
logger.js	0.00%	0.00	0	19	0	5	0	0	0	0	24	24

ภาพที่ 27 คะแนนของลำดับที่ 1 จากการวัดค่าเพจแรงก์ หลังจากทดสอบมิวเทชัน

File / Directory	Mutation score		# Killed	# Survived	# Timeout	# No coverage	# Ignored	# Runtime errors	# Compile errors	Total detected	Total undetected	Total mutants
Dispute.js	100.00%	100.00	12	0	0	0	0	0	0	12	0	12

Killed (12)

```
1   'use strict';
2
3   var resource = require('../apiResources');
4
5   var disputes = function(config) {
6     return resource.resourceActions('disputes',
7       ['list', 'retrieve', 'update',
8         'listOpen', 'listPending', 'listClosed'],
9       {'key': config['secretKey'], 'omiseVersion': config['omiseVersion']}
10    );
11  };
12
13  module.exports = disputes;
14
```

ภาพที่ 28 คะแนนของซอร์ซโค้ด resources/Dispute.js ที่ฆ่ามิวแทนต์ได้ทั้งหมด

ในทางตรงกันข้าม นำโหนดที่มีค่าเพจแรงก์น้อยสุดที่มีชื่อว่า test_omise_capability.js นำกรณีทดสอบมาประเมินด้วยเครื่องมือ Stryker ได้ผลดังภาพที่ 36

File / Directory	Mutation score		# Killed	# Survived	# Timeout	# No coverage	# Ignored	# Runtime errors	# Compile errors	Total detected	Total undetected	Total mutants
All files	19.52	19.52	58	212	15	89	0	18	0	73	301	392
errors/api-error.js		0.00	0	0	0	2	0	0	0	0	2	2
resources		2.70	4	144	0	0	0	0	0	4	144	148
api.js	46.60%	46.60	38	43	10	12	0	0	0	48	55	103
apiResources.js	21.65	21.65	16	6	5	70	0	18	0	21	76	115
logger.js		0.00	0	19	0	5	0	0	0	0	24	24

ภาพที่ 29 การประเมินกรณีทดสอบจากการวัดค่าเพจแรงก์ลำดับที่ 19

หลังจากใช้เครื่องมือ Stryker เพื่อประเมินชุดกรณีทดสอบ test_omise_capability.js ดังภาพที่ 36 หลังจาก Stryker สรุปผลคะแนนได้ร้อยละ 2.70 หมายความว่ายังมีมิวแทนต์รอดชีวิตในซอร์ซโค้ด ดังตารางที่ 9 คือการสร้างมิวแทนต์ด้วย Stryker ในซอร์ซโค้ด resources/Capability.js

ตารางที่ 7 การสร้างมิวแทนต์ด้วย Stryker ในซอร์ซโค้ด resources/Capability.js จากการจัดลำดับของค่าเพจแรงก์ที่มีอิทธิพลน้อยที่สุด

ซอร์ซโค้ด	ซอร์ซโค้ดกลายพันธุ์

{
'key': config['secretKey'],
'omiseVersion': config['omiseVersion'],
}
	{}

{'key': config['secretKey'], 'omiseVersion': config['omiseVersion']}	{'key': config[""], 'omiseVersion': config['omiseVersion']}
{'key': config['secretKey'], 'omiseVersion': config['omiseVersion']}	{'key': config['secretKey'], 'omiseVersion': config[""]}

ผู้วิจัยทำการเพิ่มกรณีทดสอบ โดยให้ความสนใจที่มิวแทนต์ที่ถูกสร้างขึ้น ดังภาพที่ 37 โดยประกาศ “stub” ไว้ในบรรทัดที่ 34 และทำการเรียก “stub” จากซอร์ซโค้ดมาทำงานในกรณีทดสอบดังแสดงในบรรทัดที่ 37 ถึง 42

```
32    it.skip('should be able to killed mutated in capabilities', function () {
33      testHelper.setupMock('capability_retrieve');
34      const apiGetStub = sandbox.stub(api, 'get');
35      omise.capability.retrieve();
36      expect(apiGetStub.withArgs(
37        { key: '123',
38        omiseVersion: '2019-05-29',
39        path: '/capability',
40        body: undefined}, undefined
41      ).calledOnce).to.be.true;
42    });
```

ภาพที่ 30 กรณีทดสอบของซอร์ซโค้ด resources/Capability.js จากการวัดค่าเพจแรงก์ที่มีอิทธิพลน้อยที่สุด

ผลลัพธ์ที่ได้จากการจัดลำดับด้วยการวัดค่าเพจแรงก์ ดังภาพที่ 38 ภาพรวมของคะแนนในการเลือกนำซอร์ซโค้ดกลายพันธุ์ลำดับที่ 1 มาทำกรณีทดสอบ จากเดิมที่มีผลรวมของคะแนนเท่ากับ 19.52 เมื่อทำการทดสอบมิวเทชันและสามารถฆ่ามิวแทนต์ทั้งหมดได้ในกรณีทดสอบของซอร์ซโค้ด resources/Capability.js ดังภาพที่ 39 มีผลรวมของคะแนนเท่ากับ 20.59 มีผลต่างอยู่ที่ 1.07 เท่านั้น

File / Directory	Mutation score		# Killed	# Survived	# Timeout	# No coverage	# Ignored	# Runtime errors	# Compile errors	Total detected	Total undetected	Total mutants
All files	20.59	20.59	62	208	15	89	0	18	0	77	297	392
errors/api-error.js	0.00	0.00	0	0	0	2	0	0	0	0	2	2
resources	4.73	4.73	7	141	0	0	0	0	0	7	141	148
api.js	46.60%	46.60	38	43	10	12	0	0	0	48	55	103
apiResources.js	22.68	22.68	17	5	5	70	0	18	0	22	75	115
logger.js	0.00	0.00	0	19	0	5	0	0	0	0	24	24

ภาพที่ 31 ลำดับคะแนนที่น้อยที่สุดจากการวัดค่าเพจแรงก์หลังจากทดสอบมิวเทชัน

File / Directory	Mutation score		# Killed	# Survived	# Timeout	# No coverage	# Ignored	# Runtime errors	# Compile errors	Total detected	Total undetected	Total mutants
Event.js	100.00%	100.00	8	0	0	0	0	0	0	8	0	8

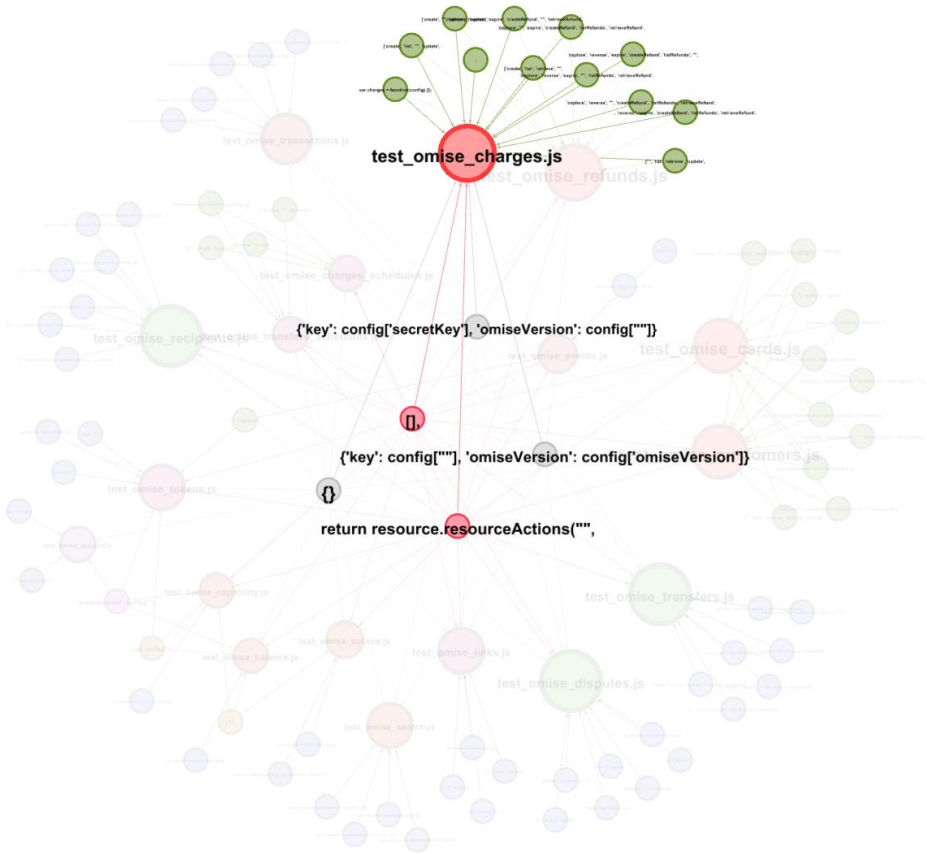
Killed (8)

```
1   'use strict';
2
3   var resource = require('../apiResources');
4
5   var events = function(config) {
6     return resource.resourceActions('events',
7       ['list', 'retrieve'],
8       {'key': config['secretKey'], 'omiseVersion': config['omiseVersion']}
9     );
10  };
11
12  module.exports = events;
13
```

ภาพที่ 32 คะแนนของซอร์ซโค้ด resources/Capability.js ที่ฆ่ามิวแทนต์ได้ทั้งหมด

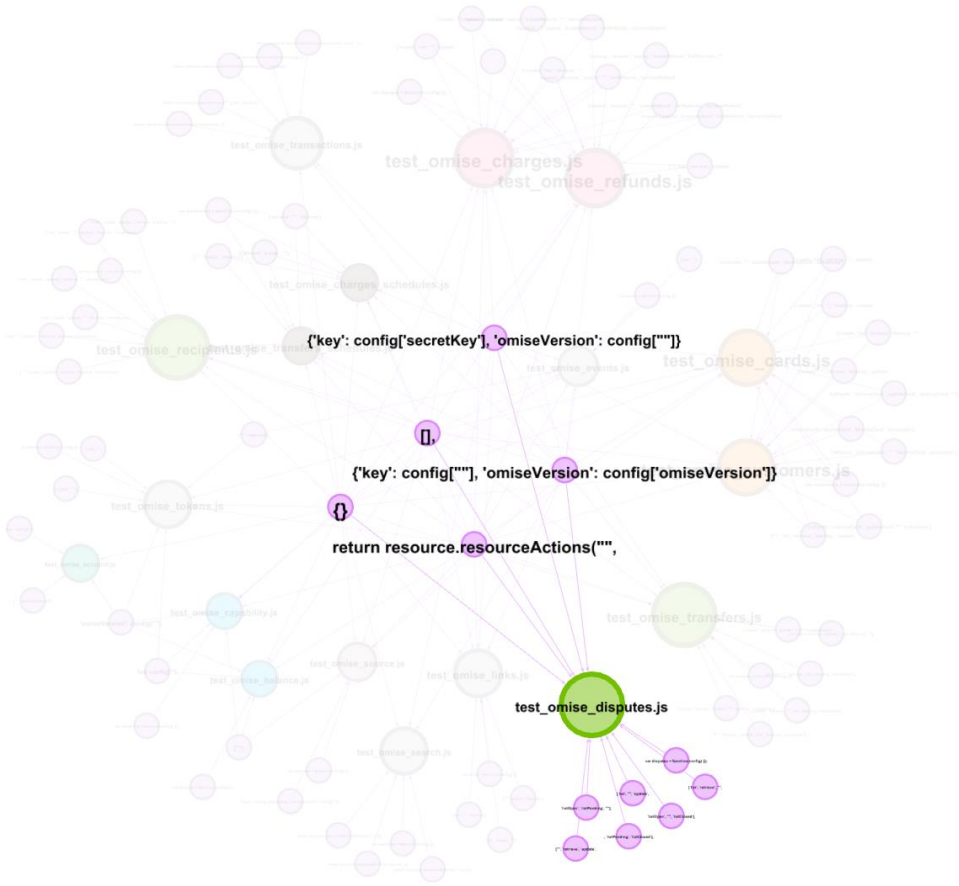
4.4 ผลการทดลอง

จากการสร้างกราฟ SNA ทั้ง 4 แบบข้างต้น ผู้วิจัยได้นำข้อมูลของโครงการ Omise มาทำการทดสอบโดยการวัดค่าความเป็นศูนย์กลางผ่านโปรแกรม Gephi เพื่อวิเคราะห์และแสดงผลทำให้ได้กราฟทั้ง 4 แบบที่ถูกสร้างขึ้นเพื่อหาความสัมพันธ์ของซอร์ซโค้ดและซอร์ซโค้ดกลายพันธุ์ ได้แก่ ค่าระดับความเป็นศูนย์กลาง ค่าความเป็นศูนย์กลางโดยวัดจากค่าคั่นกลาง ค่าความเป็นศูนย์กลางโดยวัดจากความใกล้ชิด และค่าเพจแรงก์



ภาพที่ 33 กราฟ SNA เพื่อแสดงโหนดที่สำคัญจากค่า Degree Centrality

จากภาพที่ 40 ในเครือข่ายสังคมของโครงการ Omise ซึ่งมีข้อมูลทั้งหมด 99 โหนด มีเส้นที่เชื่อมโยง หรือมิวแทนต์ทั้งสิ้น 186 เส้น โดยโหนดที่มีค่าระดับสูงที่สุด คือ resources/Charge.js มีค่าเท่ากับ ถัดมาจะเป็นโหนดที่ชื่อว่า resources/ Dispute.js มีค่าเท่ากับ 0.04313 ส่วนกราฟ SNA ที่วัดค่าเพจแรงก์นั้นมีโหนดที่สำคัญ ดังภาพที่ 43



ภาพที่ 34 กราฟ SNA เพื่อแสดงโหนดที่สำคัญจากค่า PageRank

สรุปผลการวัดค่าความเป็นศูนย์กลางทั้ง 4 แบบ ได้แก่ ค่าระดับความเป็นศูนย์กลาง ค่าความเป็นศูนย์กลางโดยวัดจากค่าคั่นกลาง ค่าความเป็นศูนย์กลางโดยวัดจากค่าความใกล้ชิด และเพจแรงก์ ดังตารางที่ 10 โดยเรียงลำดับจากซอร์ซโค้ดที่มีค่าเพจแรงก์สูงที่สุดตามลำดับ

ตารางที่ 8 รายการผลลัพธ์ของซอร์ซโค้ดที่วัดด้วยค่าความเป็นศูนย์กลางทั้ง 4 แบบ

กรณีทดสอบ	ค่าระดับ	ค่าคั่นกลาง	ค่าความใกล้ชิด	เพจแรงก์
test_omise_account.js	4	0.0	0.0	0.017637
test_omise_balance.js	7	0.0	0.0	0.016624
test_omise_capability.js	7	0.0	0.0	0.016624
test_omise_cards.js	16	0.0	0.0	0.03553
test_omise_charges_schedules.js	9	0.0	0.0	0.017797
test_omise_charges.js	17	0.0	0.0	0.038063
test_omise_customers.js	16	0.0	0.0	0.03553
test_omise_disputes.js	12	0.0	0.0	0.04313
test_omise_events.js	8	0.0	0.0	0.020331
test_omise_links.js	9	0.0	0.0	0.02793
test_omise_recipients.js	12	0.0	0.0	0.04313
test_omise_refunds.js	17	0.0	0.0	0.038063
test_omise_search.js	7	0.0	0.0	0.027255
test_omise_source.js	7	0.0	0.0	0.019096
test_omise_tokens.js	9	0.0	0.0	0.027621
test_omise_transactions.js	8	0.0	0.0	0.032321
test_omise_transfers_schedules.js	9	0.0	0.0	0.017797
test_omise_transfers.js	12	0.0	0.0	0.04313

จากตารางสรุปผลของค่าความเป็นศูนย์กลางเมื่อเปรียบเทียบการดำเนินการทดสอบมิวเทชันเฉพาะค่าความเป็นศูนย์กลางที่สูงที่สุดทั้ง 4 แบบพบว่า คะแนนที่ได้จากการดำเนินการทดสอบมิวเทชันสรุปออกมาเป็นดังตารางที่ 11

ตารางที่ 9 รายการผลลัพธ์ของซอร์ซโค้ดที่ดำเนินการทดสอบมิวเทชันจากการจัดลำดับค่าความเป็นศูนย์กลางที่มีอิทธิพลมากที่สุดทั้ง 4 แบบ

กรณีทดสอบ	ค่าความเป็นศูนย์กลาง	คะแนนของการทดสอบมิวเทชัน
test_omise_charges.js	ค่าระดับ	24.33
test_omise_disputes.js	**เพจแรงก์**	**26.47**

จากผลการจัดลำดับค่าความเป็นศูนย์กลางทั้ง 4 แบบนำกรณีทดสอบที่มีอิทธิพลมากที่สุดมาดำเนินการทดสอบมิวเทชันและเปรียบเทียบผลการดำเนินงาน พบว่าค่าของ “เพจแรงก์” มีคะแนนมิวเทชันที่สูงที่สุด เมื่อเปรียบเทียบกับอับดับที่ 1 ที่ได้จากค่าระดับ นอกจากนั้น ค่าระดับยังคงมีมิวแทนต์ที่รอดชีวิตจากการทดสอบมิวเทชัน ไม่เหมือนค่าลำดับที่ 1 ของเพจแรงก์ที่สามารถฆ่ามิวแทนต์ได้ร้อยละ 100.00

สรุปได้ว่า เครื่องมือ Stryker ช่วยประหยัดเวลาในการศึกษาตัวดำเนินการมิวเทชันกับซอร์ซโค้ดในโครงการ เพื่อสร้างมิวแทนต์ในการดำเนินการทดสอบมิวเทชัน จากลักษณะของมิวแทนต์ที่สร้างขึ้นในโครงการ ผู้วิจัยได้วิเคราะห์เพื่อกำหนดโหนดและเส้นเชื่อมจากความสัมพันธ์ของซอร์ซโค้ดกับมิวแทนต์ จากการเขียนโปรแกรมปัจจุบันมีลักษณะการเขียนโปรแกรมเชิงอ็อบเจกต์ (object-oriented programming) ที่มีการเรียกใช้ส่วนโปรแกรม (component) ต่าง ๆ ภายในโครงการ มาสร้างเป็นกราฟและวัดค่าความเป็นศูนย์กลางจากบริบทของตัววัดสามารถบ่งชี้ได้จากกราฟเครือข่าย เมื่อต้องการจัดลำดับซอร์ซโค้ดที่ทรงอิทธิพลที่สุด ที่มีการกลายพันธุ์ลักษณะแบบเดียวกัน ต้องค้นหาด้วยวิธีการเพจแรงก์ ซึ่งแสดงได้จากกราฟในภาพที่ 43 ที่เห็นได้ชัดว่าโหนดที่มีชื่อว่า test_omise_disputes.js เป็นกรณีทดสอบที่ทรงอิทธิพลมากที่สุด มีการเชื่อมโยงกับโหนดอื่น ๆ และแบ่งเศษส่วนกับโหนดอื่น ๆ มากที่สุด เป็นการวัดอิทธิพลหรือความสำคัญของโหนด โดยโหนดที่มีคอนเนคชันมากกว่ามีลักษะการถ่ายทอดซอร์ซโค้ดหรือนำเข้าซอร์ซโค้ดนั้น ๆ และมองเห็นได้ด้วยตาเปล่าจากสภาพแวดล้อมของโครงการนั้น เมื่อได้ผลลัพธ์ของการวัดค่าความเป็นศูนย์กลางจากค่าทั้ง 4 ดำเนินการทดสอบมิวเทชันเพื่อลดช่องโหว่หรือความผิดพลาดที่กรณีทดสอบเดิมไม่สามารถครอบคลุมทั้งหมด

บทที่ 5

สรุปผลการวิจัย

วิทยานิพนธ์นี้ได้นำเสนอวิธีการที่เห็นได้ชัดว่าการทดสอบเป็นหนึ่งในกิจกรรมการใช้ทรัพยากรในโครงการซอฟต์แวร์ และเหมาะอย่างยิ่งเพื่อให้ครอบคลุมเมื่อทำการทดสอบ ในวรรณกรรม การจัดลำดับความสำคัญของกรณีทดสอบเป็นหนึ่งในวิธีแก้ปัญหาที่รู้จักกันดีในการช่วยทำให้การทดสอบรวดเร็วขึ้น

การทดสอบการกลายพันธุ์เป็นเทคนิคที่ใช้สำหรับการประเมินคุณภาพของชุดทดสอบ ช่วยระบุช่องโหว่ของกรณีทดสอบด้วยการแทรกการกลายพันธุ์ลงในซอร์ซโค้ดดั้งเดิม กรณีทดสอบคุณภาพสูงคาดว่าจะสามารถฆ่าการกลายพันธุ์ได้ในระดับสูง เอกสารนี้เสนอให้ใช้การวัดศูนย์กลางของเครือข่ายที่เรียกว่า PageRank สำหรับการจัดลำดับความสำคัญของกรณีทดสอบระหว่างการทดสอบการกลายพันธุ์ที่ดำเนินการโดยเครื่องมือ Stryker ซอร์ซโค้ดที่เรียกใช้ฟังก์ชันจำนวนมากจะได้รับคะแนน PageRank สูง เนื่องจากมีการถ่ายทอดซอร์ซโค้ดออกจำนวนมากจากโหนดอื่น ขั้นตอนการทดสอบซอฟต์แวร์ที่เริ่มต้นจากซอร์ซโค้ดเหล่านี้ที่มีคะแนน PageRank สูงจะทำให้การทดสอบครอบคลุมมากขึ้น เนื่องจากผู้ทดสอบสามารถข้ามการเขียนกรณีทดสอบและละเว้นการทดสอบซอร์ซโค้ด PageRank ที่ต่ำกว่าซึ่งมีการกลายพันธุ์ประเภทเดียวกันที่ฆ่าได้สำเร็จเมื่อทำการทดสอบซอร์ซโค้ดที่มีค่า PageRank สูงเดิม

บรรณานุกรม

1. Zafarani R, Abbasi MA, Liu H. Social media mining: an introduction: Cambridge University Press; 2014.

2. Brin S, Page L. The Anatomy of a Large-Scale Hypertextual Web Search Engine. 7th International World-Wide Web Conference; Brisbane, Australia: Elsevier Science Publishers B. V.; 1998. p. 107-17.

3. Jorgensen PC. Software Testing: A Craftsman's Approach, Fourth Edition: Hoboken : CRC Press; 2013.

4. Ammann P, Offutt J. Introduction to Software Testing: Cambridge: Cambridge University Press; 2016.

5. Du Y, Pan Y, Ao H, Alexander NO, Fan Y. Automatic Test Case Generation and Optimization Based on Mutation Testing. IEEE 19th International Conference on Software Quality, Reliability and Security Companion2019. p. 522-3.

6. Maitrikul C, Limpiyakorn Y. GUI Test Case Prioritization using Social Network Analysis. 13th International Conference on Computer and Electrical Engineering; Beijing, China2020.

7. Koochakzadeh N, Alhajj R. Social Network Analysis in Software Testing to Categorize Unit Test Cases Based on Coverage Information. 011 IEEE International Conference on High Performance Computing and Communications; Banff, AB, Canada: IEEE; 2011. p. 412-6.

8. Omise [Available from: https://github.com/omise/omise-node.

9. StrykerJS [Available from: https://stryker-mutator.io/docs/stryker-js/introduction/.

10. Supported mutators [Available from: https://stryker-mutator.io/docs/mutation-testing-elements/supported-mutators.

11. Mutant states and metrics [Available from: https://stryker-mutator.io/docs/mutation-testing-elements/mutant-states-and-metrics/.

12. Gephi [Available from: https://gephi.org/.

13. Mocha [Available from: https://mochajs.org/.

ประวัติผู้เขียน

ชื่อ-สกุล	ศุภชัย ทรัพย์มาก
วัน เดือน ปี เกิด	28 กุมภาพันธ์ 2540
สถานที่เกิด	บุรีรัมย์
วุฒิการศึกษา	วิทยาศาสตรบัณฑิต
ที่อยู่ปัจจุบัน	126/47 ซ.สุขาภิบาล 5ซ.32แยก10 แขวงออเงิน เขตสายไหม กรุงเทพมหานคร
ผลงานตีพิมพ์	S Supmak, Y Limpiyakorn. (2022). Prioritization of Mutation Test Generation with Centrality Measure. ICSTS 2022: XVI. International Conference on Software Testing Strategies.