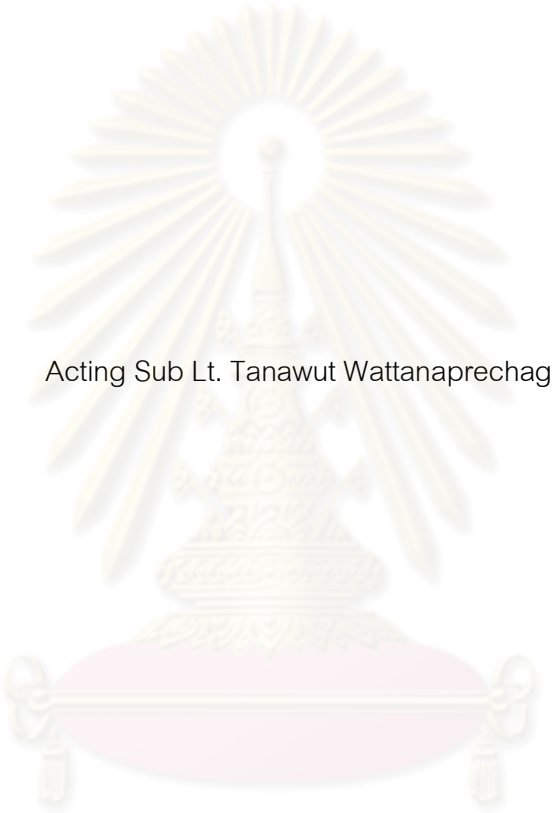
การประยุกต์กลวิธีการกลายรหัสบนโครงสร้างต้นไวยากรณ์เชิงนามธรรมสำหรับระบบฝึกฝนการแก้ไขข้อบกพร่อง

ว่าที่ร้อยตรี ธนาวุฒิ วัฒนปรีชากิจ

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2552

APPLYING CODE MUTATION ON ABSTRACT SYNTAX TREE FOR DEBUGGING TRAINING SYSTEM

Acting Sub Lt. Tanawut Wattanaprechagit



A Thesis Submitted in Partial Fulfillment of the Requirements

for the Degree of Master of Science Program in Computer Science

Department of Computer Engineering

Faculty of Engineering

Chulalongkorn University

Academic Year 2009

หัวข้อวิทยานิพนธ์	การประยุกต์กลวิธีการกลายรหัสบนโครงสร้างต้นไวยากรณ์เชิงนามธรรมสำหรับระบบฝึกฝนการแก้ไขข้อบกพร่อง
โดย	ว่าที่ร้อยตรี ธนาวุฒิ วัฒนปรีชากิจ
สาขาวิชา	วิทยาศาสตร์คอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	รองศาสตราจารย์ ดร. สมชาย ประสิทธิ์จูตระกูล

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

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

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(รองศาสตราจารย์ ดร. ธาราทิพย์ สุวรรณศาสตร์)

..... อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(รองศาสตราจารย์ ดร.สมชาย ประสิทธิ์จูตระกูล)

.....กรรมการภายนอกมหาวิทยาลัย
(อาจารย์ ดร. เนื่องวงศ์ ทวยเจริญ)

ธนาวุฒิ วัฒนปรีชากิจ : การประยุกต์กลวิธีการกลายรหัสบนโครงสร้างต้นไม้ไวยากรณ์เชิงนามธรรมสำหรับระบบฝึกฝนการแก้ไขข้อบกพร่อง (APPLYING CODE MUTATION TECHNIQUES ON ABSTRACT SYNTAX TREE FOR DEBUGGING TRAINING SYSTEM) อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก : รศ.ดร.สมชาย ประสิทธิ์จูตระกูล, 118 หน้า.

ความขาดแคลนระบบในการฝึกฝนการแก้ไขข้อบกพร่องเป็นแรงบีบคั้นให้ผู้ที่เรียนการเขียนโปรแกรมต้องขวนขวายเพื่อให้ได้มาซึ่งทักษะในการแก้ไขข้อบกพร่อง จึงมีความจำเป็นและเป็นประโยชน์อย่างมากสำหรับการคิดค้นระบบการฝึกฝนที่มีประสิทธิภาพสำหรับนักเรียน ในวิทยานิพนธ์นี้ได้นำเสนอการประยุกต์กลวิธีการกลายรหัสบนโครงสร้างต้นไม้ไวยากรณ์เชิงนามธรรม สำหรับใส่รหัสที่มีความผิดพลาดลงในโปรแกรมเพื่อนำมาสร้างเป็นระบบฝึกฝนการแก้ไขข้อบกพร่องสำหรับนักเรียน โดยผลที่ได้พบว่าระบบสามารถใส่ความผิดพลาดเชิงความหมายต่าง ๆ ลงในโปรแกรมได้ตามที่ต้องการ

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต.....
สาขาวิชา วิทยาศาสตร์คอมพิวเตอร์ ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์.....
ปีการศึกษา 2552

4970346621 : MAJOR COMPUTER SCIENCE

KEYWORDS : ABSTRACT SYNTAX TREE/ DEBUGGING / ENBUGGING

TANAWUT WATTANAPRECAHGIT: APPLYING CODE MUTAION TECHNIQUES ON ABSTRACT SYNTAX TREE FOR DEBUGGING TRAINGING SYSTEM: ASSOC. PROF. SOMCHAI PRASITJUTRAKUL, Ph.D., 118 pp.

The lack of debugging training forces programming students to acquire the skills by themselves. Therefore, it is necessary and beneficial to invent an effective training system for them. In this thesis, we present an application of the abstract syntax tree for adding defective codes into existing programs in order to create debugging training exercises for students. As a result our system can insert many semantic errors in to the programs

Department : Computer Engineering Student's Signature ธนวุฒิ วัฒนเปรชากิจ

Field of Study : Computer Science Advisor's Signature

Academic Year : 2009

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยความอนุเคราะห์อย่างยิ่งของรองศาสตราจารย์ ดร. สมชาย ประสิทธิ์จูตระกูล อาจารย์ที่ปรึกษา ซึ่งท่านได้ให้ความรู้ แนะนำแนวทางการวิจัย ตรวจสอบให้คำแนะนำ และสนับสนุนเป็นอย่างดี จนทำให้การวิจัยในครั้งนี้สำเร็จออกมาด้วยดี

ขอขอบพระคุณ รองศาสตราจารย์ ดร. ธาราทิพย์ สุวรรณศาสตร์ และอาจารย์ ดร.เนื่องวงศ์ ทวยเจริญ กรรมการสอบวิทยานิพนธ์ ที่กรุณาเสียสละเวลา ให้คำแนะนำ ตรวจสอบ และแก้ไขวิทยานิพนธ์ฉบับนี้

ท้ายที่สุด ผู้เสนอวิทยานิพนธ์ขอขอบคุณเพื่อน ๆ ทุก ๆ คน รวมทั้งครอบครัว เพื่อนร่วมงาน และผู้บังคับบัญชาในสายงาน ที่คอยติดตาม ให้กำลังใจและสนับสนุน รวมถึงท่านอื่น ๆ ที่มิได้กล่าวชื่อไว้ ณ ที่นี้ที่มีส่วนช่วยให้วิทยานิพนธ์สำเร็จได้ด้วยดี

สารบัญ

หน้า

สารบัญตาราง

หน้า

สารบัญรูป

หน้า

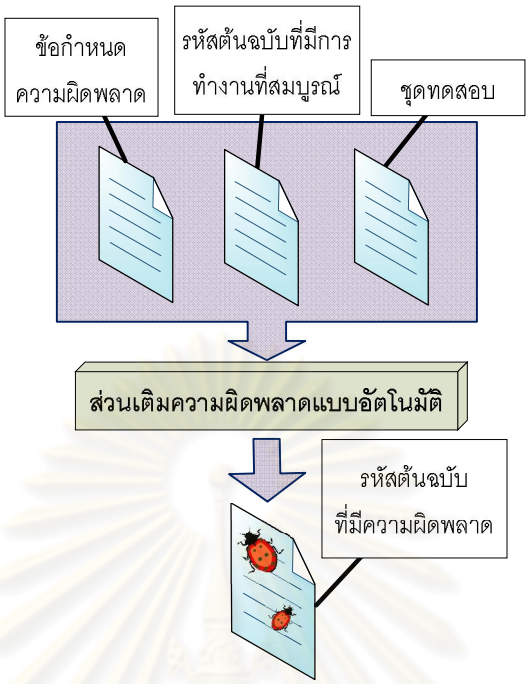
บทที่ 1

บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

การเรียนการสอนวิชาการเขียนโปรแกรมคอมพิวเตอร์ระดับพื้นฐานเป็นจุดเริ่มต้นของการพัฒนาความรู้ ความเข้าใจ และทักษะในการเขียนโปรแกรม นักเรียนจะต้องเรียนรู้ภาษาใหม่ที่ไม่คุ้นเคยพร้อมแนวคิดใหม่เพื่อสื่อสารและออกคำสั่งให้เครื่องคอมพิวเตอร์สามารถทำงานได้ตามที่ต้องการ จึงเป็นที่ทราบกันดีว่านักเรียนที่เริ่มต้นเขียนโปรแกรมคอมพิวเตอร์นั้นมักเกิดความสับสนกับโครงสร้าง และไวยากรณ์ของภาษา เช่น คำสำคัญ ตัวดัดแปร ตัวดำเนินการ ตัวแปร และค่าของตัวแปรต่าง ๆ หรือ หลักการทำงานของโปรแกรม เช่น โครงสร้างการวนซ้ำ นิพจน์เงื่อนไข หรือการประเมินค่าทางตรรกะ ความสับสนเหล่านี้นำมาซึ่งความผิดพลาดในการทำงานของโปรแกรม และเมื่อเกิดข้อบกพร่องขึ้นภายในโปรแกรม นักเรียนจะต้องคอยรับมือกับข้อบกพร่องที่เกิดขึ้นเหล่านี้ทั้งที่ยังไม่มีความชำนาญและความเข้าใจที่เพียงพอ จากการสังเกตผู้เชี่ยวชาญในการเขียนโปรแกรมนั้นจะมีความเชี่ยวชาญในการค้นหาและแก้ไขข้อบกพร่องต่างๆที่เกิดขึ้นในโปรแกรมได้อย่างรวดเร็วและถูกต้อง เนื่องจากผ่านประสบการณ์ในการค้นหาและแก้ไขข้อบกพร่องมาแล้วเป็นจำนวนมาก ดังนั้นจึงน่าจะสามารถช่วยฝึกฝนนักเรียนในการค้นหาและแก้ไขข้อบกพร่องต่างๆที่เกิดขึ้นในโปรแกรมได้ด้วยแบบฝึกหัดการแก้ไขข้อบกพร่องในรหัสต้นฉบับของโปรแกรม ซึ่งมีข้อดีคือจะช่วยฝึกฝนทักษะในการแก้ข้อบกพร่องที่มักเกิดขึ้นอย่างบ่อยครั้งที่อาจเกิดซ้ำ ๆ ในรูปแบบเดิม และจะเป็นการช่วยส่งเสริมนักเรียนให้สามารถลดความผิดพลาดในการเขียนโปรแกรมลงได้ นอกจากนั้นยังสามารถช่วยให้เกิดความเข้าใจถึงหลักการทำงานของภาษาคอมพิวเตอร์ขณะฝึกฝนการแก้ไขข้อบกพร่องต่างๆได้อีกด้วย

จากที่กล่าวมาในขั้นต้นจึงทำให้เกิดแนวคิดในการพัฒนาระบบฝึกฝนการแก้ไขข้อบกพร่องในรหัสต้นฉบับของโปรแกรมขึ้น โดยอาศัยส่วนเติมข้อบกพร่องแบบอัตโนมัติที่มุ่งเน้นในส่วนของความผิดพลาดเชิงความหมายเพื่อช่วยในการสร้างแบบฝึกหัดสำหรับฝึกฝนการแก้ไขข้อบกพร่องในรหัสต้นฉบับของโปรแกรมคอมพิวเตอร์จากการเติมข้อบกพร่องลงในรหัสต้นฉบับอย่างมีเป้าหมาย ดังรูปที่ 1.1



รูปที่ 1.1 กรอบแนวความคิดในการสร้างส่วนเติมความผิดพลาดแบบอัตโนมัติ

จากรูปที่ 1.1 ในการเติมข้อบกพร่องลงในรหัสต้นฉบับนั้นส่วนเติมข้อบกพร่องแบบอัตโนมัตินี้จะทำการรับข้อมูลของลักษณะข้อบกพร่องจากข้อกำหนดความผิดพลาดที่ต้องการเข้ามาแล้วดำเนินการค้นหารูปแบบโครงสร้างการทำงานภายในรหัสต้นฉบับที่สามารถเติมข้อบกพร่องตามความผิดพลาดที่ต้องการ แล้วทำการแก้ไข เพิ่มเติมหรือเปลี่ยนแปลงในลักษณะต่างๆ เพื่อทำให้เกิดข้อบกพร่องเชิงความหมายขึ้นในรหัสต้นฉบับอย่างสอดคล้องกับจุดประสงค์ของการเรียนการสอน หลังจากนั้นจะทำการยืนยันความผิดพลาดที่เกิดขึ้นในขั้นตอนการเติมข้อบกพร่องด้วยชุดโปรแกรมทดสอบการทำงาน เพื่อตรวจสอบว่าได้เกิดความผิดพลาดขึ้นจริง เมื่อได้รหัสต้นฉบับที่มีความผิดพลาดแล้วหลังจากนั้นก็จะสามารถนำไปใช้เป็นแบบทดสอบ หรือ แบบฝึกหัดได้ต่อไป

1.2 วัตถุประสงค์ของการวิจัย

พัฒนาส่วนเติมความผิดพลาดเชิงความหมายแบบอัตโนมัติโดยอาศัยกลวิธีการกลายรหัสให้กับระบบฝึกฝนการแก้จุดบกพร่อง เพื่อใช้เป็นแบบฝึกหัดที่จะช่วยฝึกฝนทักษะการแก้จุดบกพร่องของโปรแกรม

1.3 ขอบเขตของการวิจัย

1) ระบบฝึกฝนการแก้ข้อบกพร่องนี้จะกระทำกับรหัสต้นฉบับของโปรแกรมภาษาจาวา

2) ระบบจะสามารถเติมได้ทั้งความผิดพลาดทางไวยากรณ์และความผิดพลาดทางความหมาย

3) ระบบจะเป็นการประยุกต์ใช้ตัวดำเนินการการกลายรหัสที่มีอยู่และออกแบบตัวดำเนินการการกลายรหัสระดับโครงสร้าง เพื่อเติมข้อบกพร่องในรหัสต้นฉบับให้ระบบฝึกฝนแก้ไขข้อบกพร่อง

4) ชุดตัวอย่างรหัสต้นฉบับของโปรแกรมและชุดทดสอบที่จะใช้ในการทดสอบการทำงานจะได้มาจากชุดปฏิบัติการวิชา 2110101

5) การเติมความผิดพลาดจะดำเนินการตามข้อกำหนดความผิดพลาดเพื่อระบุถึงลักษณะหรือตำแหน่งที่จะทำการเติมซึ่งข้อกำหนดความผิดพลาด รหัสต้นฉบับ และชุดทดสอบจะถูกกำหนดจากผู้ออกโจทย์

6) การเติมความผิดพลาดให้กับโปรแกรมจะครอบคลุมความผิดพลาดที่เกิดขึ้นเป็นประจำในชุดปฏิบัติการต่างๆในวิชา 2110101 ซึ่งจะประกอบด้วยหัวข้อต่างๆ คือ ประเภทข้อมูล นิพจน์ทางคณิตศาสตร์ การเลือกทำ วงวน อาร์เรย์ เมท็อด คลาส และอ็อบเจกต์

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1) สามารถช่วยส่งเสริมการเรียนการสอนวิชาการเขียนโปรแกรมภาษาจาวาระดับพื้นฐาน

2) ช่วยฝึกฝนนักเรียนให้เกิดความชำนาญในการเขียนโปรแกรม

3) ส่งเสริมความเข้าใจในความผิดพลาดที่เกิดขึ้น เพื่อลดปัญหาการเขียนโปรแกรมที่มีความผิดพลาด

4) ช่วยฝึกฝนทักษะการค้นหา การตรวจสอบ และ การแก้ไขความผิดพลาดที่เกิดขึ้นในโปรแกรม

1.5 วิธีดำเนินการวิจัย

1) ศึกษาความผิดพลาดที่เกิดขึ้นบ่อยในวิชาการเขียนโปรแกรมภาษาจาวาระดับพื้นฐาน

2) จัดแบ่งกลุ่มของความผิดพลาดที่เกิดขึ้นบ่อยในการเรียนการเขียนโปรแกรมระดับพื้นฐาน

3) ออกแบบวิธีการเติมความผิดพลาดที่ครอบคลุมและเหมาะสมกับระดับความผิดพลาดที่สนใจ

4) พัฒนาระบบเติมความผิดพลาดเชิงความหมายจากการรับคำสั่งแนะนำ

5) ทดสอบการทำงานของส่วนเติมข้อบกพร่องแบบอัตโนมัติ

6) สรุปผลงานวิจัย และจัดทำรูปเล่มวิทยานิพนธ์

1.6 ผลงานที่เกี่ยวข้องกับงานวิจัย

งานวิจัยนี้ได้รับการคัดเลือกตีพิมพ์ในบทความวิชาการระดับประเทศ 1 บทความ(แสดงในภาคผนวก ก.) คือ

1.6.1 บทความวิชาการเรื่อง “Application of AST for Enbugging on Debugging Training System” ซึ่งได้รับการคัดเลือกเพื่อนำเสนอและตีพิมพ์ในงานประชุมวิชาการ “The 6th International Joint Conference on Computer Science and Software Engineering” ระหว่างวันที่ 13-15พฤษภาคม 2552 ณ จังหวัดภูเก็ต ประเทศไทย

บทที่ 2

เอกสารและงานวิจัยที่เกี่ยวข้อง

2.1 แนวคิดและทฤษฎี

2.1.1 การเรียนการเขียนโปรแกรมคอมพิวเตอร์ระดับพื้นฐาน

วิชาการเขียนโปรแกรมคอมพิวเตอร์ระดับพื้นฐาน เป็นรายวิชาการเรียนการสอนที่เกี่ยวกับพื้นฐานการเขียนโปรแกรมคอมพิวเตอร์ การทำงานของโปรแกรมคอมพิวเตอร์ การออกแบบและการพัฒนาโปรแกรมคอมพิวเตอร์ ภาษาคอมพิวเตอร์ต่างๆ การโปรแกรมภาษาเชิงวัตถุ โครงสร้างการทำงานของภาษาคอมพิวเตอร์ ไวยากรณ์ของภาษาคอมพิวเตอร์ คำสำคัญ ชนิดของข้อมูลประเภทต่างๆของภาษาคอมพิวเตอร์เช่น จำนวนเต็ม จำนวนจริง ตัวอักษร ชุดตัวอักษร อาร์เรย์ และข้อมูลประเภทวัตถุ การประมวลผลข้อมูล นิพจน์ต่างๆในการเขียนโปรแกรม ตัวดำเนินการ โครงสร้างควบคุมต่างๆเช่น โครงสร้างวงวน โครงสร้างเงื่อนไขและการตัดสินใจ หน่วยฟังก์ชันและการคืนค่า การติดต่ออุปกรณ์รับเข้า/ส่งออกข้อมูลพื้นฐาน การบันทึกข้อมูลเก็บลงหน่วยความจำถาวร กลวิธีและระเบียบในการเขียนโปรแกรมเพื่อประยุกต์ใช้ในการแก้ปัญหา การตรวจและการแก้ไขข้อบกพร่องของโปรแกรมโดยนักเรียนจะได้เรียนรู้การเขียนโปรแกรมและฝึกฝนการหาและแก้ไขข้อบกพร่องกับโปรแกรมที่มีขนาดไม่ใหญ่มากนัก (ประมาณ 50-300 บรรทัด) เพื่อให้มีความสามารถในการแก้ไขปัญหาต่างๆโดยประยุกต์ใช้การเขียนโปรแกรมเพื่อแก้ปัญหานั้นๆได้

2.1.2 ภาษาจาวา [1, 2]

จาวา เป็นภาษาคอมพิวเตอร์เชิงวัตถุที่ได้รับความนิยมเป็นอย่างมากซึ่งถูกพัฒนาขึ้นโดยบริษัท Sun Microsystems ในช่วงกลางปี ค.ศ. 1990 เป็นภาษาที่ถูกพัฒนาขึ้นเพื่อใช้ในสภาพแวดล้อมแบบกระจายบนระบบอินเตอร์เน็ตด้วยคุณสมบัติการทำงานที่ไม่ขึ้นอยู่กับแบบหรือชนิดของระบบคอมพิวเตอร์จากการอาศัยการทำงานของ JVM (Java Virtual Machine) จาวายังได้รวมเอาคุณสมบัติที่ดีของภาษาเชิงวัตถุอย่างภาษา Smalltalk และมีหลักโครงสร้างของภาษาที่คล้ายกับภาษา C++ แต่จะเรียบง่ายกว่าและมีโครงสร้างเป็นภาษาเชิงวัตถุมากกว่า จาวานั้นไม่เพียงสามารถที่จะใช้ในการสร้างโปรแกรมประยุกต์ทั่วไปที่ทำงานได้ทั้งบนคอมพิวเตอร์เครื่องเดียว

หรือในเครือข่ายของระบบแบบกระจาย แต่ยังสามารถที่จะใช้ในการเขียนและสร้างเป็นส่วนประกอบที่ทำงานร่วมกับโปรแกรมหรือภาษาอื่น ๆ ได้เช่น applet และด้วยการพัฒนาที่มีเป้าหมายที่แน่นอนทำให้ภาษาจาวากลายเป็นภาษาที่มีเอกลักษณ์ที่โดดเด่นหลายประการได้แก่

1) จาวาเป็นภาษาเชิงวัตถุอย่างแท้จริงทำให้สามารถใช้แนวคิดและกลยุทธของภาษาเชิงวัตถุได้อย่างสมบูรณ์

2) จาวาเป็นภาษาคอมพิวเตอร์ที่ถูกสร้างขึ้นเพื่อให้สามารถใช้งานได้บนระบบเครือข่ายและสามารถทำงานได้โดยไม่ขึ้นกับระบบปฏิบัติการโดยอาศัยการทำงานของ JVM ซึ่งจะทำหน้าที่แปลคำสั่งจาก java bytecode เป็นชุดคำสั่งที่ทำงานจริงๆบนฮาร์ดแวร์ของเครื่อง ดังนั้นไม่ว่าเครื่องคอมพิวเตอร์เครื่องนั้นจะเป็นแบบใด อยู่ในระบบปฏิบัติการใดหรือมีคำสั่งในการทำงานที่แตกต่างกันออกไปอย่างไรก็จะยังสามารถเข้าใจและทำงานตามโปรแกรมได้

3) จาวาเป็นภาษาที่ถูกพัฒนาขึ้นเพื่อให้สามารถนำไปใช้งานได้อย่างปลอดภัย จากการตัดส่วนของความสามารถในการอ้างถึงตำแหน่งของหน่วยความจำได้โดยตรงเพื่อป้องกันการทำงานที่จะเป็นอันตรายต่อระบบ และได้มีการจัดการกับส่วนการทำงานที่ทำให้เกิดความไม่ปลอดภัยกับส่วนอื่น ๆ ทำให้เชื่อมั่นได้ถึงความปลอดภัย

4) จาวาเป็นภาษาที่มีโครงสร้างของภาษาที่เป็นระเบียบ ไม่ซับซ้อน ชัดเจน ทำให้เรียนรู้ได้ง่าย

ด้วยเหตุนี้ทำให้มีการใช้งานจาวากันอย่างกว้างขวาง ทั้งในอุตสาหกรรมต่างๆ ในการเรียนการสอน ในเว็บเพจ นอกจากนั้นจาวายังมีอิสระต่อลำดับของการนิยาม โครงสร้างของเมท็อดจะถูกนิยามไว้ต่อจากการนิยามชื่อของเมท็อดเสมอ ทำให้รหัสต้นฉบับของภาษาจาวามีโครงสร้างที่มีระเบียบ

2.1.3 **ความผิดพลาดที่เกิดขึ้นได้ในโปรแกรม**

A. Ko และ B. Myers [3] ได้กล่าวถึงความผิดพลาดที่เกิดขึ้นในโปรแกรมคอมพิวเตอร์ว่าเป็นส่วนของรหัสต้นฉบับที่มีการทำงานไม่เป็นไปตามจุดประสงค์ หรือทำงานไม่ครอบคลุมตามความต้องการของผู้เขียนโปรแกรมคอมพิวเตอร์อาจเกิดขึ้นด้วยเหตุผลต่างๆกัน เป็นปัญหาที่ต้องได้รับการแก้ไข ความผิดพลาดเหล่านี้อาจมีที่มาจากข้อบกพร่องต่างๆที่ประกอบกันเป็นความผิดพลาดที่ปรากฏขึ้น โดยทั่วไปแล้วความผิดพลาดในการเขียนโปรแกรมสามารถแบ่งออกเป็นสองกลุ่มได้แก่

2.1.3.1 ความผิดพลาดทางไวยากรณ์ (syntax error) โดยมักจะเกิดขึ้นจากความไม่แม่นยำในโครงสร้างและไวยากรณ์ของภาษา ดังรูปตัวอย่างที่ 2.1 เช่น

```
Class  Abc {
…
}
```

รูปที่ 2.1 ก)

```
class  Abc{
…
}
```

รูปที่ 2.1 ข)

รูปที่ 2.1 ก) เป็นภาษาจาวาที่มีการใช้คำสำคัญผิดพลาด

ข) เป็นภาษาจาวาที่มีการใช้คำสำคัญได้อย่างถูกต้อง

จากรหัสตัวอย่างในรูปที่ 2.1 เป็นตัวอย่างที่แสดงให้เห็นถึงความผิดพลาดทางไวยากรณ์ เพราะคำสำคัญ “class” จะต้องเป็นตัวอักษรตัวเล็ก ความผิดพลาดทางไวยากรณ์นี้จะถูกตรวจสอบในขั้นตอนการแปลโปรแกรม ซึ่งจะต้องได้รับการแก้ไขก่อน โปรแกรมจึงจะสามารถทำงานได้

2.1.3.2 ความผิดพลาดทางความหมาย (semantic error) ซึ่งจะแบ่งแยกได้อีกเป็นความผิดพลาดในขณะทำงาน (runtime error) และความผิดพลาดทางตรรกะ (logic error) โดยความผิดพลาดในการทำงานนั้นจะทำให้โปรแกรมหยุดการทำงาน หรือยกเลิกการทำงาน ขณะที่ความผิดพลาดทางตรรกะนั้นโปรแกรมจะสามารถทำงานต่อไปได้แต่จะให้ผลการทำงานที่ผิดพลาด ดังเช่นรูปตัวอย่างที่ 2.2

```
public static double computeAverage(double[] array){
   double total=0.0;
   if(array==null||array.length==0){
      return 0.0;
   }else{
      for(int i=0;i< array.length  ;i++){
         total= total+array[i];
      }
   }
   return total/array.length;
}
```

รูปที่ 2.2 ก) เป็นรหัสต้นฉบับที่ทำงานได้อย่างถูกต้อง

```
public static double computeAverage(double[] array){
   double total=0.0;
   for(int i=0;i  <  array.length  ;i++){
      total= total-  array[i];
 }
   return total/ array.length;

}
```

รูปที่ 2.2 ข) เป็นรหัสต้นฉบับที่มีความผิดพลาดทางความหมาย

จากรหัสตัวอย่างในรูปที่ 2.2 ข) เป็นตัวอย่างโปรแกรมที่มีความผิดพลาดทางความหมายสองประการคือ

1) ใช้ตัวดำเนินการที่ผิดพลาดเพราะในการหาค่าเฉลี่ยจะต้องหาจากค่าของผลรวม เป็นความผิดพลาดทางตรรกะ

2) ขาดการตรวจสอบข้อมูลเข้า ทำให้เกิดความผิดพลาดขึ้นเมื่อข้อมูลเข้ามีค่าเป็น null เป็นความผิดพลาดในขณะทำงาน

โดยในความผิดพลาดทางความหมายนั้น A. Barr [4] ได้จัดแบ่งข้อบกพร่องซึ่งเป็นสาเหตุของความผิดพลาดทางความหมายออกเป็นหมวดหมู่ ตามสาเหตุที่ K. Donald [5] ได้เสนอไว้ดังนี้

1) กลุ่มของข้อบกพร่องที่เกิดขึ้นจากขั้นตอนวิธีการแก้ปัญหา ได้แก่
   3.1) ปัญหาการผิดพลาดอยู่หนึ่ง (off-by-one) เป็นข้อบกพร่องที่ทำให้เกิดการคำนวณที่ผิดพลาดไปอาจจะมากกว่าหรือน้อยกว่าอยู่หนึ่ง
   3.2) ปัญหาทางตรรกะ (logic) เป็นข้อบกพร่องที่เกิดขึ้นจากการตรวจสอบเงื่อนไขข้อกำหนดหรือตรรกะที่ผิดพลาด
   3.3) ปัญหาการตรวจสอบความสมเหตุสมผล (validation) เป็นข้อบกพร่องที่เกิดขึ้นจากการขาดการตรวจสอบขอบเขตหรือค่าของตัวแปรที่ดี ได้แก่ ปัญหาการหารด้วยศูนย์ เป็นต้น

2) กลุ่มของข้อบกพร่องที่เกิดขึ้นจากการที่ข้อมูลไม่ได้รับการประมวลผลอย่างถูกต้องเหมาะสม ได้แก่
   3.1) ปัญหาดัชนี (index) เป็นข้อบกพร่องที่เกิดจากการอ้างดัชนีของอาร์เรย์ที่ผิดพลาด
   3.2) ปัญหาขอบเขต (limit) เป็นข้อบกพร่องที่เกิดจากการประมวลผลจุดเริ่มต้นหรือจุดสุดท้ายของข้อมูลผิดพลาด
   3.3) ปัญหาการคำนวณทางตัวเลข (number) เป็นข้อบกพร่องที่เกิดจากปัญหาการเก็บค่าของตัวเลข ได้แก่ ข้อบกพร่องในการคำนวณเลขทศนิยม เป็นต้น

3) กลุ่มของข้อบกพร่องที่เกิดขึ้นจากความผิดพลาดทั่วไปในการเขียน ได้แก่
   3.1) ปัญหาเงื่อนไขเริ่มต้น (initial) เป็นข้อบกพร่องที่เกิดจากการขาดการกำหนดค่าเริ่มต้นให้กับตัวแปร
   3.2) ปัญหาทางตำแหน่งหรือลำดับ (location) เป็นข้อบกพร่องที่เกิดจากการวางข้อความประมวลผลผิดตำแหน่ง

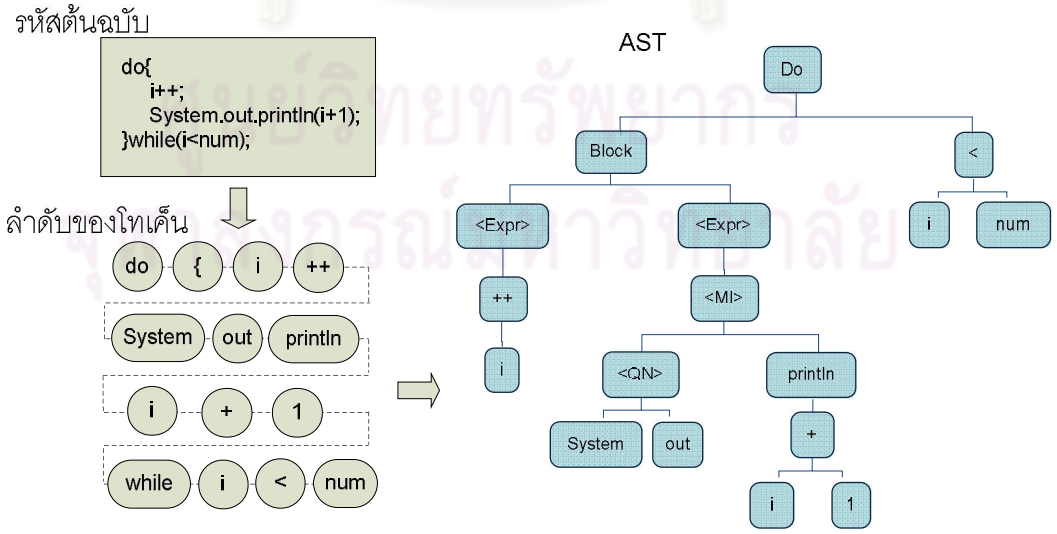
3.3) ปัญหาเกี่ยวกับตัวแปร (variable) เป็นข้อบกพร่องที่เกิดขึ้นจากการใช้ชื่อของตัวแปรผิดพลาด อาจเนื่องมาจากความคล้ายคลึงของชื่อตัวแปร

3.4) ปัญหาทางการใช้นิพจน์ (expression) เป็นข้อบกพร่องที่เกิดขึ้นจากความสับสนของนักเขียนโปรแกรมต่อการใช้นิพจน์ที่ผิดพลาด ได้แก่ '==', 'equals()' เป็นต้น

ในการการเขียนโปรแกรมจริงอาจเป็นการยากที่จะแยกความผิดพลาดต่างๆออกจากกันโดยเด็ดขาดเพราะความผิดพลาดบางประการนำมาซึ่งความผิดพลาดอื่นๆ หรือความผิดพลาดหลายประการอาจเกิดขึ้นพร้อมๆกัน

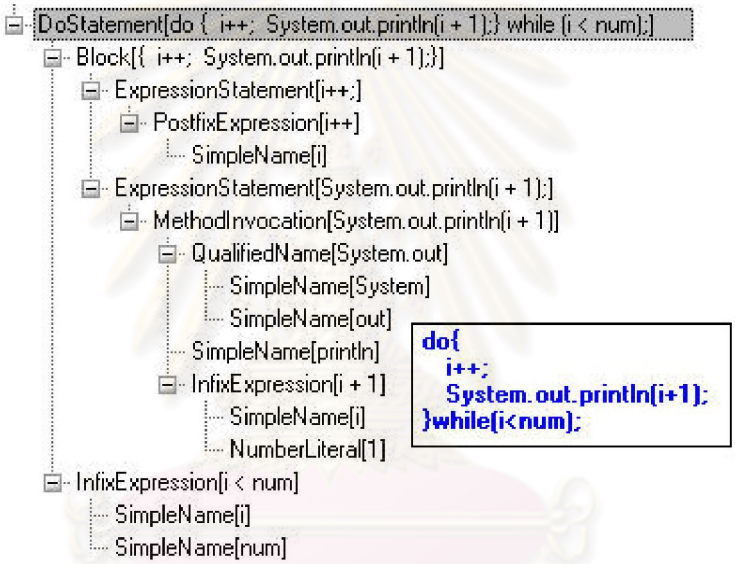
2.1.4 **โครงสร้างต้นไม้ไวยากรณ์เชิงนามธรรม** [6]

ภาษาคอมพิวเตอร์นั้นถูกสร้างขึ้นโดยอ้างอิงกับกฎและไวยากรณ์ของภาษาอย่างเคร่งครัด โดยทั่วไปแล้วกฎของภาษาคอมพิวเตอร์จะถูกแบ่งออกเป็นสองส่วนเพื่อง่ายแก่การแปลโปรแกรม คือ ส่วนที่เกี่ยวข้องกับกฎของคำสำคัญ และกฎที่เกี่ยวข้องกับการประกอบกันของคำสำคัญหรือไวยากรณ์ของภาษาคอมพิวเตอร์ ซึ่งในขั้นตอนการแปลโปรแกรม โปรแกรมจะเริ่มทำงานจากการวิเคราะห์กฎของคำสำคัญจากลำดับของตัวอักษรให้อยู่ในลักษณะของกลุ่มของลำดับอักขระที่มีความหมายเรียกว่าโทเคน (token) โดยการแปลผลจะขึ้นอยู่กับไวยากรณ์ของศัพท์ หลังจากนั้นก็จะถูกแปลผลโดยอาศัยไวยากรณ์ของภาษาทำให้อยู่ในรูปแบบของโครงสร้างต้นไม้ที่เรียกว่า Concrete Syntax Tree (CST) หรือ Abstract Syntax Tree (AST) ดังรูปที่ 2.3



รูปที่ 2.3 การเปลี่ยนรหัสต้นฉบับให้อยู่ในรูปแบบของ AST

โดยโครงสร้างต้นไม้ไวยากรณ์เชิงนามธรรม (Abstract Syntax Tree) นั้นจะเป็นโครงสร้างต้นไม้ของโปรแกรมที่แสดงถึงข้อมูลของการทำงานหรือพฤติกรรมของโปรแกรม แต่ไม่มีส่วนที่แสดงถึงโครงสร้างไวยากรณ์ของภาษา ดังนั้นโครงสร้างต้นไม้ไวยากรณ์เชิงนามธรรมจึงเป็นการแทนโปรแกรมคอมพิวเตอร์ด้วยโครงสร้างต้นไม้ซึ่งแต่ละปมของต้นไม้จะแสดงถึงองค์ประกอบย่อยของรหัสภายในโปรแกรมด้วยการประกอบกันของปมลูกอย่างมีความหมาย ดังเช่นรูปที่ 2.4 เป็นรูปที่แสดงถึงการทำงานแบบวนซ้ำของต้นไม้ไวยากรณ์เชิงนามธรรมของโปรแกรมคอมพิวเตอร์ภาษาจาวาที่ได้จากการแปลโปรแกรมภายใต้การทำงานของโปรแกรมสภาพแวดล้อมสำหรับการพัฒนาโปรแกรม Eclipse IDE



รูปที่ 2.4 AST ของ do-while จากโปรแกรม ASTParser ของ Eclipse IDE ซึ่งเป็นโปรแกรมที่ช่วยในการเขียนโปรแกรม

2.1.4.1 Eclipse AST [7]

Eclipse AST เป็นกรอบการทำงานพื้นฐานสำหรับเครื่องมือช่วยเหลืออันทรงพลังต่างๆ มากมายของโปรแกรม Eclipse IDE เช่น เครื่องมือการทำ Refactoring, Quick Find และ quick assist ทำให้ Eclipse IDE กลายเป็นโปรแกรมสภาพแวดล้อมสำหรับการพัฒนาโปรแกรมภาษาจาวาแบบเบ็ดเสร็จที่มีความสมบูรณ์เป็นอย่างมาก ในการทำงาน Eclipse จะทำการส่งผ่านรหัสต้นฉบับของโปรแกรมภาษาจาวาให้เปลี่ยนเป็นโครงสร้างต้นไม้ ซึ่งการจัดการแก้ไขเปลี่ยนแปลงบนโครงสร้างต้นไม้เหล่านี้มีความสะดวกสบายและน่าเชื่อถือในการจัดการ ทำให้การวิเคราะห์แก้ไข และเปลี่ยนแปลงรหัสต้นฉบับของโปรแกรมเป็นไปอย่างมีประสิทธิภาพมากกว่าการเข้าไปจัดการโดยตรงกับรหัสต้นฉบับของโปรแกรมแบบตัวอักษร

ตัวอย่างในการประยุกต์การใช้งาน Eclipse AST เช่น ในการเขียนโปรแกรมภาษาจาวา ผู้เขียนโปรแกรมไม่ควรประกาศตัวแปรท้องถิ่นก่อนการใช้งานซึ่งเราสามารถเขียนโปรแกรมประยุกต์เพื่อตรวจสอบการประกาศตัวแปรต่างๆ และทำการย้ายการประกาศตัวแปรที่มีการประกาศที่ไม่ถูกต้องให้ไปอยู่ในตำแหน่งที่ถูกต้องได้ ดังนั้นในการทำงานของโปรแกรมประยุกต์ที่จะถูกเขียนขึ้นนี้จะต้องมีความสามารถในการทำงานดังนี้คือ

1) ลบการประกาศตัวแปรที่ไม่จำเป็นออก ซึ่งถ้าตัวแปรเหล่านั้นถูกสร้างขึ้นและกำหนดค่าตั้งต้นเพื่อให้ถูกกำหนดค่าลบล้างในตอนหลัง

2) การย้ายตำแหน่งการประกาศตัวแปร ถ้าตัวแปรถูกประกาศขึ้นโดยยังไม่ได้มีการอ้างอิงหรือใช้งานในทันที ควรจะมีการย้ายตำแหน่งการประกาศตัวแปรไปยังบรรทัดที่เหมาะสมก่อนการใช้งานตัวแปรนั้นๆ

ซึ่งการประยุกต์การใช้งาน Eclipse AST เพื่อการแก้ไขรหัสต้นฉบับแบบอัตโนมัตินั้นจะมีขั้นตอนการทำงาน ดังนี้

1) รหัสต้นฉบับของโปรแกรมภาษาจาวา ในการเริ่มต้นการทำงานจะเริ่มต้นจากรหัสต้นฉบับของโปรแกรมภาษาจาวาที่จะทำการปรับปรุง โดยรหัสต้นฉบับนี้จะอยู่ในรูปแบบของแฟ้มข้อมูล หรือ ลำดับแถวอักขระก็ได้

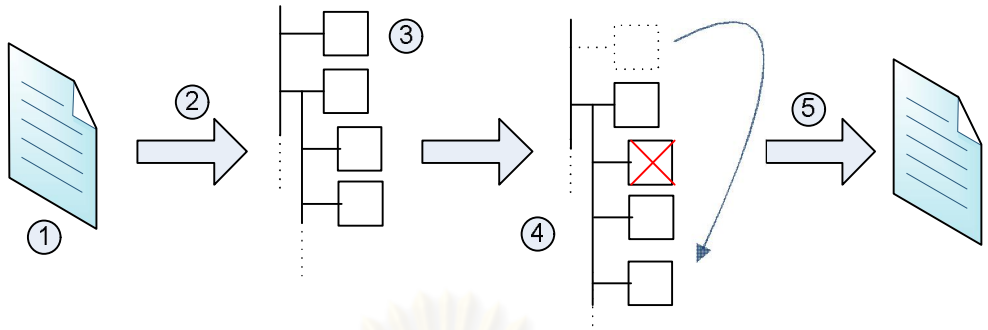
2) แปลงผลและแจงส่วน รหัสต้นฉบับของโปรแกรมที่จะทำการปรับปรุงนี้จะถูกแปลงผลและแจงส่วน โดยอาศัยการทำงานของคลาส org.eclipse.jdt.core.dom.ASTParser

3) ASTของรหัสต้นฉบับ จากขั้นตอนการแปลงผลและแจงส่วนจะได้ผลลัพธ์เป็นโครงสร้างต้นไม้ไวยากรณ์เชิงนามธรรม ซึ่งเป็นโครงสร้างต้นไม้ที่แสดงถึงโครงสร้างการทำงานทั้งหมดของโปรแกรมที่ได้นำมาแปลงผล

4) การจัดการดำเนินการเปลี่ยนแปลงโครงสร้างต้นไม้ของรหัสต้นฉบับ เมื่อได้โครงสร้างต้นไม้ไวยากรณ์เชิงนามธรรมของโปรแกรมมาแล้วจะเข้าทำการแก้ไขโครงสร้างของต้นไม้โดยตรงเพื่อทำให้เกิดการเปลี่ยนแปลงในรหัสต้นฉบับของโปรแกรมตามแบบอย่างที่ต้องการ

5) แปลงผลการเปลี่ยนแปลงกลับมาอยู่ในรูปแบบของรหัสต้นฉบับ หลังจากได้โครงสร้างต้นไม้อย่างที่ต้องการแล้วจะทำการแปลงผลกลับเป็นรหัสต้นฉบับในรูปแบบตัวอักษรเหมือนเดิม

โดยในงานวิจัยนี้จะใช้ประโยชน์จากการการเปลี่ยนแปลงโครงสร้างต้นไม้ไวยากรณ์ของรหัสต้นฉบับนี้เพื่อการเปลี่ยนแปลงกระบวนการทำงานของโปรแกรมแต่ยังสามารถรักษาความถูกต้องของไวยากรณ์ของภาษาไว้ได้ เพื่อให้ตรงกับจุดประสงค์ในการมุ่งเน้นที่ความผิดพลาดเชิงความหมาย หรือความผิดพลาดเนื่องจากขั้นตอนการทำงานมากกว่าความผิดพลาดเชิงไวยากรณ์



รูปที่ 2.5 ขั้นตอนการทำงานในการเปลี่ยนแปลงรหัสต้นฉบับของโปรแกรม

2.1.5 กลวิธีการกลายรหัส

การกลายรหัสเพื่อการทดสอบ (mutation testing) [6] เป็นวิธีการสำหรับการทดสอบโปรแกรม ซึ่งจะมีกระบวนการกลาย (mutation) หรือการเปลี่ยนแปลงรหัสต้นฉบับของโปรแกรมเล็กน้อย ในการเปลี่ยนแปลงนั้นการดำเนินการจะถูกอ้างอิงจากตัวดำเนินการกลายรหัส (mutation operators) โดยการกลายรหัสเพื่อการทดสอบนั้นถูกเสนอขึ้นเพื่อระบุและค้นหาจุดอ่อนของชุดข้อมูลที่จะใช้ในการทดสอบระบบ จากแนวความคิดที่ว่าถ้ามีการกลายรหัสต้นฉบับแล้วไม่สามารถทำให้เกิดการเปลี่ยนแปลงใดๆ โดยปรกติแล้วจะหมายถึงการเปลี่ยนแปลงที่เกิดขึ้นกับผลลัพธ์ จะแสดงว่า ชุดข้อมูลที่นำมาใช้ในการทดสอบระบบนั้นไม่สามารถครอบคลุมการทำงานของโปรแกรมได้ การกลายรหัสนั้นเกิดขึ้นในส่วนของรหัสที่ไม่ถูกใช้งาน รหัสที่ถูกเขียนขึ้นมาเกินความจำเป็น หรือการกลายรหัสที่เกิดขึ้นนั้นเป็นรหัสที่มีการทำงานเหมือนกับรหัสก่อนการเปลี่ยนแปลง เรียกรหัสที่มีการทำงานที่เหมือนกันนี้ว่าเป็นรหัสที่สมมูลกัน โดยจำนวนครั้งของการกลายรหัสจะมากขึ้นตามขนาดของโปรแกรมที่นำมาทดสอบ

การกลายรหัสต้นฉบับเพื่อการทดสอบนั้นสามารถทำได้โดยการเลือกชนิดของตัวดำเนินการที่ต้องการเปลี่ยนแปลงแล้วทำการกลายรหัสต้นฉบับโดยทุกๆครั้งที่ทำการกลายรหัสจะเรียกส่วนของผลที่เกิดจากการแปลงว่า mutant และถ้าชุดของข้อมูลที่ใช้ในการทดสอบสามารถตรวจสอบพบการเปลี่ยนแปลงได้จะเรียกว่าการเปลี่ยนแปลงนั้นถูกพบ ดังเช่นรูปตัวอย่างที่2.6

```
if (a && b)
   c = 1;
else
   c = 0;
```

รูปที่ 2.6 ก)

```
if (a || b)
   c = 1;
else
   c = 0;
```

รูปที่ 2.6 ข)

รูปที่ 2.6 ก) รหัสต้นฉบับก่อนทำการแปลง

ข) รหัสต้นฉบับหลังการแปลง

เมื่อแทนที่ตัวดำเนินการ '&&' ด้วย '||' ชุดข้อมูลที่ใช้ในการทดสอบการกลายรหัสดังรูปตัวอย่างที่ 2.6 จะถือว่าผ่านการทดสอบเมื่อผ่านเงื่อนไขดังนี้

1) ชุดของข้อมูลเข้าต้องทำให้เกิดการทำงานที่แตกต่างกันระหว่างก่อนและหลังทำการเปลี่ยนแปลง เช่นเมื่อกำหนดให้ a และ b ในตัวอย่างมีค่าเป็นจริง และเท็จตามลำดับจะทำให้เกิดการทำงานที่แตกต่างกัน

2) ผลของความแตกต่างที่เกิดจากการกลายรหัสต้นฉบับต้องส่งผลต่อข้อมูลออก เช่นค่าของตัวแปร c ในตัวอย่างที่เปลี่ยนแปลงไป จะสามารถถูกตรวจพบได้เมื่อมีการแสดงค่าออกมาทางหน้าจอ

โดยกำหนดให้การกลายรหัสเพื่อการทดสอบอย่างอ่อน (weak mutation testing) นั้น ข้อมูลที่ถือว่าผ่านการทดสอบต้องการเพียงเงื่อนไขแรกในการผ่านการทดสอบ แต่ถ้าเป็นการกลายรหัสเพื่อการทดสอบอย่างเข้ม (strong mutation testing)นั้นข้อมูลต้องผ่านทั้งสองเงื่อนไขจึงจะถือว่าผ่านการทดสอบ โดยที่การกลายรหัสนั้นต้องไม่อยู่ในรูปแบบที่สมมูลกัน(equivalent mutant) เช่น

```
int index=0;
while (...){
          . . .
   index++;
   if (index==10)
    break;
}
```

รูปที่ 2.7 ก)

```
int index=0;
while (...){
    . . .;
   index++;
   if (index>=10)
      break;
}
```

รูปที่ 2.7 ข)

รูปที่ 2.7 ก) รหัสต้นฉบับก่อนการกลายรหัส

ข) รหัสต้นฉบับหลังการกลายรหัส

ดังรูปตัวอย่างที่ 2.7 เมื่อทำการเปลี่ยนตัวดำเนินการแบบเปรียบเทียบความสัมพันธ์จาก '==' เป็น '>=' ซึ่งจะพบว่าผลจากการเปลี่ยนแปลงโปรแกรมจะทำงานเหมือนกันทุกประการ ทำให้ไม่สามารถหาชุดข้อมูลที่จะทดสอบผลของการกลายรหัสต้นฉบับที่เกิดการแปลงแบบสมมูลกันได้

2.2 เอกสารและงานวิจัยที่เกี่ยวข้อง

2.2.1 งานวิจัยที่เกี่ยวข้องกับการเรียนการสอนวิชาพื้นฐานการเขียนโปรแกรม

ถึงแม้จะเป็นที่ทราบกันมานานถึงความสำคัญของการส่งเสริมทักษะการเขียนโปรแกรมที่ดีให้แก่นักเรียน แต่งานวิจัยด้านการเรียนการสอนทางด้านวิทยาศาสตร์คอมพิวเตอร์ (Computer

Science Education) ก็ยังเป็นงานที่ใหม่ และมีงานวิจัยในด้านนี้ออกมาน้อยมากเมื่อเทียบกับงานวิจัยในกลุ่มอื่นๆ ส่วนหนึ่งของงานวิจัยเหล่านี้เป็นงานที่พยายามทำการศึกษาเพื่อที่จะอธิบายถึงปัจจัยที่มีผลต่อความสามารถในการเรียนรู้ ทักษะในด้านต่างๆของการเขียนโปรแกรมของนักเรียนในระดับเริ่มต้น การอ่านทำความเข้าใจรหัสต้นฉบับ การทำความเข้าใจขั้นตอนการทำงาน การเขียนรหัสต้นฉบับ การออกแบบ ความสามารถ พฤติกรรม และความสัมพันธ์ที่ก่อให้เกิดการเรียนรู้ของนักเรียน เช่น

P. Arnold และกลุ่ม[8] ได้ทำการศึกษารวบรวมวรรณกรรมศึกษาและผลงานวิจัยต่างๆในด้านการเรียนการสอนพื้นฐานการเขียนโปรแกรม โดยมีวัตถุประสงค์เพื่อรวบรวมงานวิจัยและรายงานถึงลักษณะขั้นตอนในงานวิจัย สำหรับการออกแบบหลักสูตรหรือรายวิชาการเรียนการสอนพื้นฐานการเขียนโปรแกรมและการพัฒนาการเรียนการสอนในด้านต่างๆเช่น ภาษาคอมพิวเตอร์ที่ควรเลือกใช้ การเลือกเครื่องมือและสภาพแวดล้อมที่จะช่วยส่งเสริมการเรียนการสอนและการใช้เครื่องมือเหล่านั้นในการส่งเสริมการเรียนการสอน การทดสอบและประเมินผลการเรียนการสอน และจะทำอย่างไรให้การเรียนการสอนพื้นฐานการเขียนโปรแกรมนั้นเหมาะสมสำหรับการเรียนการสอนในระดับต่อๆไป ทั้งสามารถนำไปใช้งานจริงได้ ซึ่งจะเป็นประโยชน์กับผู้ที่เกี่ยวข้องกับองค์กรทางด้านการศึกษาที่เกี่ยวข้องกับการสอนวิชาคอมพิวเตอร์ หรือผู้ที่ต้องการวางแผน ออกแบบ พัฒนา แก้ไขปรับปรุงหรือผู้ที่ต้องการนำข้อมูลไปประยุกต์ใช้ในการเรียนการสอนพื้นฐานการเขียนโปรแกรม โดยงานวิจัยนี้ได้ทำการจัดกลุ่มวรรณกรรมศึกษาและงานวิจัยที่เกี่ยวข้องกันออกเป็นกลุ่มต่างๆกันดังนี้คือ

1) ด้านหลักสูตรการเรียนการสอน
2) ด้านการเรียนการสอน
3) ด้านภาษา
4) ด้านเครื่องมือและสภาพแวดล้อม

โดยในรายละเอียดพบว่าถึงแม้งานวิจัยด้านการศึกษาการเรียนการสอนการเขียนโปรแกรมจะเป็นที่สนใจแต่กลับมีงานวิจัยน้อยมากที่ให้ความสนใจกับการพัฒนาทักษะและความเข้าใจการแก้ไขข้อบกพร่องที่เกิดขึ้นกับโปรแกรม

McCracken [9] และ Leeds Working Group [10] ต่างก็ได้พยายามที่จะศึกษาถึงความสามารถในการเขียนโปรแกรมของนักเรียนหลังผ่านการเรียนวิชาการเขียนโปรแกรมระดับพื้นฐานทั้งในทักษะด้าน การอ่านรหัสต้นฉบับของโปรแกรม การทำความเข้าใจต่อการ

ทำงาน ทักษะการเขียนโปรแกรม การประกอบกันของโปรแกรมและการออกแบบโปรแกรมเพื่อใช้ในการแก้ปัญหา จากการตั้งโจทย์ปัญหาต่างๆเพื่อใช้ในการทดสอบเช่น ให้นักเรียนทำการประเมินค่าของตัวแปรต่างๆจากส่วนสั้นๆของรหัสต้นฉบับ ให้นักเรียนหาข้อบกพร่องจากตัวอย่างแล้วแก้ไขให้ถูกต้อง การให้นักเรียนเขียนรหัสต้นฉบับให้สมบูรณ์เพื่อใช้ในการแก้ปัญหาจากบางส่วนของรหัสต้นฉบับที่กำหนดให้ หรือให้นักเรียนออกแบบโปรแกรมเพื่อใช้ในการแก้ปัญหาอย่างง่าย ฯลฯ พบว่านักเรียนส่วนใหญ่จะมีคะแนนในส่วนการหา การแก้ข้อบกพร่อง การเติมรหัสต้นฉบับให้สมบูรณ์และการออกแบบโปรแกรมอยู่ในระดับต่ำ เมื่อทำการสอบถามถึงปัญหาของนักเรียนพบว่านักเรียนที่ได้คะแนนน้อยจะบอกว่ามีประสบการณ์ในการเขียนโปรแกรมค่อนข้างน้อย

ในงานวิจัยของ G. Lee [11] ได้สร้างระบบ Debug It เป็นระบบที่มีส่วนต่อประสานกราฟิกกับผู้ใช้เพื่อช่วยฝึกฝนการแก้ไขโปรแกรมจากฐานข้อมูลโจทย์ปัญหา โดยมุ่งเน้นที่จะให้นักเรียนเข้าใจและแก้ไขความเข้าใจที่ผิดพลาดต่อการเขียนโปรแกรม และพัฒนาทักษะในการแก้ข้อบกพร่อง ต่อมา A. Marzieh [12] ได้ศึกษาผลของการสอนวิชาคอมพิวเตอร์เบื้องต้นผ่านแบบฝึกหัดการแก้ข้อบกพร่องของโปรแกรมแล้วทำการศึกษาผลจากการพัฒนาของนักเรียนในสองแนวทางคือ จำนวนของข้อบกพร่องที่ตรวจพบ และคะแนนของนักเรียนที่สามารถทำได้ผ่านการเปรียบเทียบกับนักเรียนปีที่ได้รับและไม่ได้รับการศึกษาการเรียนการเขียนโปรแกรมผ่านแบบฝึกหัดการแก้ไขข้อบกพร่อง จากข้อมูลชี้ให้เห็นว่าการศึกษาการเขียนโปรแกรมผ่านแบบฝึกหัดการแก้ไขข้อบกพร่องช่วยให้นักเรียนลดความผิดพลาดที่เกิดขึ้นในโปรแกรมได้อย่างมีนัยสำคัญพร้อมกับคะแนนในการทดสอบที่มากขึ้น งานวิจัยเหล่านี้ได้แสดงให้เห็นถึงความสำคัญของการฝึกฝนทักษะการแก้ข้อบกพร่องของโปรแกรมโดยที่งานวิจัยของ G. Lee และ A. Marzieh ยังได้แสดงให้เห็นถึงสิ่งที่เหมือนกันก็คือนักเรียนรู้สึกชื่นชอบการเรียนที่มีแบบฝึกฝนที่ช่วยให้เห็นถึงข้อบกพร่องที่เกิดขึ้นได้ในการเขียนโปรแกรม

2.2.2 งานวิจัยที่เกี่ยวข้องกับรูปแบบความผิดพลาดลักษณะต่างๆและการแก้ไขข้อบกพร่อง

“เพราะว่ารหัสต้นฉบับของโปรแกรมที่ถูกต้องไม่ได้หล่นลงมาจากฟ้า” Matthew C.[13] จึงได้ทำการวิจัยกระบวนการในการเขียนโปรแกรมของนักเรียนที่เพิ่งเริ่มต้นเขียนโปรแกรม เพื่อศึกษาพฤติกรรมในการเขียนโปรแกรมและพฤติกรรมในการแปลผลโปรแกรมของนักเรียนจากระบบอัตโนมัติในห้องปฏิบัติการเขียนโปรแกรมคอมพิวเตอร์ ซึ่งทำให้เขาพบรายละเอียดชนิดของข้อบกพร่องและการกระจายตัวของการตรวจพบข้อบกพร่องในลักษณะต่างๆ ในขณะที่งานวิจัย

ของ P. j. Vipindeep V [14]ได้ทำการรวบรวมข้อบกพร่องทั่วไปที่มักเกิดขึ้นในการเขียนโปรแกรม และตัวอย่างของโปรแกรมที่ผิดพลาดในลักษณะต่างๆ 28ลักษณะโดยมีจุดประสงค์เพื่อให้ผู้เขียนโปรแกรมรับรู้ข้อบกพร่องในลักษณะต่างๆที่มักจะเกิดขึ้นและสามารถนำไปใช้ในการป้องกันข้อบกพร่องในโปรแกรมที่จะเขียนขึ้นได้ ทั้งนี้ยังมีข้อเตือนที่มักพบในหนังสือที่เกี่ยวกับการสอนการเขียนโปรแกรมและบทความอีกมากมายบนอินเตอร์เน็ตที่กล่าวถึงลักษณะข้อบกพร่องต่างๆที่ต้องระมัดระวังเพราะมักจะเกิดขึ้นในโปรแกรมเสมอๆ ซึ่งในงานวิจัยนี้ได้รวบรวมข้อบกพร่องต่างๆที่มักเกิดขึ้นเหล่านี้ไว้เป็นข้อมูลในการวิเคราะห์และสร้างลักษณะของข้อบกพร่องที่จะทำการเติมลงในโปรแกรม

2.2.3 งานวิจัยที่เกี่ยวข้องกับการแก้ไขรหัสต้นฉบับอัตโนมัติ

Yu-Seung Ma, Jeff Offutt and Yong-Rae Kwon [15, 16, 17] ได้ร่วมกันพัฒนา μJava ขึ้นเป็นระบบการกลายรหัสต้นฉบับของภาษาจาวาที่ประกอบด้วยระดับของการเปลี่ยนแปลงสองระดับทั้งระดับเมท็อดซึ่งมีตัวดำเนินการในการเปลี่ยนแปลง 12 ตัวดำเนินการใน 6 กลุ่มและระดับคลาสซึ่งมีตัวดำเนินการ 29 ตัวดำเนินการใน 2 กลุ่ม

1) ตัวดำเนินการแปลงรหัสระดับเมท็อด จะมีกลุ่มของตัวดำเนินการหลักๆได้แก่
   1.1) ตัวดำเนินการทางคณิตศาสตร์ (arithmetic operator) โดยในภาษาจาวาประกอบด้วยตัวดำเนินการกับตัวเลขทั้งที่เป็นชนิดจำนวนเต็มและทศนิยมคือ ‘+’, ‘-’, ‘*’, ‘/’ และ ‘%’ ซึ่งเป็นตัวดำเนินการทวิภาค แต่ ‘+’และ ‘-’ สามารถเป็นตัวดำเนินการเอกภาคได้ และตัวดำเนินการทางคณิตศาสตร์ที่เป็นอย่างย่อคือ op++, ++op, op-- และ –op
   1.2) ตัวดำเนินการเปรียบเทียบหรือความสัมพันธ์ระหว่างค่าสองค่า (relational operator) โดยในภาษาจาวาประกอบด้วยตัวดำเนินการ ‘>’, ‘>=’, ‘<’, ‘<=’, ‘==’ และ ‘!=’
   1.3) ตัวดำเนินการทางเงื่อนไข (conditional operator) ในภาษาจาวาประกอบด้วยตัวดำเนินการ ‘&&’, ‘||’, ‘&’, ‘|’ และ ‘^’ ซึ่งเป็นตัวดำเนินการทวิภาค และตัวดำเนินการเอกภาคคือ '!'
   1.4) ตัวดำเนินการเปลี่ยนตำแหน่ง (shift operator) เป็นตัวดำเนินการที่ใช้ในการเลื่อนบิตของข้อมูล ในภาษาจาวาประกอบด้วยตัวดำเนินการ ‘>>’, ‘<<’, and ‘>>>’

1.5) ตัวดำเนินการทางตรรกะ (logical operator) โดยประกอบไปด้วยตัวดำเนินการทวิภาค ‘&’, ‘|’ and ‘^’ และตัวดำเนินการเอกภาค ‘~’

1.6) ตัวดำเนินการกำหนดค่า (assignment operator) คือ ‘=’ และในรูปแบบย่อซึ่งประกอบด้วย ‘+=’, ‘-=’, ‘*=’, ‘/=’, ‘%=’, ‘&=’, ‘|=’, ‘^=’, ‘<<=’, ‘>>=’ และ ‘>>>=’

2) ตัวดำเนินการแปลงรหัสระดับคลาส จะมีกลุ่มของตัวดำเนินการหลักๆได้แก่

2.1) ตัวดำเนินการเข้าถึง (Access Control) ในปัญหาการควบคุมการเข้าถึงเป็นเรื่องที่นักเขียนโปรแกรมทำผิดเป็นประจำในการพัฒนาโปรแกรมเชิงวัตถุ ความหมายของระดับของการเข้าถึงเป็นที่สิ่งเข้าใจผิดบ่อยครั้ง การเข้าถึงตัวแปรและเมท็อดก็ขาดการใส่ใจขณะทำการออกแบบ ถึงแม้ว่าการขาดการนิยามที่ถูกต้องของการควบคุมการเข้าถึงไม่ได้เป็นเหตุให้เกิดความผิดพลาดขึ้นทันทีแต่ก็นำมาซึ่งการพฤติกรรมการทำงานที่ผิดพลาดของวัตถุ เมื่อทำงานร่วมกับวัตถุอื่นๆ เมื่อมีการเปลี่ยนแปลงหรือเมื่อมีการสืบทอดคุณสมบัติของวัตถุในการพัฒนาโปรแกรม

2.2) ตัวดำเนินการรับทอด (Inheritance) ถึงแม้ว่าการสืบทอดคุณสมบัติจะมีประโยชน์อย่างมาก แต่การใช้งานอย่างไม่ถูกต้องก็สามารถนำมาซึ่งความผิดพลาดได้ ทั้งการอ้างถึงตัวแปรต่างๆจากการสืบทอด การเข้าครอบงำพฤติกรรมที่สืบทอดมา การอ้างถึงคลาสแม่และตัวดำเนินการสร้างวัตถุ มักเป็นสิ่งที่เกิดความผิดพลาดขึ้นเป็นประจำ

2.3) ตัวดำเนินการภาวะพหุสัณฐาน (Polymorphism) จากคุณสมบัติ ภาวะพหุสัณฐานทำให้การเขียนโปรแกรมเชิงวัตถุสามารถอ้างถึงคลาสที่ต่างกันในแต่ละครั้งของการทำงานหรือในเวลาที่ต่างกันในแต่ละครั้งของการทำงาน ซึ่งทำให้เกิดพฤติกรรมของการทำงานที่แตกต่างกันไปตามความแตกต่างของการอ้างถึง หรือ คุณสมบัติที่อนุญาตให้สามารถมีเมท็อดที่มีชื่อเดียวกันภายใต้คลาสเดียวกันได้ตราบเท่าที่เมท็อดเหล่านั้นมีอาร์กิวเมนต์ที่แตกต่างกันเป็นคุณสมบัติที่สำคัญจึงต้องมีการทดสอบเพื่อความถูกต้องของโปรแกรม

2.4) ตัวดำเนินการต่อลักษณะพิเศษของโปรแกรมภาษาจาวา (Language-Specific Features) เพราะว่าการทดสอบด้วยการเปลี่ยนแปลงนั้นขึ้นอยู่กับตัวดำเนินการเปลี่ยนแปลงซึ่งต้องสอดคล้องกับระเบียบข้อบังคับของภาษา ตัวอย่างเช่น java ได้

มีการนิยามคำที่แตกต่างจากภาษาทั่วไปเช่น this, static, default constructors และ การกำหนดค่าเริ่มต้น จึงต้องมีการทดสอบความถึงความถูกต้องด้วย

การเปลี่ยนแปลงรหัสต้นฉบับเป็นการเลียนแบบความผิดพลาดที่เกิดขึ้นจากนักเขียนโปรแกรม ซึ่งสามารถเกิดขึ้นได้โดยไม่ขึ้นว่าผู้เขียนโปรแกรมเป็นนักเรียนหรือเป็นผู้ที่มีความเชี่ยวชาญในการเขียนโปรแกรม

2.2.4 งานวิจัยในการประยุกต์ใช้โครงสร้างต้นไม้ไวยากรณ์เชิงนามธรรม

Reiss,S.P.[18] ได้พัฒนาระบบสำหรับตรวจสอบรหัสต้นฉบับหรือส่วนของรหัสต้นฉบับที่ไม่ได้มีการใช้งานและมีโอกาสนำมาซึ่งการทำงานที่ผิดพลาดของโปรแกรมขึ้น โดยกระบวนการการค้นหารหัสของโปรแกรมที่ไม่ถูกใช้งานนี้ระบบจะทำการแปลงผลและแจงส่วนเพื่อเปลี่ยนรูปแบบของรหัสของโปรแกรมในรูปแบบอักษรให้เป็นโครงสร้างต้นไม้ไวยากรณ์เชิงนามธรรม แล้วทำการเปรียบเทียบรูปแบบโครงสร้างที่ได้กับโครงสร้างต้นไม้ไวยากรณ์เชิงนามธรรมที่ได้วิเคราะห์โครงสร้างไว้ก่อนแล้วในคลังข้อมูลเพื่อวิเคราะห์หาส่วนของโครงสร้างต้นไม้หรือรูปแบบในโครงสร้างต้นไม้ที่มีโอกาสไม่ถูกใช้งาน ซึ่งในงานวิจัยนี้ได้เลือกใช้ Eclipse AST เป็นโครงสร้างต้นไม้ไวยากรณ์เชิงนามธรรมในการพัฒนา ซึ่งงานวิจัยนี้ได้กล่าวว่าโครงสร้างต้นไม้ไวยากรณ์เชิงนามธรรมของ Eclipse IDE เป็นโครงสร้างต้นไม้ที่มีความสมบูรณ์สำหรับรหัสต้นฉบับของโปรแกรมภาษาจาวา และยังมีเครื่องมือสำหรับการใช้งานกับโครงสร้างนี้ที่หลากหลาย ทำให้เป็นตัวเลือกที่ดีที่สุดสำหรับงานที่ต้องการอาศัยโครงสร้างต้นไม้นี้ไปประยุกต์ในการใช้งาน

Neamtiu, I., Foster, J.S., และ Hicks, M.[19] ได้ร่วมกันพัฒนาระบบการเปรียบเทียบรหัสต้นฉบับของโปรแกรม เพื่อทำให้เห็นถึงการเปลี่ยนแปลงและพัฒนาที่เกิดขึ้นในรหัสต้นฉบับโดยอาศัยการเปรียบเทียบจากโครงสร้างต้นไม้ไวยากรณ์ โดยในการทำงานของโปรแกรมนั้นโปรแกรมจะทำการตรวจสอบความสัมพันธ์แบบหนึ่งทั่วถึงของปมแต่ละปมในโครงสร้างต้นไม้ไวยากรณ์เชิงนามธรรมโดยเริ่มจากการหาความสัมพันธ์ระหว่างชื่อของฟังก์ชันในรหัสของโปรแกรมรุ่นเก่าและรุ่นใหม่ ระบบนี้ถูกพัฒนาขึ้นโดยมีจุดประสงค์เพื่อทำความเข้าใจในการพัฒนาและเปลี่ยนแปลงไปของโปรแกรมในแต่ละรุ่น เพื่อนำความเข้าใจที่ได้ไปใช้ในการดูแลและพัฒนาโปรแกรมต่อๆไป

บทที่ 3

การวิเคราะห์กรอบการทำงานและออกแบบระบบ

ในงานวิจัยนี้ผู้วิจัยได้เลือกใช้ภาษายูเอ็มแอล(UML: Unified Model Language) เป็นภาษาและเครื่องมือที่ใช้แสดงการวิเคราะห์และการทำงานในส่วนต่างๆของระบบ โดยอาศัยแนวความคิดและ ทฤษฎีที่เกี่ยวข้องจากบทที่ 2 เป็นแหล่งข้อมูลอ้างอิง

3.1 การวิเคราะห์ระบบ

ในงานวิจัยนี้ผู้วิจัยได้เลือกใช้แผนภาพยูสเคส(Use Case Diagram) เป็นเครื่องมือสำหรับการแสดงกรอบการทำงานและความต้องการของระบบเป็นแผนภาพ โดยได้แบ่งแผนภาพเป็นสองระดับและใช้ตารางรายละเอียดยูสเคส (Use Case Description) เพื่ออธิบายในรายละเอียดของยูสเคสที่มีความสำคัญโดยมีรายละเอียดดังนี้คือ

1) แผนภาพยูสเคสระดับที่ 1 เป็นแผนภาพหลักที่ใช้แสดงความสัมพันธ์โดยรวมในการทำงาน โดยจะแสดงถึงแอคเตอร์ (Actor) และความสัมพันธ์ของแต่ละแอคเตอร์ในระบบ

2) แผนภาพยูสเคสระดับที่ 2 เป็นแผนภาพที่อธิบายถึงรายละเอียดย่อยของยูสเคสที่ประกอบอยู่ในแผนภาพระดับที่1 เพื่อแสดงให้เห็นถึงรายละเอียดหน้าที่ความรับผิดชอบและขอบเขตการทำงานของแต่ละแอคเตอร์ โดยแบ่งการแสดงแผนภาพในระดับนี้ออกตามแอคเตอร์

3.1.1 แผนภาพยูสเคสระดับที่ 1

เป็นแผนภาพที่แสดงให้เห็นถึงความต้องการของระบบโดยรวม ซึ่งประกอบไปด้วยแอคเตอร์ 3 แอคเตอร์ โดยที่แต่ละแอคเตอร์จะมีหน้าที่และการทำงานที่แตกต่างกันไปรายละเอียดดังตารางที่ 3.1 และประกอบไปด้วยยูสเคส 4 ยูสเคสรายละเอียดดังรูปที่ 3.1

ตารางที่ 3.1 บทบาทและหน้าที่โดยรวมของแอคเตอร์ในระบบ

แอคเตอร์	รายละเอียดหน้าที่
นักพัฒนา	ดูแลและพัฒนาการทำงานให้การเติมข้อบกพร่องลงในรหัสต้นฉบับของโปรแกรมมีความหลากหลาย เป็นไปตามจุดประสงค์การเรียนรู้ และระบบสามารถทำงานได้อย่างมีประสิทธิภาพ

ตารางที่ 3.1 บทบาทและหน้าที่โดยรวมของแอคเตอร์ในระบบ(ต่อ)

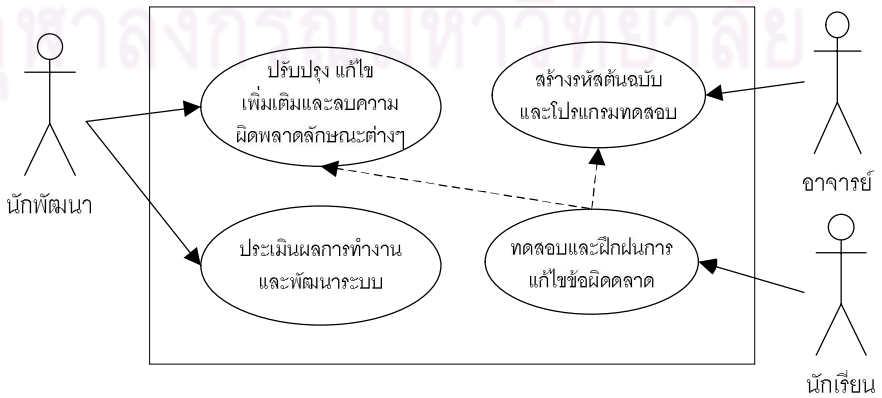
แอคเตอร์	รายละเอียดหน้าที่
อาจารย์ผู้สอน	สร้างรหัสต้นฉบับของโปรแกรมที่มีการทำงานที่ถูกต้องและสร้างโปรแกรมที่จะนำไปใช้เพื่อตรวจสอบการทำงานของรหัสต้นฉบับ เพื่อประกอบกันเป็นคลังข้อมูลไว้ใช้ในการฝึกฝน
นักเรียน	เป็นผู้ใช้งานระบบในการฝึกฝนการแก้ไขรหัสต้นฉบับที่มีการทำงานที่ผิดพลาดเนื่องจากข้อผิดพลาดที่ถูกเติมลงไป

1) ยูสเคสการประเมินผลและการพัฒนาการทำงานของโปรแกรม เป็นยูสเคสสำหรับการจัดการและการพัฒนาการทำงานต่างๆของระบบเพื่อให้ระบบสามารถทำงานได้อย่างเป็นปรกติและมีประสิทธิภาพ โดยมีนักพัฒนาเป็นแอคเตอร์ที่เกี่ยวข้อง

2) ยูสเคสเพิ่มเติมและปรับปรุงลักษณะความผิดพลาด เป็นยูสเคสสำหรับจัดการลักษณะความผิดพลาดต่างๆซึ่งบางครั้งอาจจะต้องทำการเพิ่มเติมลักษณะความผิดพลาดพิเศษที่ต้องการให้นักเรียนทำการฝึกฝน หรือเพื่อจัดกลุ่มของลักษณะข้อบกพร่องในแบบต่างๆเข้าด้วยกัน เพื่อให้สอดคล้องกับการเรียนการสอน โดยมีนักพัฒนาเป็นแอคเตอร์ที่เกี่ยวข้อง

3) ยูสเคสการสร้างแบบฝึกหัดต้นฉบับ เป็นยูสเคสสำหรับการสร้างรหัสต้นฉบับไว้เป็นฐานข้อมูลของโจทย์ปัญหาของระบบ เพื่อให้ระบบนำไปใช้สร้างเป็นแบบฝึกหัดตามจุดประสงค์ของการเรียนการสอน โดยมีอาจารย์ผู้สอนเป็นแอคเตอร์ที่เกี่ยวข้อง

4) ยูสเคสการใช้ระบบในการทดสอบหรือฝึกฝนแก้ไขโปรแกรมที่มีการทำงานที่ผิดพลาด เป็นยูสเคสการใช้งานหลักที่มีนักเรียนเป็นแอคเตอร์ที่เกี่ยวข้อง



รูปที่ 3.1 แผนภาพยูสเคสหลักของระบบ

3.1.2 แผนภาพยูสเคสระดับที่ 2

3.1.2.1 ยูสเคสในการพัฒนาระบบ ปรับปรุงแก้ไขและเพิ่มเติมลักษณะความผิดพลาด เป็นยูสเคสที่แสดงถึงหน้าที่การทำงานของนักพัฒนา เพื่อให้เกิดความหลากหลายในลักษณะของความผิดพลาดที่จะนำไปใช้ในการเติมลงในรหัสต้นฉบับ โดยจะประกอบด้วยยูสเคสย่อยๆดังนี้

1) ยูสเคสการเพิ่มลักษณะความผิดพลาด เป็นการเพิ่มลักษณะความผิดพลาดใหม่ที่มีความสำคัญหรืออาจเป็นความผิดพลาดที่มักเกิดขึ้นอย่างบ่อยครั้งโดยลักษณะความจะมีทั้งความผิดพลาดเชิงไวยากรณ์(Syntactic error)และความผิดพลาดเชิงความหมาย(Semantic error) เพื่อให้ระบบนำไปใช้ในการกลายรหัสต้นฉบับในขั้นตอนการสร้างแบบฝึกหัดหรือแบบทดสอบต่างๆ โดยนักพัฒนาจะเป็นผู้ทำการวิเคราะห์รูปแบบของความผิดพลาด เขียนโปรแกรมเพื่อค้นหาลักษณะรูปแบบต่างๆในโครงสร้างต้นไม้ไวยากรณ์สำหรับการทำการกลายรหัสดังตารางที่ 3.2

2) ยูสเคสการแก้ไขลักษณะความผิดพลาด เป็นยูสเคสสำหรับการแก้ไขลักษณะความผิดพลาดที่ได้ดำเนินการสร้างมาก่อนหน้านี้ให้มีความเหมาะสมมากขึ้น โดยนักพัฒนาจะเป็นผู้ที่มีหน้าที่ในการวิเคราะห์และปรับรูปแบบของความผิดพลาดต่างๆตามแต่ลักษณะ

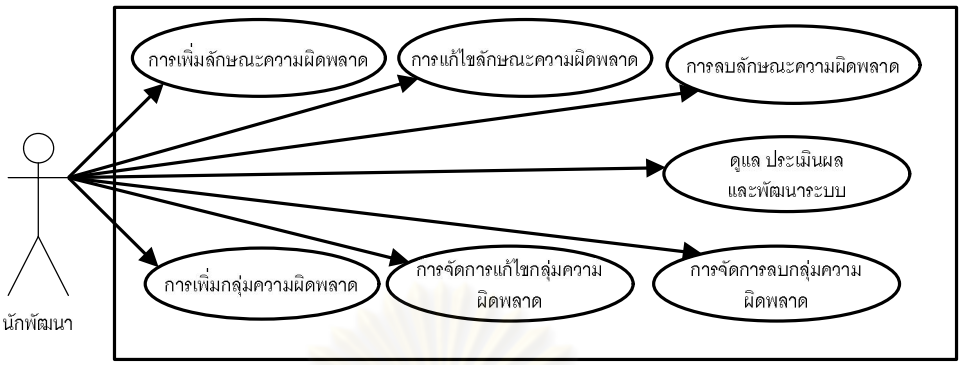
3) ยูสเคสการลบลักษณะความผิดพลาด เป็นยูสเคสสำหรับเข้าทำการแก้ไขเพื่อลบลักษะของข้อบกพร่องที่ไม่ต้องการ หรือลักษณะของข้อบกพร่องที่มีการทำงานที่ผิดพลาดออกซึ่งเป็นหน้าที่ของนักพัฒนาที่เมื่อวิเคราะห์พบการทำงานที่ผิดพลาดหรือลักษณะข้อผิดพลาดที่ไม่จำเป็นอาจเนื่องจากมีลักษณะข้อบกพร่องอื่นที่ทำงานทดแทนได้ ก็สามารถทำการลบลักษณะข้อบกพร่องที่ไม่จำเป็นนั้นออกได้

4) ยูสเคสการเพิ่มกลุ่มความผิดพลาด เป็นยูสเคสสำหรับเพิ่มกลุ่มลักษณะความผิดพลาดโดยจะเป็นการนำลักษณะความผิดพลาดแบบต่างๆที่มักเกิดขึ้นร่วมกันมารวมกันไว้เป็นกลุ่มเพื่อความสะดวกในการนำไปใช้งาน ซึ่งนักพัฒนาจะเป็นผู้รวบรวมและวิเคราะห์ข้อมูลของกลุ่มความผิดพลาดแบบต่างๆ สำหรับนำมาสร้างเป็นกลุ่มของความผิดพลาด ดังตารางที่ 3.3

5) ยูสเคสการแก้ไขจัดการกลุ่มความผิดพลาด เป็นยูสเคสสำหรับแก้ไขกลุ่มของความผิดพลาดให้เกิดความเหมาะสมตามลักษณะความผิดพลาดที่เกิดขึ้นจริงซึ่งอาจเปลี่ยนแปลงไปจากที่วิเคราะห์ไว้ได้

6) ยูสเคสการลบกลุ่มความผิดพลาด เป็นยูสเคสสำหรับการแก้ไขเพื่อลบกลุ่มของความผิดพลาดบางอย่างออก เนื่องจากมีการจัดกลุ่มใหม่ที่มีความเหมาะสมกว่า

7) ยูสเคสการพัฒนาและประเมินผลการเติมข้อผิดพลาด เป็นยูสเคสเพื่อให้ระบบสามารถทำงานได้อย่างมีประสิทธิภาพมากที่สุด



รูปที่ 3.2 ยูสเคสแสดงหน้าที่การทำงานของนักพัฒนา

ตารางที่ 3.2 รายละเอียดยูสเคสการเพิ่มลักษณะความผิดพลาด

Use Case No:	1	
Use Case Name:	ยูสเคสการเพิ่มลักษณะความผิดพลาด	
Brief Description:	เป็นยูสเคสสำหรับการเพิ่มเติมลักษณะข้อบกพร่องใหม่ให้กับระบบ	
Primary Actor:	นักพัฒนา	
Relationships:	Association: - Include: - Extend: - Generalization: -	
Pre-condition:	นักพัฒนาจะต้องวิเคราะห์รูปแบบลักษณะของข้อบกพร่องที่ต้องการเติม	
Post-Condition:	-	
Normal Flows:	Step	Action
	1	เขียนโปรแกรมที่เป็นคลาสลูกของ ASTVisitor เพื่อใช้ในการหารูปแบบที่จะแก้ไขให้เกิดข้อผิดพลาด
	2	เขียนโปรแกรมที่เป็นคลาสลูกของ ASTModifier เพื่อเข้าไปแก้ไขรหัสต้นฉบับ
	3	เพิ่มเติมชื่อของลักษณะข้อบกพร่องลงใน ModType

ตารางที่ 3.3 รายละเอียดยูสเคสการเพิ่มกลุ่มลักษณะความผิดพลาด

Use Case No:	4	
Use Case Name:	ยูสเคสการเพิ่มกลุ่มความผิดพลาด	
Brief Description:	เป็นยูสเคสสำหรับการเพิ่มเติมกลุ่มของข้อบกพร่องใหม่ให้กับระบบ	
Primary Actor:	นักพัฒนา	
Relationships:	Association: - Include: - Extend: - Generalization: -	
Pre-condition:	นักพัฒนาวิเคราะห์ลักษณะความผิดพลาดต่างๆเพื่อรวมรวมเป็นกลุ่มความผิดพลาดเพื่อความสะดวกในการใช้งาน	
Post-Condition:	-	
Normal Flows:	Step	Action
	1	เพิ่มชื่อกลุ่มลงใน ModType
	2	เพิ่มรายชื่อของลักษณะความผิดพลาดแต่ละลักษณะที่ต้องการรวมรวมไว้ด้วยในลงใน ModType

3.1.2.2 ยูสเคสในการสร้างคลังข้อมูลของโปรแกรมต้นฉบับ เป็นยูสเคสของอาจารย์ผู้สอนที่มีหน้าที่ในการออกแบบรหัสต้นฉบับของโปรแกรม โปรแกรมตรวจสอบการทำงานของรหัสต้นฉบับและส่วนอื่นๆ ที่จะนำมาประกอบกันในการเติมข้อบกพร่องลักษณะต่างๆของระบบเพื่อใช้เป็นแบบฝึกหัดสำหรับการฝึกฝนของนักเรียน โดยประกอบด้วยยูสเคสย่อยๆดังนี้

1) การสร้างรหัสต้นฉบับของโปรแกรม เพื่อให้เป็นไปตามจุดประสงค์ของการเรียนการสอน อาจารย์ผู้สอนจะเป็นผู้สร้างรหัสต้นฉบับของโปรแกรมที่จะนำมาเติมข้อผิดพลาด ซึ่งรหัสต้นฉบับของโปรแกรมที่ถูกสร้างขึ้นนั้นจะเปรียบเสมือนฐานข้อมูลของระบบเพราะรหัสต้นฉบับของโปรแกรมนั้นจะถูกนำไปเติมความผิดพลาดหรือกลุ่มของความผิดพลาดในลักษณะต่างๆที่แตกต่างกันออกไปตามแต่ข้อกำหนดความผิดพลาดที่ถูกกำหนดขึ้นเพื่อให้ได้แบบฝึกหัดที่

หลากหลาย โดยมีรายละเอียดดังตารางที่ 3.4 ซึ่งในการสร้างรหัสต้นฉบับของโปรแกรมนี้จะประกอบด้วยยูสเคสย่อยๆอีกสองยูสเคสคือ

1.1) การเขียนหมายเหตุประกอบการทำงาน โดยหมายเหตุประกอบการทำงานนี้จะเป็นส่วนที่ใช้ในการอธิบายให้นักเรียนทราบถึงหลักการทำงานของโปรแกรมอย่างคร่าวๆ ว่าโปรแกรมนี้คือโปรแกรมอะไร มีข้อมูลเข้าคืออะไร ข้อมูลออกคืออะไร เพื่อใช้เป็นข้อมูลในการค้นหาความผิดพลาดที่ถูกเติมลงในโปรแกรมเพื่อให้โปรแกรมนั้นสามารถกลับมาทำงานได้อย่างถูกต้อง ซึ่งมีรายละเอียดดังตารางที่ 3.5

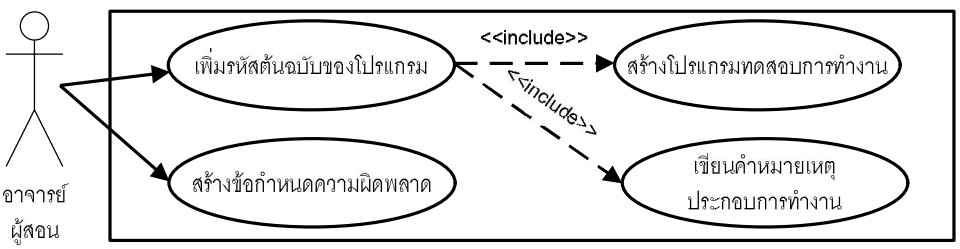
1.2) การสร้างโปรแกรมสำหรับการทดสอบการทำงาน เป็นโปรแกรมที่จะถูกนำมาใช้ในการทดสอบการทำงานของโปรแกรมภายหลังผ่านขั้นตอนการเติมความผิดพลาดและใช้ในการตรวจสอบรหัสต้นฉบับหลังการฝึกการแก้ไขข้อบกพร่องของนักเรียนว่าสามารถแก้ไขได้อย่างถูกต้องหรือไม่

2) การสร้างข้อกำหนดความผิดพลาด เป็นการกำหนดขอบเขตของความผิดพลาดที่จะทำการเติมลงในรหัสต้นฉบับเพื่อให้เป็นไปตามจุดประสงค์ของการเรียนการสอนในแต่ละช่วง โดยในการกำหนดข้อกำหนดความผิดพลาดนี้จะมีสิ่งที่ต้องกำหนดสองอย่างคือ

2.1) กำหนดลักษณะความผิดพลาดที่ต้องการ โดยจะสามารถระบุถึงลักษณะความผิดพลาดที่ต้องการเติมลงในรหัสต้นฉบับเป็นความผิดพลาดแต่ละลักษณะหรือกลุ่มของความผิดพลาดก็ได้ ถ้าการระบุลักษณะความผิดพลาดเป็นกลุ่มความผิดพลาด ระบบจะทำการสุ่มลักษณะความผิดพลาดจากกลุ่มลักษณะความผิดพลาดที่ได้กำหนดไว้ แล้วทำการเติมลงในรหัสต้นฉบับให้โดยอัตโนมัติ

2.2) ขอบเขตของรหัสต้นฉบับที่จะทำการเติม โดยขอบเขตของรหัสต้นฉบับที่จะระบุนี้สามารถระบุได้ในสองลักษณะคือ คลาส หรือ เมท็อดของคลาส ที่ต้องการจะเติมความผิดพลาดลงไป

โดยข้อกำหนดความผิดพลาดนี้จะถูกระบุโดยเป็นโครงสร้าง XML เพื่อให้สามารถเข้าถึงและดำเนินการตามข้อกำหนดได้อย่างถูกต้อง ซึ่งมีรายละเอียดดังตารางที่ 3.6



รูปที่ 3.3 ยูสเคสแสดงหน้าที่การทำงานของอาจารย์ผู้สอน

ตารางที่ 3.4 รายละเอียดยูสเคสการเพิ่มรหัสต้นฉบับของโปรแกรม

Use Case No:	1	
Use Case Name:	การเพิ่มรหัสต้นฉบับของโปรแกรม	
Brief Description:	เป็นยูสเคสการเพิ่มรหัสต้นฉบับของโปรแกรมเพื่อเป็นรหัสตั้งต้นก่อนนำไปแก้ไขเพื่อใช้เป็นแบบฝึกหัด	
Primary Actor:	อาจารย์ผู้สอน	
Relationships:	Association: - Include: ยูสเคสการสร้างโปรแกรมทดสอบการทำงาน, ยูสเคสการเขียนคำหมายเหตุประกอบการทำงาน Extend: - Generalization: -	
Pre-condition:	จุดประสงค์การเรียนการสอน	
Post-Condition:	-	
Normal Flows:	Step	Action
	1	ผู้สอนวิเคราะห์จุดประสงค์การเรียนการสอน
	2	เลือกตัวอย่างรหัสของโปรแกรมให้มีโครงสร้างการทำงานตามจุดประสงค์การเรียนการสอน
	3	เขียนเป็นรหัสต้นฉบับของโปรแกรมที่จะนำไปใช้งาน

ตารางที่ 3.5 รายละเอียดยูสเคสการสร้างไฟล์ข้อกำหนดความผิดพลาด

Use Case No:	2.1
Use Case Name:	การสร้างโปรแกรมทดสอบการทำงาน
Brief Description:	เป็นยูสเคสการสร้างโปรแกรมสำหรับตรวจสอบการทำงานของโปรแกรมว่ามีการทำงานที่ถูกต้องตามที่ต้องการหรือไม่
Primary Actor:	อาจารย์ผู้สอน

ตารางที่ 3.5 รายละเอียดยูสเคสการสร้างไฟล์ข้อกำหนดความผิดพลาด (ต่อ)

<table>
<tr><td>Relationships:</td><td colspan="2">Association: -
Include: -
Extend: -
Generalization: -</td></tr>
<tr><td>Pre-condition:</td><td colspan="2">โปรแกรมต้นฉบับ</td></tr>
<tr><td>Post-Condition:</td><td colspan="2"></td></tr>
<tr><td rowspan="4">Normal Flows:</td><td>Step</td><td>Action</td></tr>
<tr><td>1</td><td>ทำความเข้าใจการการงานของโปรแกรมต้นฉบับ</td></tr>
<tr><td>2</td><td>สร้างกรณีทดสอบการทำงานของโปรแกรม</td></tr>
<tr><td>3</td><td>เขียนรวมเป็นโปรแกรมทดสอบการทำงานที่มีการรายค่าที่ได้จากการตรวจสอบระหว่าง 0-1 โดย 1 หมายถึงตรวจไม่พบการทำงานที่ผิดพลาดของโปรแกรม</td></tr>
</table>

ตารางที่ 3.6 รายละเอียดยูสเคสการสร้างข้อกำหนดความผิดพลาด

Use Case No:	2
Use Case Name:	สร้างข้อกำหนดความผิดพลาด
Brief Description:	เป็นยูสเคสการสร้างข้อกำหนดความผิดพลาดเพื่อกำหนดเงื่อนไขสำหรับการเติมข้อบกพร่องประเภทต่างๆลงในรหัสต้นฉบับ
Primary Actor:	อาจารย์ผู้สอน
Relationships:	Association: - Include: Extend: - Generalization: -
Pre-condition:	จุดประสงค์การเรียนการสอน
Post-Condition:	-

ตารางที่ 3.6 รายละเอียดยูสเคสการสร้างข้อกำหนดความผิดพลาด (ต่อ)

Normal Flows:	Step	Action
	1	ผู้สอนวิเคราะห์จุดประสงค์การเรียนการสอน
	2	เลือกขอบเขตในรหัสต้นฉบับของโปรแกรมเพื่อทำการเติมข้อบกพร่อง
	3	ระบุชนิดและจำนวนของข้อบกพร่องที่ต้องการลงในไฟล์ข้อกำหนดความผิดพลาดโดยมีโครงสร้างเป็น XML

3.1.2.3 ยูสเคสในการใช้งานของนักเรียน เป็นยูสเคสในการใช้งานสำหรับการฝึกฝนทักษะในการแก้ไขข้อบกพร่องที่ถูกเติมลงในรหัสต้นฉบับภายใต้โปรแกรมต่างๆ โดยในการใช้งานนี้จะมียูสเคสย่อยๆ สองยูสเคสคือ

1) ยูสเคสการทำแบบฝึกหัด เป็นยูสเคสหลักสำหรับการใช้งาน นักเรียนจะทำการเรียกโปรแกรมเพื่อให้โปรแกรมทำการเติมข้อบกพร่องต่างๆลงในรหัสต้นฉบับของโปรแกรมตามข้อกำหนดที่อาจารย์ผู้สอนได้กำหนดไว้ล่วงหน้าแล้วอย่างคร่าวๆ เมื่อได้แบบฝึกหัดมาแล้วนักเรียนก็จะทำการอ่านคำอธิบายหมายเหตุประกอบการทำงานเพื่อทำให้ทราบว่าเป็นโปรแกรมอะไรมีการทำงานโดยภาพรวมอย่างไรแล้วจึงเข้าไปทำการค้นหาและแก้ไขข้อบกพร่องต่างๆที่ถูกซ่อนอยู่ภายในโปรแกรม

2) ยูสเคสการตรวจแบบฝึกหัดหลังการแก้ไข ภายหลังการแก้ไขข้อบกพร่องต่างๆนักเรียนจะทำการตรวจสอบการทำงานของโปรแกรมว่ามีความถูกต้องครบถ้วนแล้วหรือไม่จากการเรียกโปรแกรมตรวจสอบการทำงานขึ้น เมื่อโปรแกรมทำการตรวจสอบแล้วพบว่ามีการทำงานได้อย่างถูกต้องครบถ้วนเหมือนก่อนถูกเติมความผิดพลาดลงไปก็จะรายงานว่าความผิดพลาดที่เติมลงไปได้รับการแก้ไขหมดแล้ว

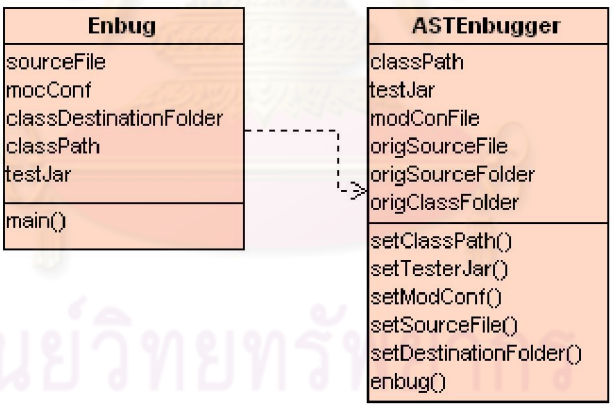
รูปที่ 3.4ยูสเคสแสดงหน้าที่การทำงานของนักเรียน

3.2 การออกแบบระบบ

ในการออกแบบระบบผู้วิจัยได้เลือกใช้แผนภาพคลาส(class diagrams) ในการแสดงองค์ประกอบของคลาสต่างๆ ความสัมพันธ์ระหว่างคลาสที่ได้จากการออกแบบ โดยในการแสดงการออกแบบระบบนี้จะแบ่งแผนภาพคลาสออกเป็นสามส่วนคือแผนภาพคลาสที่เกี่ยวข้องกับขั้นตอนการเตรียมสภาพแวดล้อมก่อนการเติมข้อผิดพลาดและแผนภาพคลาสที่เกี่ยวข้องกับขั้นตอนการเติมความผิดพลาด และส่วนแผนภาพคลาสที่เกี่ยวข้องกับข้อบกพร่องในแต่ละชนิด ซึ่งการออกแบบการทำงานของระบบนี้ได้คำนึงถึงความสอดคล้องกับกรอบการทำงานของ ASTEclipse ซึ่งทำให้ได้ระบบที่มีรายละเอียดการทำงานดังนี้

3.2.1 แผนภาพคลาสที่เกี่ยวข้องกับการเตรียมสภาพแวดล้อม

แผนภาพคลาสของขั้นตอนการเตรียมสภาพแวดล้อมเป็นแผนภาพที่แสดงถึงคลาสต่างๆ และความสัมพันธ์ระหว่างคลาสที่เกี่ยวข้องกับการรับข้อมูล แปลความ เตรียมสภาพแวดล้อมต่างๆ ก่อนเริ่มทำการเติมความผิดพลาดลงในรหัสต้นฉบับดังรูปที่ 3.5 ซึ่งประกอบไปด้วยคลาสต่างๆ ดังนี้



รูปที่ 3.5 แผนภาพคลาสของการเตรียมสภาพแวดล้อมในการทำงาน

1) คลาส Enbug เป็นคลาสที่ใช้ในการเริ่มต้นการทำงาน โดยคลาสนี้จะใช้ในการรับและแปลผลค่าพารามิเตอร์ต่างๆเพื่อการเตรียมพร้อมสำหรับการทำงาน โดยการทำงานของคลาสนี้จะรับค่าพารามิเตอร์ต่างๆทาง คุณสมบัติ(properties) โดยมีรายละเอียดดังตารางที่ 3.7

ตารางที่ 3.7 ค่าคุณสมบัติต่างๆที่คลาส Enbug รับเข้าเพื่อใช้ในการทำงาน

คุณสมบัติ	ความหมายและหน้าที่
sourceFile	ตำแหน่งของไฟล์รหัสต้นฉบับของโปรแกรมที่จะใช้ในการเติมข้อบกพร่อง

ตารางที่ 3.7 ค่าคุณสมบัติต่างๆที่คลาส Enbug รับเข้าเพื่อใช้ในการทำงาน (ต่อ)

คุณสมบัติ	ความหมายและหน้าที่
modConf	ตำแหน่งของไฟล์ข้อกำหนดความผิดพลาดที่จะใช้ประกอบการเติมข้อบกพร่อง
classDestinationFolder	ตำแหน่งที่จะใช้ในการวางไฟล์โปรแกรมที่ได้จากการแปลโปรแกรมเพื่อให้สามารถทำงานได้
classPath	ตำแหน่งของคลาสองค์ประกอบต่างๆที่จะใช้ประกอบการทำงานของโปรแกรม
testerJar	ตำแหน่งของไฟล์โปรแกรมที่ใช้ในการทดสอบการทำงานของโปรแกรมต้นฉบับที่จะนำมาเติมข้อบกพร่อง

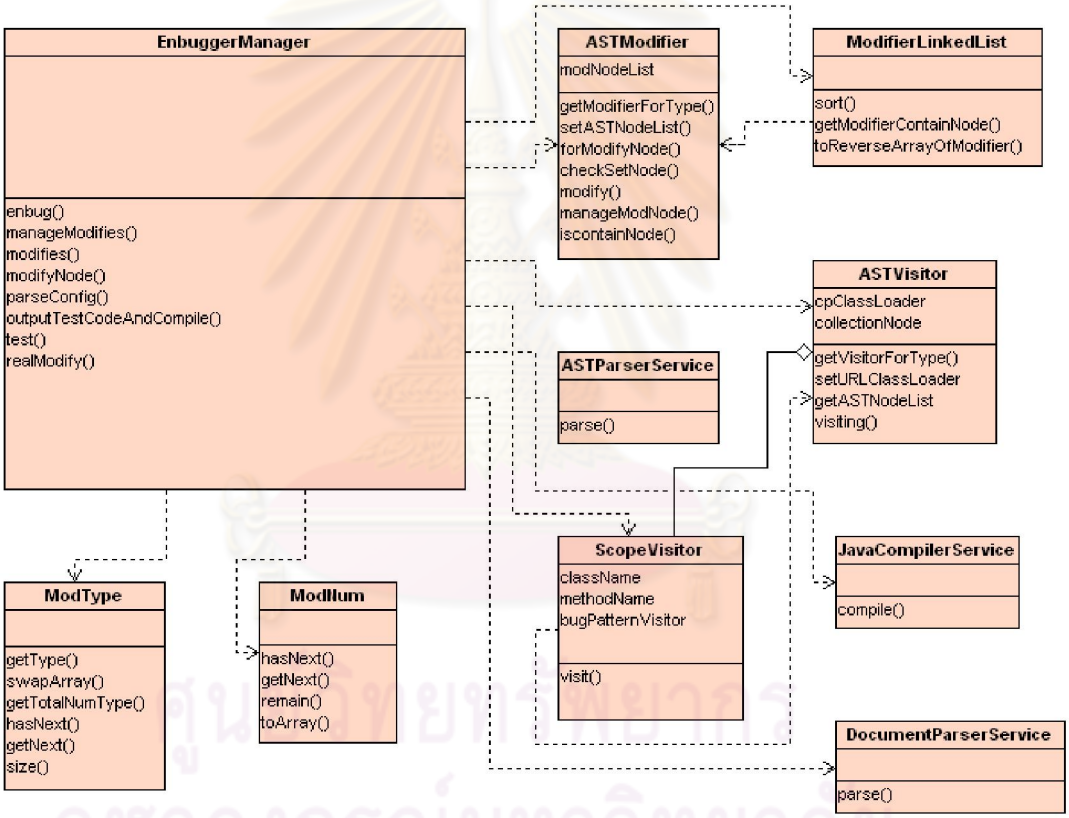
2) คลาส ASTEnbugger เป็นคลาสหลักในการทำงานของระบบโดยคลาสนี้ใช้สำหรับเตรียมสภาพแวดล้อมและลักษณะประจำ(attribute)ต่างๆเพื่อทำหน้าที่เก็บข้อมูลที่จำเป็นในการทำงาน โดยประกอบด้วยเมท็อดในการกำหนดค่าต่างๆดังตารางที่ 3.8 เมื่อได้รับค่าของข้อมูลที่จำเป็นในการเติมความผิดพลาดเข้ามาจะทำการประมวลผลและเตรียมสภาวะแวดล้อมขั้นต้นเช่น การสำรองข้อมูลของรหัสต้นฉบับของโปรแกรมที่จะทำการแปลง ตรวจสอบสิทธิในการใช้พื้นที่เพื่อการแปรโปรแกรม และอื่นๆ

ตารางที่ 3.8 เมท็อดของคลาสASTEnbugger

เมท็อด	หน้าที่
setClassPath()	ใช้ในการกำหนดค่าตำแหน่งของคลาสองค์ประกอบต่างๆที่จะใช้ประกอบการทำงานของโปรแกรมรหัสต้นฉบับ
setTesterJar()	ใช้ในการกำหนดค่าตำแหน่งของไฟล์โปรแกรมที่ใช้ในการทดสอบการทำงานของโปรแกรมต้นฉบับที่จะนำมาเติมข้อบกพร่อง
setModConf()	ใช้ในการกำหนดค่าตำแหน่งของไฟล์ข้อกำหนดความผิดพลาดที่จะใช้ประกอบการเติมข้อบกพร่อง
setSourceFile()	ใช้ในการกำหนดค่าเป็นตำแหน่งของไฟล์รหัสต้นฉบับของโปรแกรมที่จะใช้ในการเติมข้อบกพร่อง
setDestinationFolder()	ใช้ในการกำหนดค่าตำแหน่งที่จะใช้ในการวางไฟล์โปรแกรมที่ได้จากการคอมไพลล์แล้วสามารถทำงานได้

3.2.2 แผนภาพคลาสที่เกี่ยวข้องกับขั้นตอนการเติมความผิดพลาด

แผนภาพคลาสที่เกี่ยวข้องกับขั้นตอนการเติมความผิดพลาดเป็นแผนภาพของคลาสหลักในการทำงาน ที่จะทำหน้าที่ในการค้นหารูปแบบที่เหมาะสมหรือรูปแบบที่เป็นไปได้ในการเติมความผิดพลาดลงไป ในรหัสต้นฉบับที่จะทำการเติมความผิดพลาด สุ่มตำแหน่งที่จะเติมความผิดพลาดจากรูปแบบทั้งหมดที่ค้นพบและเติมข้อบกพร่องลงในรหัสต้นฉบับ ทดสอบและรวบรวมลักษณะความผิดพลาดให้เป็นรหัสที่มีการทำงานที่ผิดพลาด ซึ่งในการทำงานจะประกอบไปด้วยคลาสต่างๆดังรูปที่ 3.6 ซึ่งมีรายละเอียดดังนี้



รูปที่ 3.6 แผนภาพคลาสที่เกี่ยวข้องกับขั้นตอนการเติมความผิดพลาด

1) EnbuggerManager เป็นคลาสหลักที่มีหน้าที่ในการจัดการเติมความผิดพลาดต่างๆ ตั้งแต่เรียกการแปลงรหัสต้นฉบับแล้วทำการเปลี่ยนเป็นโครงสร้างต้นไม้ไวยากรณ์เชิงนามธรรม จัดการเรื่องการสุ่มลักษณะของความผิดพลาดจากกลุ่มความผิดพลาด ควบคุมการหารูปแบบเหมาะสมในการเติมความผิดพลาดของคลาสในกลุ่มคลาส Visitor สุ่มตำแหน่งของโครงสร้างต้นไม้ไวยากรณ์ที่จะทำการเติมความผิดพลาดจากแบบแผนที่ค้นหาได้ ควบคุมการเติมความ

ผิดพลาด ทดสอบและรวบรวมลักษณะต่างๆที่ผ่านการทดสอบการทำงานแล้ว โดยประกอบด้วยเมท็อดต่างๆดังตารางที่ 3.9

ตารางที่ 3.9 เมท็อดของคลาส EnbuggerManager

เมท็อด	หน้าที่
parseConfig()	ใช้ในการสั่งดำเนินการแปลความข้อกำหนดความผิดพลาดจากโครงสร้างไฟล์ XML สำหรับนำไปดำเนินการตามข้อกำหนด
manageModifies()	ใช้ในการจัดการสร้างวัตถุของคลาสที่เป็นคลาสลูกของคลาส ASTVisitor และ ASTModifier ที่จะใช้ในการเติมข้อผิดพลาดลักษณะต่างๆ และมีหน้าที่ควบคุมจำนวนครั้งของข้อผิดพลาด ในกรณีที่ต้องการเติมข้อผิดพลาดลงในรหัสมากว่าหนึ่งตำแหน่ง
modifies()	เป็นเมท็อดสำหรับการดำเนินการค้นหาแบบแผนที่เหมาะสมสำหรับการเติมความผิดพลาดทั้งหมดในช่วงของรหัสต้นฉบับที่ได้กำหนดไว้แล้วเก็บข้อมูลที่ได้ไว้ในรูปแบบของลำดับรายการ
modifyNode()	เป็นเมท็อดสำหรับการทำงานในขั้นตอนการแก้ไขโครงสร้างต้นไม้ไวยากรณ์เพื่อให้เกิดความผิดพลาดในรหัสต้นฉบับ โดยจะมีการจดจำลักษณะที่แก้ไขไว้เพื่อไว้ใช้ในการรวบรวมลักษณะความผิดพลาดในตอนหลัง
outputTestCodeAndCompile()	เป็นเมท็อดสำหรับแปลงผลของโครงสร้างที่ถูกดำเนินการแก้ไขเรียบร้อยแล้วกลับเป็นรหัสต้นฉบับในรูปแบบอักษรเหมือนเดิมและส่งไปยังเนื้อที่ชั่วคราวที่ได้สร้างขึ้นแล้วทำการแปลโปรแกรมเพื่อเตรียมพร้อมสำหรับการทำงานในขั้นตอนการทดสอบ
test()	เป็นเมท็อดสำหรับทำการทดสอบการทำงานรหัสต้นฉบับที่ได้ดำเนินการแก้ไขเปลี่ยนแปลงเรียบร้อยแล้วเพื่อเป็นการป้องกันการเกิดการแก้ไขในรูปแบบสมมูลที่การแก้ไขจะไม่ส่งผลกระทบใดๆต่อการทำงาน
realModify()	เป็นเมท็อดที่นำการเปลี่ยนแปลงทั้งหมดที่ได้รับการทดสอบแล้วมาดำเนินการรวบรวมและนำไปแก้ไขโครงสร้างของรหัสต้นฉบับในครั้งสุดท้ายซึ่งจะทำให้ได้โครงสร้างต้นไม้ไวยากรณ์ที่มีความผิดพลาดในลักษณะต่างๆที่ต้องการปรากฏขึ้น

2) คลาส JavaCompilerService เป็นคลาสสำหรับคอยให้การบริการในการแปลโปรแกรมเพื่อให้สามารถทำงานได้ โดยจะใช้งานด้วยการเรียกผ่านเมท็อด compile()

3) คลาส DocumentParserService เป็นคลาสสำหรับคอยให้บริการแปลรหัสต้นฉบับให้อยู่ในรูปแบบวัตถุของคลาส org.eclipse.jface.textDocuement ก่อนจะนำไปดำเนินการเปลี่ยนเป็นโครงสร้างต้นไม้ไวยากรณ์เชิงนามธรรมต่อไป โดยจะเรียกใช้งานผ่านเมท็อด parse()

4) คลาส ASTParserService เป็นคลาสสำหรับคอยให้บริการแปล วัตถุของคลาส org.eclipse.jface.textDocuement ให้อยู่ในรูปแบบของต้นไม้ไวยากรณ์เชิงนามธรรม โดยจะเรียกใช้งานผ่านเมท็อด parse()

5) คลาส ModType เป็นคลาสสำหรับใช้ในการจัดการลักษณะของกลุ่มความผิดพลาด โดยจะประกอบด้วยเมท็อดที่สำคัญๆดังตารางที่ 3.10

ตารางที่ 3.10 เมท็อดของคลาส ModType

เมท็อด	หน้าที่
getType()	เป็นเมท็อดที่มีหน้าที่ในการรวมรวบและตอบรายการของข้อบกพร่องในรูปแบบอาร์เรย์สำหรับการนำไปใช้ในการสร้างเป็นวัตถุของคลาส ASTVisitor และ ASTModifier
swapArray()	เป็นเมท็อดในการสุ่มเพื่อการเปลี่ยนแปลงลำดับของข้อบกพร่องในอาร์เรย์สำหรับกลุ่มของข้อบกพร่องที่มีลักษณะของข้อบกพร่องมากกว่าหนึ่งลักษณะ เพื่อให้การเติมข้อบกพร่องลงในรหัสต้นฉบับนั้นเป็นไปอย่างสุ่ม
hasNext()	เป็นเมท็อดสำหรับการตรวจสอบว่ายังมีลักษณะของข้อบกพร่องแบบอื่นๆที่อยู่ในเงื่อนไขข้อกำหนดที่ต้องการหรือไม่ สำหรับการทดลองเติมข้อบกพร่องครั้งถัดไป
getNext()	เป็นเมท็อดสำหรับการขอลำดับรายการของข้อบกพร่องลำดับถัดไปสำหรับการเติมข้อบกพร่อง
size()	เป็นเมท็อดสำหรับตรวจสอบว่ากลุ่มของข้อบกพร่องที่ต้องการนั้นมีลักษณะข้อบกพร่องที่อยู่ในขอบเขตกี่ลักษณะ

6) คลาส ModNum เป็นคลาสสำหรับใช้ในการจัดการจำนวนครั้งของลักษณะความผิดพลาดที่ต้องการเติมลงในรหัส โดยจะประกอบด้วยเมท็อดที่สำคัญๆดังตารางที่ 3.11

ตารางที่ 3.11 เมท็อดของคลาส ModNum

เมท็อด	หน้าที่
hasNext()	เป็นเมท็อดสำหรับการตรวจสอบว่ายังมีจำนวนข้อบกพร่องที่ต้องการเติมลงในรหัสต้นฉบับหลงเหลืออยู่อีกหรือไม่
getNext()	เป็นเมท็อดสำหรับการขอลำดับรายการจำนวนของข้อบกพร่องที่จำทำการเติมในลักษณะถัดไป
remain()	เป็นเมท็อดที่ใช้ในการคืนค่าจำนวนของข้อบกพร่องที่ไม่สามารถเติมลงในรหัสต้นฉบับได้ เพื่อนำไปรวมกับจำนวนข้อผิดพลาดที่เหลือและในไปใช้กับข้อบกพร่องครั้งถัดๆไป

7) คลาส ModifierLinkedList เป็นคลาสที่ใช้ในการบันทึกลักษณะความผิดพลาดที่ได้แก้ไขลงในรหัสต้นฉบับและผ่านการทดสอบว่าเกิดความผิดพลาดขึ้นจริง และมีหน้าที่เรียงลำดับการดำเนินการเติมความผิดพลาดในขั้นตอนสุดท้ายด้วย

8) คลาส ASTVisitor เป็น คลาสแม่ของคลาสกลุ่ม Visitor สำหรับใช้ในการค้นหาโครงสร้างของต้นไม้ที่มีแบบแผนที่เหมาะสมที่จะทำการแก้ไขเพิ่มเติมข้อบกพร่องและมีเมท็อดสำคัญต่างๆที่ใช้ในการสำรวจ ซึ่งจะเป็นเพียงคลาสนามธรรม(abstract) เท่านั้นไม่สามารถนำไปใช้งานได้ในทันทีจะต้องทำการสืบทอดลักษณะและเมท็อดเสียก่อน โดยในคลาสลูกของ ASTVisitor จะสามารถเข้าควบคุมพฤติกรรมในการค้นหาโครงสร้างของคลาสแม่ได้ ทำให้สามารถค้นหาแบบแผนต่างๆตามที่ต้องการได้ โดยคลาส ASTVisitor นั้นจะประกอบด้วยเมท็อดสำคัญดังตารางที่ 3.12

ตารางที่ 3.12 เมท็อดของคลาส ASTVisitor

เมท็อด	หน้าที่
setURLClassLoader()	เป็นเมท็อดไว้สำหรับการกำหนดค่าของ classpath ให้แก่ ClassLoader เพื่อไว้ใช้ในการทำการสะท้อน(reflection) ของภาษาจาวาในการค้นหาข้อมูลเชิงวัตถุที่ถูกอ้างถึงในรหัสต้นฉบับ
getASTNodeList()	เป็นเมท็อดที่ใช้ในการเข้าถึงลำดับรายการที่มีโครงสร้างเหมาะสมในการเติมความผิดพลาดหรือแก้ไข
getVisitorForType()	เป็นเมท็อดเพื่อให้บริการในการสร้างวัตถุของคลาสลูกของ ASTVisitor จากชื่อของคลาสที่กำหนดให้

Visiting()	เป็นเมท็อดที่จะถูกเรียกใช้เมื่อต้องการเริ่มต้นการสำรวจโครงสร้างต้นไม้ไวยากรณ์ของคลาส ASTVisitor

9) คลาส ScopeVisitor เป็นคลาสลูกของคลาส ASTVisitor ที่ใช้ในการสำรวจโครงสร้างต้นไม้เพื่อค้นหาบริเวณของรหัสที่ต้องการเติมข้อบกพร่องลงไปซึ่งเป็นสิ่งที่ถูกกำหนดไว้ในไฟล์ข้อกำหนดความผิดพลาด

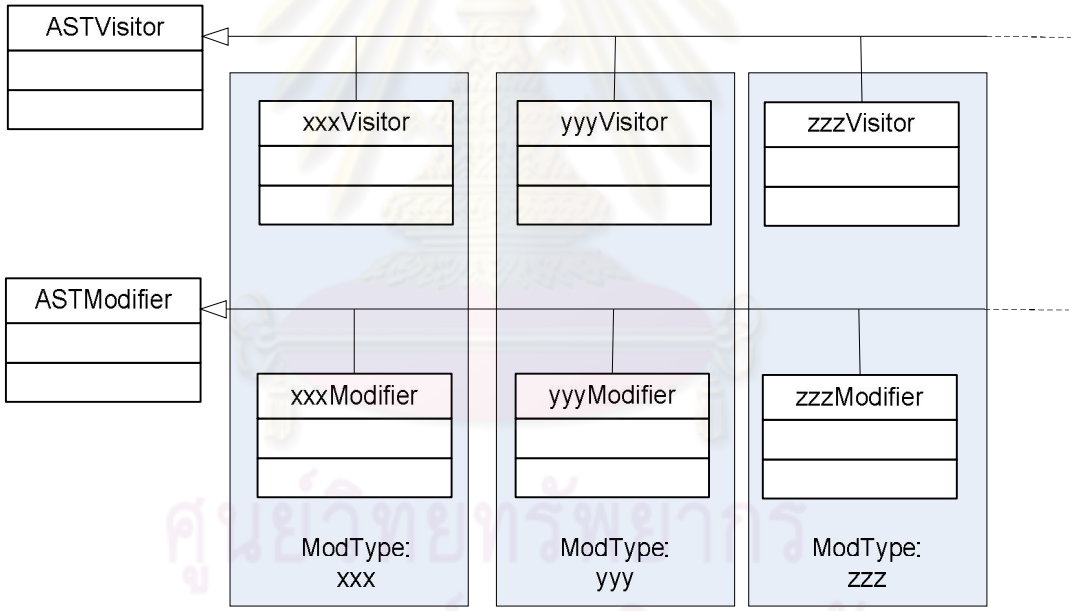
10) คลาส ASTModifier เป็นคลาสแม่ของคลาสกลุ่ม Modifier สำหรับใช้ในการแก้ไขโครงสร้างของต้นไม้ไวยากรณ์ให้เกิดข้อบกพร่องตามที่ต้องการ โดยจะมีเมท็อดสำคัญหลักที่ใช้ในการแก้ไขโครงสร้างต้นไม้ เพื่อให้คลาสลูกมาทำการสืบทอดและเข้าควบคุมพฤติกรรมในการเติมความผิดพลาดต่างๆ โดยจะประกอบด้วยเมท็อดที่สำคัญๆดังตารางที่ 3.13

ตารางที่ 3.13 เมท็อดของคลาส ASTModifier

เมท็อด	หน้าที่
getModifierForType()	เป็นเมท็อดเพื่อให้บริการในการสร้างวัตถุของคลาสลูกของ ASTModifier จากชื่อของคลาสที่กำหนดให้
setASTNodeList	เป็นเมท็อดสำหรับการกำหนดค่าปมของโครงสร้างต้นไม้ไวยากรณ์ที่ได้จำการสำรวจของ ASTvisitor สำหรับนำไปใช้ในการเติมข้อบกพร่องในรหัสต้นฉบับ
checkSetNode()	เป็นเมท็อดที่ใช้ในการตรวจสอบปมของโครงสร้างต้นไม้ไวยากรณ์ว่ามีการเปลี่ยนแปลงอื่นที่ได้เปลี่ยนแปลงปมนี้ไปแล้วหรือยังเพื่อป้องกันการแก้ไขเพิ่มเติมข้อบกพร่องซ้ำซ้อน
managerModNode()	เป็นเมท็อดสำหรับการจัดการกับปมของโครงสร้างต้นไม้ไวยากรณ์ที่ได้จากการสำรวจและจดจำของ ASTVisitor ก่อนการแก้ไขเปลี่ยนแปลง
isContainNode()	เป็นเมท็อดที่จะถูกเรียกใช้งานในขั้นตอนรวบรวมข้อบกพร่อง สำหรับใช้ในการเรียงลำดับก่อนการเติมข้อบกพร่องทั้งหมดลงในรหัสต้นฉบับ เพราะการแก้ไขโครงสร้างต้นไม้ไวยากรณ์นั้นจะต้องแก้จากปลายสุดของโครงสร้างต้นไม้เพื่อป้องกันความผิดพลาด

3.2.3 ส่วนแผนภาพของคลาสที่เกี่ยวข้องกับข้อบกพร่องในแต่ละชนิด

ความผิดพลาดแต่ละลักษณะที่จะทำการเติมนั้นจะประกอบขึ้นจากคลาสสองคลาสคือ คลาส Visitor และ Modifier เป็นกลุ่มของของคลาสที่ทำหน้าที่ในการค้นหาแบบแผนในรหัสต้นฉบับ (Visitor) และ เข้าแก้ไขเพื่อให้เกิดความผิดพลาด (Modifier) ในแต่ละลักษณะตามแต่ที่นักพัฒนาได้วิเคราะห์ไว้ โดยคลาสในกลุ่ม visitor จะมี ASTVisitor เป็นคลาสแม่ ซึ่งในการทำงานนั้น visiter จะเข้าทำการ override เมท็อด visit() โดยจะมีพารามิเตอร์เป็นชนิดของปมของโครงสร้างต้นไม้ภายใต้กรอบการทำงานของ Eclipse AST เพื่อให้ visitor สามารถเข้าไปเดินสำรวจในปมของโครงสร้างต้นไม้ตามที่ต้องการได้ และกลุ่ม Modifier จะมี ASTModifier เป็นคลาสแม่ โดยจะมีหน้าที่แก้ไขโครงสร้างต้นไม้เพื่อให้เกิดข้อผิดพลาดตามที่ออกแบบไว้ โดยมีความสัมพันธ์ดังรูปที่3.7



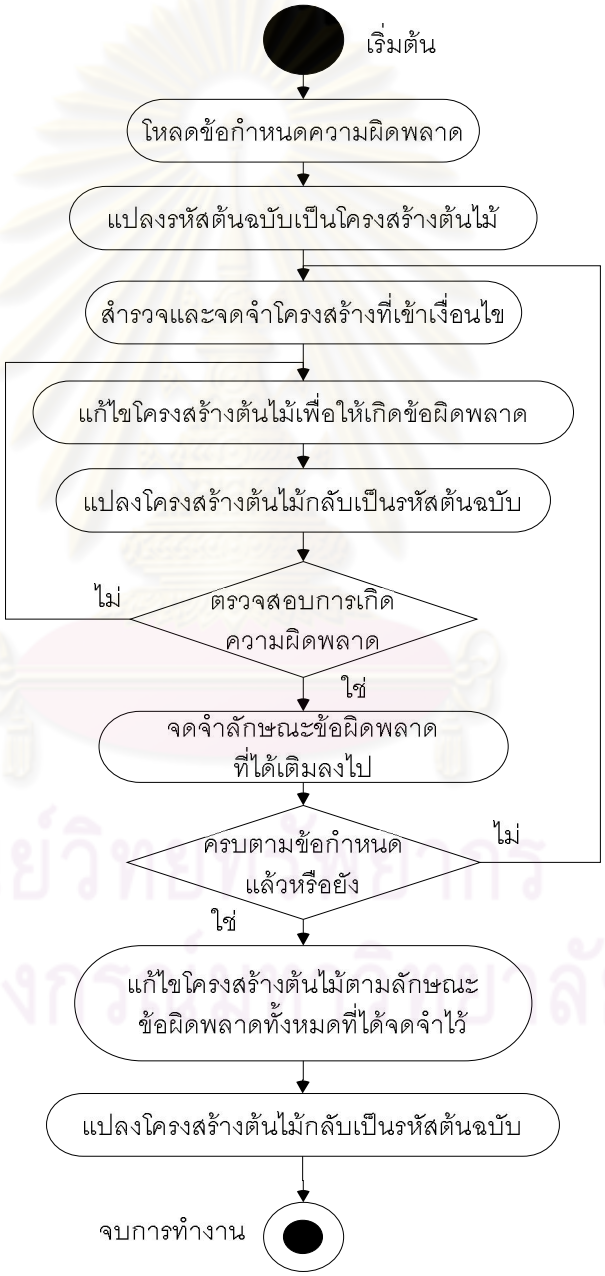
รูปที่ 3.7 แผนภาพคลาสกลุ่ม Visitor และ Modifier

3.3 ขั้นตอนการทำงานของระบบ

ในขั้นตอนการทำงานของระบบนั้นสามารถแยกพิจารณาขั้นตอนการทำงานออกเป็นสองส่วนย่อยๆคือ ขั้นตอนการทำงานโดยรวมของระบบ และการร่วมกันทำงานของ ASTVisitor และ ASTModifier

3.3.1 ขั้นตอนการทำงานโดยรวมของระบบ

จากกรอบการทำงานของ Eclipse ASTและจากการวิเคราะห์การใช้งานดังที่ได้แสดงในแผนภาพยูสเคส จะสามารถกำหนดขั้นตอนในการเติมความผิดพลาดลงไปในรหัสต้นฉบับของโปรแกรมเพื่อให้เกิดความสอดคล้องกันได้ โดยขั้นตอนในการเติมความผิดพลาดลงไปในรหัสต้นฉบับนี้จะลำดับการทำงานดังรูปที่ 3.8 และมีรายละเอียดขั้นตอนการทำงานดังนี้



รูปที่ 3.8 ภาพรวมขั้นตอนในการเติมความผิดพลาดลงไปในรหัสต้นฉบับ

1) ภายหลังจากการเตรียมสภาพแวดล้อมในการทำงานเรียบร้อยแล้วระบบจะเริ่มดำเนินการเติมข้อบกพร่องจากการสร้างวัตถุของคลาส EnbuggerManager และเรียกใช้งานเมท็อด enbug() เป็นการเริ่มต้นกระบวนการเติมข้อผิดพลาดดังรหัสตัวอย่างในรูปที่ 3.9

```
ASTEnbugger astEn = ASTEnbugger.getASTEnbugger(sourceFile,
                    classPath, classDestinationFolder,
                    testerJar, modConf);
astEn.enbug();
```

รูปที่ 3.9 เริ่มต้นกระบวนการการเติมข้อบกพร่อง

2) โหลดข้อกำหนดความผิดพลาด ในการฝึกฝนการแก้ไขข้อบกพร่องในแต่ละครั้ง อาจมีจุดประสงค์ในการเรียนการสอนที่แตกต่างกันซึ่งทำให้ลักษณะของความผิดพลาดและบริเวณที่จะทำการเติมความผิดพลาดนั้นแตกต่างกันไปในแต่ละครั้ง ซึ่งข้อมูลลักษณะและบริเวณนี้จะถูกระบุไว้ในข้อกำหนดความผิดพลาด โดยในการโหลดข้อกำหนดความผิดพลาดและการแปรผลนั้นระบบจะอาศัยการทำงานของเมท็อด parserConfig() ของคลาส EnbuggerManager ในการทำงาน โดยจะได้ผลการทำงานออกมาในรูปแบบของรายการโยง ภายในประกอบด้วยข้อมูลขอบเขตของรหัสที่ต้องการเติมข้อบกพร่อง ชนิดของกลุ่มของข้อบกพร่องที่ต้องการเติมและจำนวนที่ต้องการ หลังจากนั้นจะทำการสร้างลำดับรายการของข้อบกพร่องและจำนวนที่จะทำการเติมลงในรหัสต้นฉบับเพื่อดำเนินการเติมความผิดพลาดลงในรหัสต้นฉบับจนกว่าจะครบตามจำนวนที่ต้องการ ดังรหัสตัวอย่างในรูปที่ 3.10

```
LinkedList<Config> confList = parseConfig();
for (Config conf : confList) {
  String className = conf.getClassName(0);
  String methodName = conf.getMethodName(1);
  String modName = conf.getModName(2);
  String totalModNum = conf.getTotalModNum(3);
  ModType modTypes = new ModType(modName);
  ModNum modNums = new ModNum(modNum,
                       modTypes.getTotalNumType());
  int remain = 0;
  while (modNums.hasNext()) {
     remain = modifies(className, methodName,
                  modTypes.getNext(),modNums.getNext());
     modNums.remain(remain);
  }
}
```

รูปที่ 3.10 โหลดข้อกำหนดความผิดพลาดและเริ่มการทำงาน

3) แปลงรหัสต้นฉบับเป็นโครงสร้างต้นไม้ไวยากรณ์เชิงนามธรรม จากกรอบการทำงานของEclipse AST นั้นเราสามารถเรียกใช้งานคลาส ASTParser เพื่อให้ทำการแปลงรหัสต้นฉบับไปเป็นโครงสร้างต้นไม้ไว้สำหรับเปลี่ยนแปลงโครงสร้างการทำงาน โดยมีรายละเอียดการทำงานดังรหัสตัวอย่างในรูปที่ 3.11 ดังนี้

```
Document d = new Document(new String(workCodeDocument.get()));
CompilationUnit ast = ASTParserService.parse(d);
ast.recordModifications();
```

รูปที่ 3.11 การแปลรหัสต้นฉบับของโปรแกรมให้อยู่ในรูปแบบโครงสร้างต้นไม้

4) ค้นหาโครงสร้างที่เข้าเงื่อนไข ในการทำงานกับโครงสร้างต้นไม้ไวยากรณ์เชิงนามธรรมของ Eclipse นั้นจะสามารถค้นหารูปแบบของปมในต้นไม้ที่มีโครงสร้างตามที่ต้องการได้โดยอาศัยการสำรวจของ ASTVisitor โดยจะสำรวจลงไปตามลำดับของปมในต้นไม้เมื่อพบปมของโครงสร้างต้นไม้ที่เข้าเงื่อนไขจะให้ทำการจดจำตำแหน่งไว้สำหรับนำไปใช้ในขั้นตอนการกลายรหัส ผลที่ได้คือรายการโยงของปมในโครงสร้างต้นไม้ไวยากรณ์ที่มีรูปแบบเหมาะสมกับการเติมความผิดพลาดโดยมีรายละเอียดการทำงานดังรหัสตัวอย่างในรูปที่ 3.12 ดังนี้

```
ScopeVisitor visit = new ScopeVisitor(className,methodName,modType);
visit. setURLCLassLoader(origClassFolder, classPath);
visit.visiting(ast);
LinkedList<LinkedList<ASTNode>> list = visit.getASTNodeList();
```

รูปที่ 3.12 ค้นหาโครงสร้างที่เข้าเงื่อนไข

5) แก้ไขโครงสร้างต้นไม้ไวยากรณ์ของโปรแกรม จากตำแหน่งของโครงสร้างต้นไม้ที่มีรูปแบบที่เหมาะสมตามที่ต้องการทั้งหมดที่ได้จดจำไว้จากขั้นตอนที่ผ่านมา ระบบจะทำการสุ่มเลือกจุดที่จะทำการเติมข้อบกพร่องลงไปโดยการแก้ไขโครงสร้างต้นไม้ไวยากรณ์เพื่อให้เกิดข้อผิดพลาดโดยมีรายละเอียดการทำงานดังรหัสตัวอย่างในรูปที่ 3.13 ดังนี้

```
int order[] = randomIndex(list.size());
for (int i = 0; i < order.length && modifyCount < modNum; i++) {
      ASTModifier modifier = ASTModifier.getModifierForType(
                        modType, list.get(order[i]));

      if (modifier != null){
            modifier.modify();
      }
}
```

รูปที่ 3.13 แก้ไขโครงสร้างต้นไม้ไวยากรณ์ของโปรแกรม

6) แปลงโครงสร้างต้นไม้ไวยากรณ์ที่ถูกแก้ไขแล้วกลับสู่รหัสต้นฉบับ โดยอาศัยการทำงาน ASTParser

7) ทดสอบการเกิดความผิดพลาด เพื่อให้เกิดความมั่นใจว่าการเปลี่ยนแปลงที่เกิดขึ้นในแต่ละจุดนั้นก่อให้เกิดข้อบกพร่องขึ้นในโปรแกรม เป็นการป้องกันมิให้เกิดการแก้ไขที่สมมูลกับรหัสต้นฉบับเดิม โดยอาศัยโปรแกรมทดสอบการทำงานของโปรแกรมที่อาจารย์ผู้สอนเป็นผู้เขียนขึ้น และโปรแกรมนี้จะถูกนำไปใช้อีกครั้งในการทดสอบการทำงานของโปรแกรมภายหลังจากที่นักเรียนได้ทำการแก้ไขความผิดพลาดต่างๆที่ได้เติมลงไปในโปรแกรมเพื่อไว้ใช้ยืนยันว่าโปรแกรมได้ถูกแก้ไขโดยสมบูรณ์

8) จดจำลักษณะความผิดพลาดที่ได้แก้ไขไป หลังจากการทดสอบและตรวจพบว่าเกิดความผิดพลาดขึ้นจริงจะทำการจดจำลักษณะข้อบกพร่องที่ได้แก้ไขไปเพื่อนำไปใช้ในการแก้ไขอีกทีในตอนสุดท้ายโดยมีรายละเอียดการทำงานดังรหัสตัวอย่างในรูปที่ 3.14 ดังนี้

```
ast.rewrite(d, null).apply(d, TextEdit.UPDATE_REGIONS)
d.repairLineInformation();
if (!outputTestCodeAndCompile(d)||test() != 1.0) {
    realModList.add(modifier);
}
```

รูปที่ 3.14 การแปลงโครงสร้างกลับสู่รหัสต้นฉบับและการทดสอบการทำงานและบันทึกการลักษณะข้อผิดพลาดที่ได้แก้ไขไป

9) ตรวจสอบความครบถ้วนตามข้อกำหนด ในการสร้างแบบฝึกหัดนั้นอาจมีบริเวณของรหัสต้นฉบับหลายบริเวณหรืออาจมีความต้องการในการเติมความผิดพลาดหลายๆลักษณะ โดยอาจต้องการให้เกิดข้อบกพร่องในแต่ละลักษณะมากกว่าหนึ่งครั้งในรหัสต้นฉบับของโปรแกรม จึงต้องมีการตรวจสอบเพื่อให้การดำเนินการแก้ไขเป็นไปอย่างครบถ้วน

10) รวมรวมและแก้ไขความผิดพลาดทั้งหมดตามที่ได้จดจำไว้ หลังจากการตรวจสอบว่าได้ดำเนินการเติมความผิดพลาดลงในรหัสต้นฉบับของโปรแกรมทีละจุดและทดสอบความผิดพลาดเหล่านั้นอย่างครบถ้วนแล้ว ระบบจะทำการรวบรวมข้อมูลจากที่ได้จดจำไว้ในขั้นตอนที่ 7 แล้วเติมความผิดพลาดทั้งหมดเหล่านั้นลงไปในโปรแกรมในลักษณะและตำแหน่งเดิม บนโครงสร้างของต้นไม้ไวยากรณ์ ต้นฉบับ

11) แปลงรหัสต้นฉบับจากโครงสร้างต้นไม้ไวยากรณ์กลับสู่รหัสต้นฉบับ เช่นเดียวกับขั้นตอนที่ 5 แต่จะกระทำการแปลงโครงสร้างต้นไม้ไวยากรณ์ที่ได้ดำเนินการแก้ไขตามข้อกำหนดความผิดพลาดทั้งหมด ซึ่งจะถูกดำเนินการภายใต้เมท็อด `realModify()` หลังจากนั้นจะทำการแปลงโครงสร้างต้นไม้ไวยากรณ์ที่ได้รับการแก้ไขกลับสู่รหัสต้นฉบับ เพื่อนำใช้เป็นแบบฝึกหัดและข้อสอบต่อไป

3.3.2 ขั้นตอนในการทำงานร่วมกันของ Visitor และ Modifier

ในการทำงานร่วมกันของ Visitor และ Modifier นั้นจะเป็นการร่วมกันในการทำหน้าที่เพื่อหารูปแบบและทำการแก้ไขเพื่อให้เกิดข้อบกพร่องต่างๆตามที่นักพัฒนาได้วิเคราะห์ไว้สำหรับข้อบกพร่องแต่ละชนิด ทำให้การทำงานของ Visitor และ Modifier ย่อมแตกต่างกันไปตามแต่และลักษณะของข้อบกพร่อง ซึ่งในที่นี้จะขออธิบายการทำงานโดยยกตัวอย่างการแก้ไขรหัสต้นฉบับเพื่อก่อให้เกิดการความผิดพลาดเนื่องจากการย้ายข้อความสั่งภายใน else ออกจาก ข้อความ if...else ดังตัวอย่างในรูปที่ 3.15

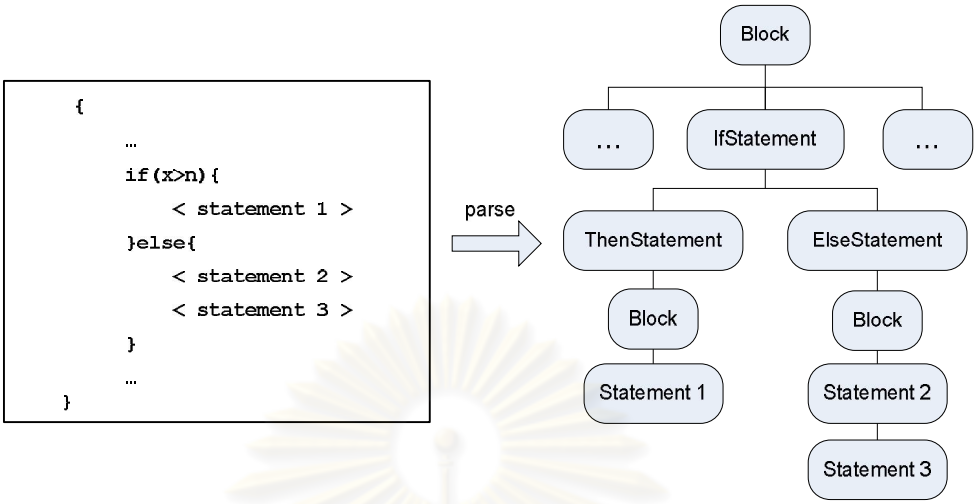
```
{
    …
    if(x>n){
        < statement 1 >
    }else{
        < statement 2 >
        < statement 3 >
    }
    …
}
```

รูปที่ 3.15.ก รหัสต้นฉบับก่อนการแก้ไข

```
{
    …
    if(x>n){
        < statement 1 >
    }else
        < statement 2 >
    < statement 3 >
    …
}
```

รูปที่ 3.15.ข รหัสต้นฉบับหลังการแก้ไข

โดยการทำงานของ Visitor และ Modifier จะเริ่มต้นการทำงานหลังจากการแปลงรหัสต้นฉบับของโปรแกรมให้อยู่ในรูปแบบของโครงสร้างต้นไม้ไวยากรณ์โดยการเรียกใช้งานคำสั่ง `ASTParserService.`*`parse`*`(<Document>)` โดย `ASTParser` จะคืนโครงสร้างของต้นไม้ไวยากรณ์ของรหัสต้นฉบับออกมาในรูปแบบของ `CompilationUnit` ซึ่งเป็นรากของโครงสร้างต้นไม้ ดังรูปที่ 3.16 และหลังจากได้โครงสร้างต้นไม้มาแล้วจะมีรายละเอียดขั้นตอนในการเติมข้อบกพร่องดังนี้



รูปที่ 3.16 โครงสร้างต้นไม้ไวยากรณ์เชิงนามธรรมของรหัสต้นฉบับ

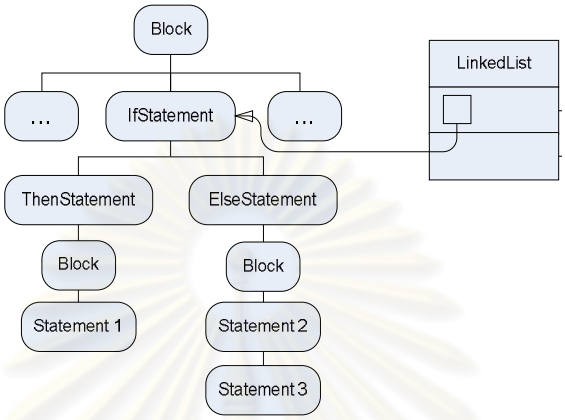
1) การค้นหาและสำรวจโครงสร้างต้นไม้ไวยากรณ์ การค้นหาและสำรวจโครงสร้างต้นไม้นั้นมีจุดประสงค์เพื่อหารูปแบบของโครงสร้างที่เหมาะสมและสอดคล้องสำหรับแก้ไขเพื่อให้เกิดข้อบกพร่องตามที่นักพัฒนาได้วิเคราะห์ไว้ ดังเช่นในตัวอย่างนี้จะเป็นการหาโครงสร้างของข้อความ if…else โดยที่กลุ่มของข้อความภายใต้ else นั้นจะต้องมากกว่า1ข้อความ ดังตัวอย่างรหัสเมท็อด visit() ของคลาส Visitor ในตัวอย่างที่ 3.17

```
class MovingElseStatementVisitor extends ASTVisitor{
   @overide mehtod
   public boolean visit(IfStatement node){
       if(node.getElseStatement()!=null&&
          (node.getElseStatement().getNodeType()==ASTNode.BLOCK)&&
          ((Block)node.getElseStatement()).statements().size()>1){

           if(node.getParent().getNodeType()==ASTNode.BLOCK){
             LinkedList<ASTNode> list= new LinkedList<ASTNode>();
             list.add(node);
             collectionNode.add(list);
        }
        }
       return true;
   }
}
```

รูปที่ 3.17 เมท็อด visit ของ คลาส visitor

2) รูปแบบทั้งหมดที่เป็นไปตามเงื่อนไขสำหรับการเพิ่มเติมข้อบกพร่องที่ได้จากการค้นหาในโครงสร้างต้นไม้ไวยากรณ์ จะถูกบันทึกตำแหน่งของปมลงในลำดับรายการโยงตัวอย่างการทำงานมีขั้นตอนดังรูปที่ 3.18



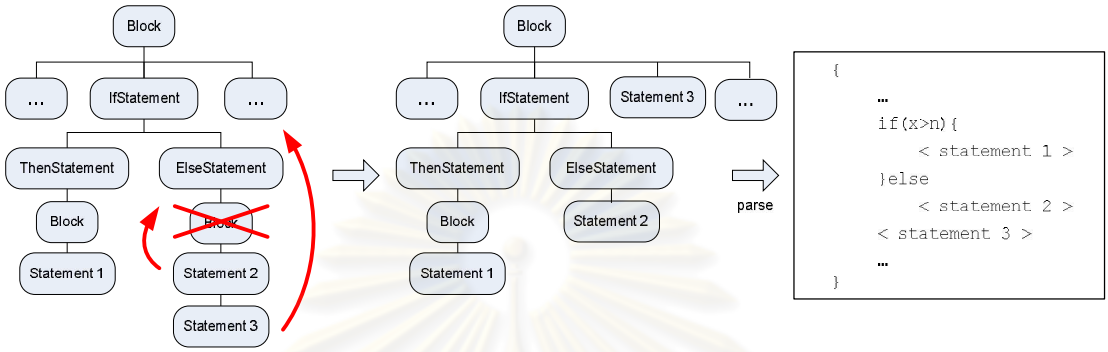
รูปที่ 3.18 ตำแหน่งของปมทั้งหมดที่มีรูปแบบตามเงื่อนไขที่กำหนดจะถูกบันทึกลงรายการโยง

3) หลังจากได้ตำแหน่งทั้งหมดของปมที่เป็นไปตามเงื่อนไขแล้วจะทำการส่งต่อตำแหน่งเหล่านี้ให้กับ Modifier ผ่านทาง EnbuggerManager ซึ่ง EnbuggerManager จะทำการสุ่มตำแหน่งที่จะทำการแก้ไขให้กับ Modifier และให้ Modifier เข้าทำการแก้ไขโดยมีรายละเอียดดังรหัสตัวอย่างในรูปที่ 3.19 โดยผลที่ได้จะทำให้โครงสร้างต้นไม้ของรหัสต้นฉบับเปลี่ยนแปลงไปดังรูปที่ 3.20

```
class MovingElseStatementModifier extends ASTModifier{
   @override
   public void modify(){
     AST ast = primaryModNode.getParent().getAST();
     Block ifBlock = (Block)primaryModNode.getElseStatement();
     List oldList = ifBlock.statements();
     Statement firstSt =(Statement)oldList.get(0);
     oldList.remove(firstSt);
     Statement s = ASTNode.copySubtree(ast, firstSt);
     primaryModNode.setElseStatement(s);
     List oldListForAdd = ASTNode.copySubtrees(ast,oldList);
     Block b=(Block)primaryModNode.getParent();
     List newList = b.statements();
     newList.addAll(newList.indexOf(primaryModNode)+1,
                    oldListForAdd);
   }
}
```

รูปที่ 3.19 เมท็อด visit ของ คลาส visitor

4) เมื่อได้โครงสร้างที่เปลี่ยนแปลงแล้วก็จะทำการแปลงโครงสร้างกลับสู่รหัสต้นฉบับซึ่งจะเกิดข้อบกพร่องขึ้นตามที่ต้องการพร้อมสำหรับการนำไปทดสอบก่อนนำไปใช้เป็นแบบฝึกหัดต่อไป โดยจะได้ผลดังรูปที่ 3.20



รูปที่ 3.20 การแก้ไขโครงสร้างต้นไม้ของเมท็อด modify ของคลาส Modifierและการแปลงกลับ

3.4 การใช้งาน

จากการพัฒนาระบบผู้วิจัยได้ทำการพัฒนาระบบขึ้นบนโปรแกรมภาษาจาวา และรวบรวมไว้เป็นโปรแกรมสำเร็จเพื่อให้ง่ายแก่การใช้งาน การสั่งการเพื่อให้ระบบทำงานสามารถทำได้โดยใช้คำสั่งที่มีโครงสร้างดังนี้

<JDK java> -cp <class path> <properties> Enbug

1. <JDK Java> เป็นคำสั่งในการเรียก java.exe JDK environment
2. <class path> เป็นตำแหน่งไฟล์ของระบบที่จะใช้ในการประกอบการทำงานของระบบการเติมข้อบกพร่องอัตโนมัติรวมถึงตำแหน่งไฟล์ของระบบการเติมข้อบกพร่องอัตโนมัตินี้ด้วย
3. <properties> เป็นคุณสมบัติที่จะต้องระบุเพื่อใช้ในการทำงานสำหรับการเติมข้อบกพร่องลงในรหัสต้นฉบับประกอบด้วย

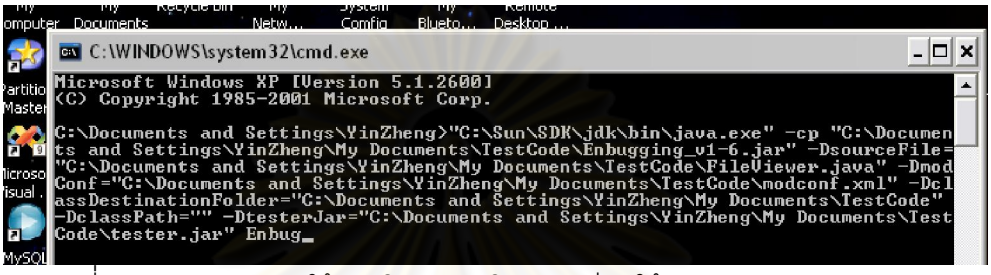
   -DsourceFile ใช้ในการระบุตำแหน่งของไฟล์รหัสต้นฉบับของโปรแกรมภาษาจาวาที่ต้องการเติมข้อบกพร่อง

   -DmodConf ใช้ในการระบุตำแหน่งของไฟล์ข้อกำหนดความผิดพลาดที่ใช้เป็นข้อมูลในการเติมความผิดพลาด

   -DclassDestinationFolder ตำแหน่งที่ใช้ในการวางไฟล์ที่ได้จากการแปลผลของโปรแกรม เพื่อใช้ในการทดสอบการทำงานขณะเติมข้อบกพร่องลงในโปรแกรม

-DclassPath เป็นตำแหน่งเพิ่มเติมสำหรับไฟล์ประกอบการทำงานของโปรแกรมที่จะทำการเติมข้อบกพร่อง

-DtesterJar เป็นตำแหน่งของโปรแกรมที่ใช้ในการทดสอบการทำงานของโปรแกรมที่ต้องการเติมข้อบกพร่อง



รูปที่ 3.21 ตัวอย่างการใช้งานโปรแกรมโดยการเรียนใช้งานผ่าน commandline

บทที่ 4

การพัฒนาส่วนเติมความผิดพลาดอัตโนมัติ

จากการวิเคราะห์และออกแบบการทำงานในบทที่ 3 และ ข้อมูลลักษณะความผิดพลาดต่างๆ ที่มักเกิดขึ้นอย่างบ่อยครั้งกับนักเรียนที่ได้มีการรายงานถึงในตัวอย่างเอกสารและรายงานการวิจัยในบทที่ 2 สามารถนำมาพัฒนาเป็นโปรแกรมเพื่อให้บริการการเติมความผิดพลาดแบบอัตโนมัติจากลักษณะของข้อบกพร่องแบบต่างๆโดยได้แบ่งลักษณะของข้อบกพร่องที่จะทำการเติมลงในรหัสต้นฉบับไว้เป็นข้อบกพร่องทั่วไปและกลุ่มของข้อบกพร่อง ซึ่งลักษณะของข้อบกพร่องต่างๆ เหล่านี้จะถูกนำไปใช้ในไฟล์ข้อกำหนดความผิดพลาดที่มีโครงสร้างเป็น xml เพื่อการระบุขอบเขตและลักษณะของความผิดพลาดที่จะเติมลงไปในรหัสต้นฉบับให้เป็นไปตามวัตถุประสงค์ในการเรียนการสอน โดยจะมีรายละเอียดดังนี้คือ

4.1.ข้อบกพร่องทั่วไป

ข้อบกพร่องทั่วไปเป็นข้อบกพร่องในแต่ละลักษณะที่มักเกิดขึ้นเป็นประจำในการเรียนการสอนวิชาการเขียนคอมพิวเตอร์ระดับพื้นฐานที่ได้รวบรวมจากรายงานต่างๆ ซึ่งได้ทำการแบ่งเป็นระดับย่อยๆได้อีกสามระดับคือ ความข้อบกพร่องทั่วไปในระดับเมท็อดเป็นข้อบกพร่องที่เกิดขึ้นบ่อยครั้งที่สุดในการเรียนการสอน ข้อบกพร่องทั่วไปในระดับคลาส และข้อบกพร่องพิเศษจากโครงสร้างภาษาจาวา โดยจะมีรายละเอียดของข้อบกพร่องต่างๆดังนี้

4.1.1 ข้อบกพร่องทั่วไปในระดับเมท็อด

ข้อบกพร่องทั่วไปในระดับเมท็อดเป็นลักษณะข้อบกพร่องที่พบบ่อยครั้งที่สุดในการเรียนการสอนการเขียนโปรแกรมระดับพื้นฐานเป็นข้อบกพร่องที่มีความซับซ้อนน้อย มักเกิดขึ้นภายในเมท็อดเท่านั้น ไม่ได้มีความเกี่ยวพันกับโครงสร้างของคลาส โดยมีลักษณะข้อบกพร่องในระดับนี้ทั้งหมด 6 กลุ่มลักษณะคือ

1) ข้อบกพร่องที่เกี่ยวข้องกับค่าของตัวเลขและค่าคงที่
2) ข้อบกพร่องที่เกี่ยวข้องกับตัวดำเนินการ
3) ข้อบกพร่องที่เกี่ยวข้องกับตัวแปร
4) ข้อบกพร่องที่เกี่ยวข้องกับนิพจน์(expression)
5) ข้อบกพร่องที่เกี่ยวข้องกับข้อความสั่ง(statement)
6) ข้อบกพร่องที่เกี่ยวกับโครงสร้างของเมท็อด

4.1.1.1 ข้อบกพร่องที่เกี่ยวข้องกับค่าตัวเลขและค่าคงที่

ข้อบกพร่องที่เกี่ยวข้องกับตัวเลขและค่าคงที่เป็นข้อบกพร่องทั่วไปที่เกิดจากแก้ไขเปลี่ยนแปลงค่าตัวเลขคงที่ต่างๆ และการใช้รูปแบบของตัวตัวเลขผิดรูปแบบ ทั้งตัวเลขที่เป็นจำนวนเต็มและเลขทศนิยม โดยจะมีการดำเนินการเปลี่ยนแปลงดังตารางที่ 4.1

ตารางที่ 4.1 การดำเนินการเปลี่ยนแปลงเพื่อให้เกิดข้อบกพร่องที่เกี่ยวข้องกับค่าคงที่

ชนิดข้อบกพร่องที่ดำเนินการ	ตัวอย่างรหัสต้นฉบับก่อนแก้ไข	ตัวอย่างรหัสต้นฉบับหลังแก้ไข

1) การดำเนินการเปลี่ยนค่าคงที่ต่างๆให้มีค่าเปลี่ยนแปลงไปเล็กน้อย ML_CoVaCh (METHOD_LEVEL_CONSTANT_VALUE_CHANGING)	a=a+1;
	b=1;
	c=10000l;
	d=h-1.2f;
	e=1.7d;
	character f='a';
	boolean f=false;
		a=a+2;
		b=-1;
		c=1000l;
		d=h-1.3f;
		e=1.6d;
		character f='b';
		boolean f=true;

2) การดำเนินการเปลี่ยนรูปแบบการแสดงผลเลขจำนวนเต็ม ML_InLiMiFo (METHOD_LEVEL_INTEGER_LITERAL_MISSING_FORMAT)	int a = 10;
	int b = 12;
	short c = 25;
		a = 010;
		b = 0x12;
		short c = 025;

3) การดำเนินการเปลี่ยนรูปแบบการแสดงผลของเลขทศนิยม ML_FlLiMiFo (METHOD_LEVEL_FLOAT_LITERAL_MISSING_FORMAT)	float a=1.1f;
	b+=1.0f;
	double c =i/7.0;
	c = 3.0;
		float a=1.1;
		b+=1.0;
		double c = i/7;
		c = 3.0f;

4.1.1.2 ข้อบกพร่องที่เกี่ยวข้องกับตัวดำเนินการ

ข้อบกพร่องทั่วไปที่เกี่ยวข้องกับตัวดำเนินการเป็นข้อพร่องที่เกิดขึ้นเนื่องจากการใช้ตัวดำเนินการต่างๆผิดพลาด ซึ่งเกิดจากการเพิ่มตัวดำเนินการบางตัวหรือลบตัวดำเนินการบางตัวออกไปเช่น ตัวดำเนินการทางเลขคณิตแบบเอกภาค(-) และตัวดำเนินการทางตรรกะแบบเอกภาค (!) การแก้ไขตัวดำเนินการในลักษณะต่างๆเช่น ตัวดำเนินการทางเลขคณิตแบบทวิภาค ตัวดำเนินการเลื่อนบิต ตัวดำเนินการเปรียบเทียบและความสัมพันธ์ ตัวดำเนินการทางตรรกะ และตัวดำเนินการการกำหนดค่า โดยจะมีรายละเอียดการดำเนินการเปลี่ยนแปลงดังตารางที่ 4.2

ตารางที่ 4.2 การดำเนินการเปลี่ยนแปลงเพื่อให้เกิดข้อบกพร่องที่เกี่ยวข้องกับตัวดำเนินการ

ชนิดข้อบกพร่องที่ดำเนินการ	ตัวอย่างรหัสต้นฉบับก่อนแก้ไข	ตัวอย่างรหัสต้นฉบับหลังแก้ไข

1) การดำเนินการเปลี่ยนตัวดำเนินการทางเลขคณิตแบบทวิภาค ML_ArBiOp (METHOD_LEVEL_ARITHMETIC_BINARY_ OPERATOR)	`a = b+10;`
	`b = a+5;`
	`d = b*9;`
	`e = b+c/a;`
	`f = c%b;`
		`a = b-10;`
		`b = a*5;`
		`d = b/9;`
		`e = b-c*a;`
		`f = c+b;`

2) การดำเนินการเปลี่ยนตัวดำเนินการทางเลขคณิตแบบเอกภาค ML_ArUnOp (METHOD_LEVEL_ARITHMETIC_UNARY_ OPERATOR)	`a = -b+10;`
	`b = a+c;`
	`d = b*-c;`
	`e = b+c/-a;`
		`a = b+10;`
		`b = a+-c;`
		`d = -b*c;`
		`e = -b+c/a;`

3) การดำเนินการเปลี่ยนตัวดำเนินการทางเลขคณิตแบบย่อ ML_ArShOp (METHOD_LEVEL_ARITHMETIC_SHORT_ OPERATOR)	`a++;` `b--;`	`++a;` `b++;`

ตารางที่ 4.2 การดำเนินการเปลี่ยนแปลงเพื่อให้เกิดข้อบกพร่องที่เกี่ยวข้องกับตัวดำเนินการ (ต่อ)

ชนิดข้อบกพร่องที่ดำเนินการ	ตัวอย่างรหัสต้นฉบับก่อนแก้ไข	ตัวอย่างรหัสต้นฉบับหลังแก้ไข

4) การดำเนินการเปลี่ยนตัวดำเนินการกำหนดค่าทางเลขคณิต ML_AsArOp (METHOD_LEVEL_ ASSIGN_ARITHMETIC_ OPERATOR)	a+=b+c;
	d=”test”;
	b-=c*a;
		a-=b+c;
		d+=”test”;
		b=c*a;

5) การดำเนินการเปลี่ยนตัวดำเนินการกำหนดค่าทางตรรกะ ML_AsLoOp (METHOD_LEVEL_ ASSIGN_LOGIC_ OPERATOR)	a=(b&c)|d;
	c|=d;
	c=d^a;
	d&=!a;
		a&=(b&c)|d;
		c=d;
		c^=d^a;
		d|=!a;

6) การดำเนินการเปลี่ยนตัวดำเนินการกำหนดค่าเลื่อนบิต ML_AsShOp (METHOD_LEVEL_ ASSIGN_SHIFT_ OPERATOR)	a=b-1;
	c<<=d;
	d>>=3;
		a=>>b-1;
		c=d;
		d>>>=3;

7) การดำเนินการเปลี่ยนตัวดำเนินการเลื่อนบิต ML_ShOp) METHOD_ LEVEL_SHIFL_ OPERATOR)	a=b^(c<<1); b=a>>>3;	a=b^(c>>1); b=a<<3;

ตารางที่ 4.2 การดำเนินการเปลี่ยนแปลงเพื่อให้เกิดข้อบกพร่องที่เกี่ยวข้องกับตัวดำเนินการ (ต่อ)

ชนิดข้อบกพร่องที่ดำเนินการ	ตัวอย่างรหัสต้นฉบับก่อนแก้ไข	ตัวอย่างรหัสต้นฉบับหลังแก้ไข

8) การดำเนินการเปลี่ยนตัวดำเนินการเปรียบเทียบความสัมพันธ์ ML_ReOp (METHOD_LEVEL_ RELATIONAL_ OPERATOR)	`a==b^(c<1);`
	`d>=a;`
	`c!=b;`
	`c<d;`
		`a!=b^(c<1);`
		`d<=a;`
		`c==b;`
		`c>d;`

9) การดำเนินการเปลี่ยนตัวดำเนินทางตรรกะแบบทวิภาค ML_LoCoBiOp (METHOD_LEVEL_ LOGICAL_ CONDITIONAL_ BINARY_OPERATOR)	`a||b;`
	`c&d;`
	`a&&c;`
	`b|d;`
	`a^e;`
		`a&&b;`
		`c|d;`
		`a^c;`
		`b&d;`
		`a|e;`

10) การดำเนินการเปลี่ยนตัวดำเนินทางตรรกะแบบเอกภาคML_LoCoUnOp (METHOD_LEVEL_ LOGICAL_ CONDITIONAL_ UNARY_OPERATOR)	`a||b;`
	`c&d;`
	`a&&c;`
		`a||!b;`
		`!c&d;`
		`!(a&&c);`

4.1.1.3 ข้อบกพร่องที่เกี่ยวข้องกับตัวแปร

ข้อบกพร่องทั่วไปที่เกี่ยวข้องกับตัวแปรเป็นความผิดพลาดที่เกิดขึ้นเนื่องจากการเปลี่ยนแปลงค่าเริ่มต้นของตัวแปรทั้งจากการเพิ่มเข้าหรือลบค่าตั้งต้นเหล่านั้นออก การเปลี่ยนแปลงชนิดของตัวแปรในการประกาศ หรือการเปลี่ยน ตัวแปรด้วยตัวแปรอื่นๆที่มีชนิดของตัวแปรเหมือนกันสามารถทำให้เกิดการทำงานที่ผิดพลาด โดยจะมีการดำเนินการเปลี่ยนแปลงดังตารางที่ 4.3

ตารางที่ 4.3 การดำเนินการเปลี่ยนแปลงเพื่อให้เกิดข้อบกพร่องที่เกี่ยวข้องกับตัวแปรต่างๆ

ชนิดข้อบกพร่องที่ดำเนินการ	ตัวอย่างรหัสต้นฉบับก่อนแก้ไข	ตัวอย่างรหัสต้นฉบับหลังแก้ไข

1) การดำเนินการเปลี่ยนตัวแปรชนิดพื้นฐาน ML_PrVaCh (METHOD_LEVEL_ PRIMITIVE_VARIABLE_ CHANGING)	`int a,b,c;`
	`double d,e;`
	`d=e*(a/b);`
		`int a,b,c;`
		`double d,e;`
		`d=d*(a/c);`

2) การดำเนินการเพิ่มค่าเริ่มต้นให้กับตัวแปรชนิดพื้นฐาน ML_PrCoInIn (METHOD_LEVEL_ PRIMITIVE_CONSTANT_ INITIALIZE_INSERTION)	`int a;`
	`long b;`
	`float c;`
	`double d;`
	`boolean e;`
	`character f;`
		`int a=1;`
		`long b=1l;`
		`float c=1.0f;`
		`double d=1.0d;`
		`boolean e=false;`
		`character f='d';`

3) การดำเนินการตัดค่าเริ่มต้นของตัวแปรชนิดพื้นฐานออก ML_PrCoInDe (METHOD_LEVEL_ PRIMITIVE_CONSTANT _INITIALIZE_DELETION)	`int a=1;`
	`long b=1l;`
	`float c=1.0f;`
	`double d=1.0d;`
	`boolean e=false;`
	`character f='d';`
		`int a;`
		`long b;`
		`float c;`
		`double d;`
		`boolean e;`
		`character f;`

4) การดำเนินการแก้ไขชนิดตัวแปรแบบพื้นฐานในการประกาศ ML_VaDeTyCh (METHOD_LEVEL_ VARIABLE_DECLARATIO N _TYPE_CHANGING)	`int x;`
	`long y;`
	`char z;`
	`double a;`
		`byte x;`
		`short y;`
		`int z;`
		`float a;`

ตารางที่ 4.3 การดำเนินการเปลี่ยนแปลงเพื่อให้เกิดข้อบกพร่องที่เกี่ยวข้องกับนิพจน์ต่างๆ (ต่อ)

ชนิดข้อบกพร่องที่ดำเนินการ	ตัวอย่างรหัสต้นฉบับก่อนแก้ไข	ตัวอย่างรหัสต้นฉบับหลังแก้ไข
5) การดำเนินการตัดค่าเริ่มต้นของตัวแปรที่ไม่ใช่ชนิดพื้นฐาน ML_NPrCoInDe (METHOD_LEVEL_ NONPRIMITIVE_ CONSTANT_INITIALIZE _ DELETION)	`CA a=new CA();`	`CA a;`

4.1.1.4 **ข้อบกพร่องที่เกี่ยวข้องกับนิพจน์** (expression)

ข้อบกพร่องทั่วไปที่เกี่ยวข้องกันการทำงานของนิพจน์เป็นความผิดพลาดที่เกี่ยวข้องกับนิพจน์แบบต่างๆเช่น นิพจน์การคำนวณค่าทางเลขคณิต นิพจน์การเรียกการทำงานของเมท็อด นิพจน์การเข้าถึงตำแหน่งข้อมูลในอาร์เรย์ซึ่งเป็นความผิดพลาดที่เกิดขึ้นจาก เพิ่มเติม แก้ไขเปลี่ยนแปลง สลับสับเปลี่ยนหรือลบข้อความบางข้อความออกเพื่อให้เกิดการประเมินค่าของนิพจน์ที่ผิดพลาด โดยจะมีการดำเนินการเปลี่ยนแปลงดังตารางที่ 4.4

ตารางที่ 4.4 การดำเนินการเปลี่ยนแปลงเพื่อให้เกิดข้อบกพร่องที่เกี่ยวข้องกับนิพจน์ต่างๆ

ชนิดข้อบกพร่องที่ดำเนินการ	ตัวอย่างรหัสต้นฉบับก่อนแก้ไข	ตัวอย่างรหัสต้นฉบับหลังแก้ไข
1) การดำเนินการลบนิพจน์การเปลี่ยนแปลงชนิดข้อมูลแบบพื้นฐาน ML_PrCaDe (METHOD_LEVEL_ PRIMITIVE_CASTING_ DELETION)	`double d =135.4d, e =43.2d;` `int i=(int)d + (int)e;`	`double d =135.4d, e =43.2d;` `int i=(int)d + e;`

ตารางที่ 4.4 การดำเนินการเปลี่ยนแปลงเพื่อให้เกิดข้อบกพร่องที่เกี่ยวข้องกับนิพจน์ต่างๆ (ต่อ)

ชนิดข้อบกพร่องที่ดำเนินการ	ตัวอย่างรหัสต้นฉบับก่อนแก้ไข	ตัวอย่างรหัสต้นฉบับหลังแก้ไข

2) การดำเนินการตัดคำสำคัญ thisด้านหน้าตัวแปรออก ML_ThFiAcDe (METHOD _LEVEL_THIS_FIELD_ ACCESS_DELETION)	`private int a;`
	`void m1(int a){`
	`  this.a=a;`
	`  ...`
	`}`
		`private int a;`
		`void m1(int a){`
		`  a=a;`
		`  ...`
		`}`

3) การดำเนินการเพิ่ม this ด้านหน้าตัวแปร ML_ThFiAcIn (METHOD_ LEVEL_THIS_FIELD_ ACCESS_INSERTION)	`private int a;`
	`void M1(int a){`
	`  ...`
	`  b=a+2;`
	`}`
		`private int a;`
		`void m1(int a){`
		`  ...`
		`  b=this.a+2;`
		`}`

4) การดำเนินการลบคำสำคัญ super ด้านหน้า เมท็อด ออก ML_SuMeAcDe (METHOD_LEVEL_ SUPER_METHOD_ ACCESS_DELETION)	`super.methodA();`	`methodA();`
5) การดำเนินการตัดคำสำคัญ super ด้านหน้าตัวแปรออก ML_SuFiAcDe (METHOD _LEVEL_SUPER_FIELD_ ACCESS_DELETION)	`super.memberA=a;`	`memberA=a;`

ตารางที่ 4.4 การดำเนินการเปลี่ยนแปลงเพื่อให้เกิดข้อบกพร่องที่เกี่ยวข้องกับนิพจน์ต่างๆ (ต่อ)

ชนิดข้อบกพร่องที่ดำเนินการ	ตัวอย่างรหัสต้นฉบับก่อนแก้ไข	ตัวอย่างรหัสต้นฉบับหลังแก้ไข
6) การดำเนินการสลับพารามิเตอร์ในการเรียกใช้งานเมท็อด ML_MeInPaSw (MEHTOD_LEVEL_METHOD_INVOCATION_PARAMETER_SWAPPING)	`method(a,b);`	`method(b,a);`
7) การดำเนินการแก้ไขการเรียกเมท็อดโดยไม่มีการรับค่า ML_FIMeIn (METHOD_LEVEL_FLOATING_METHOD_INVOCATION)	`a=b.method();`	`b.method();`
8) การดำเนินการแก้ไขการเข้าถึงอาร์เรย์ ML_ArAcCh (METHOD_LEVEL_ARRAY_ACCESS_CHANGING)	`a[i];` `a[i];`	`a[i+1];` `a[i-1];`
9) การดำเนินการสลับการเข้าถึงอาร์เรย์ ML_ArAcSw (METHOD_LEVEL_ARRAY_ACCESS_SWAPPING)	`a[i][j]`	`a[j][i]`

10) การดำเนินการแก้ไขนิพจน์เพิ่มค่า ML_UpExCh (METHOD_LEVEL_UPDATE_EXPRESSION_CHANGING)	`i++`
	`i--`
	`i++`
	`i--`
		`i+=2`
		`i-=2`
		`i`
		`i`

4.1.1.5 ความผิดพลาดที่เกี่ยวข้องกับข้อความสั่ง

ข้อบกพร่องทั่วไปที่เกี่ยวข้องกับการทำงานของข้อความสั่งเป็นความผิดพลาดที่เกี่ยวข้องกับข้อความสั่งหรือโครงสร้างของข้อความสั่งแบบต่างๆเช่น if...else, switch, for, do, while ,break ,return หรือข้อความการเรียกใช้งานเมท็อด ซึ่งเป็นความผิดพลาดที่เกิดขึ้นจากการแก้ไขเปลี่ยนแปลง สลับสับเปลี่ยนหรือลบข้อความสั่งบางข้อความออก โดยจะมีการดำเนินการเปลี่ยนแปลงดังตารางที่ 4.5

ตารางที่ 4.5 การดำเนินการเปลี่ยนแปลงเพื่อให้เกิดข้อบกพร่องที่เกี่ยวข้องกับข้อความสั่งต่างๆ

ชนิดข้อบกพร่องที่ดำเนินการ	ตัวอย่างรหัสต้นฉบับก่อนแก้ไข	ตัวอย่างรหัสต้นฉบับหลังแก้ไข

1) การดำเนินการแก้ไขลำดับในการตัดสินใจแบบ switch ML_SwCaSw (METHOD_LEVEL_SWITCH_CASE_SWAPPING)	switch(a){
	case 1 ...
	case 2 ...
	case 3 ...
	}
		switch(a){
		case 1 ...
		case 3 ...
		case 2 ...
		}

2) การดำเนินการในการลบเงื่อนไขในการตัดสินใจข้อความ switch ของ ML_SwCaDe (METHOD_LEVEL_SWITCH_CASE_DELETION)	switch(a){
	case 1 ...
	case 2 ...
	case 3 ...
	case 4 ...
	}
		switch(a){
		case 1 ...
		...
		case 3 ...
		...
		}

3) การดำเนินการลบข้อความสั่งหยุดในข้อความ switch ของ ML_SwBrDe (METHOD_LEVEL_SWITCH_BREAK_DELETION)	switch(a){
	case 1 ...
	break;
	case 2 ...
	case 3 ...
	break;
	case 4 ...
	break;
	}
		switch(a){
		case 1 ...
		break;
		case 2 ...
		case 3 ...
		case 4 ...
		break;
		}

ตารางที่ 4.5 การดำเนินการเปลี่ยนแปลงเพื่อให้เกิดข้อบกพร่องที่เกี่ยวข้องกับข้อความสั่งต่างๆ (ต่อ)

ชนิดข้อบกพร่องที่ดำเนินการ	ตัวอย่างรหัสต้นฉบับก่อนแก้ไข	ตัวอย่างรหัสต้นฉบับหลังแก้ไข

4) การดำเนินการลบวงเล็บปีกกาจากข้อความสั่งของ if…else ของ ML_If1StCh (METHOD_LEVEL_IF1_STATEMENT_CHANGING)	`if(<condition1>){`
	`  ...`
	`}else{`
	`  <statement>`
	`  ...`
	`}`
	`if(condition2){`
	`  <statement>`
	`  ...`
	`}`
		`if(<condition1>){`
		`  ...`
		`}else`
		`  <statement>`
		`...`
		
		`if(condition2)`
		`  <statement>`
		`...`

5) การดำเนินการสลับกลุ่มข้อความภายใน ข้อความสั่งของif…else ML_If2StCh (METHOD_LEVEL_IF2_STATEMENT_CHANGING)	`if(<condition1>){`
	`  <statement1>`
	`}else{`
	`  <statement2>`
	`}`
		`if(<condition1>){`
		`  <statement2>`
		`}else{`
		`  <statement1>`
		`}`

6) การดำเนินการลบข้อความ else จากข้อความ if…else ที่ซ้อนกันหลายๆชั้นของ ML_If3StCh (METHOD_LEVEL_IF3_STATEMENT_CHANGING)	`if(<condition1>){`
	`  <statement1>`
	`}else`
	`if(<condition2>){`
	`  <statement2>`
	`}else{`
	`  <statement3>`
	`}`
		`if(<condition1>){`
		`  <statement1>`
		`}`
		`if(<condition2>){`
		`  <statement2>`
		`}else{`
		`  <statement3>`
		`}`

7) การดำเนินการลบข้อความหยุดจากข้อความการวนซ้ำของ ML_ItBrDe (METHOD_LEVEL_ITERATION_BREAK_DELETION)	`for(){`
	`  <statement>`
	`  if(condition){`
	`  break;`
	`  }`
	`}`
		`for(){`
		`  <statement>`
		`  if(condition){`
		`  }`
		`}`

ตารางที่ 4.5 การดำเนินการเปลี่ยนแปลงเพื่อให้เกิดข้อบกพร่องที่เกี่ยวข้องกับข้อความสั่งต่างๆ (ต่อ)

ชนิดข้อบกพร่องที่ดำเนินการ	ตัวอย่างรหัสต้นฉบับก่อนแก้ไข	ตัวอย่างรหัสต้นฉบับหลังแก้ไข

8) การดำเนินการลบข้อความการประเมินค่าต่างๆจากข้อความการวนซ้ำของ ML_ItExDe (METHOD_LEVEL_ITERATION_EXPRESSION_DELETION)	`for(i=0;i<n;i++){`
	`  <statement>`
	`}`
	`while(i++){`
	`  <statement>`
	`}`
		`for(i=0;;){`
		`  <statement>`
		`}`
		`while(){`
		`  <statement>`
		`}`

9) การดำเนินการเปลี่ยนข้อความการวนซ้ำของ for จาก ML_ItFoStCh (METHOD_LEVEL_ITERATE_FOR_STRUCTURE_CHANGING)	`for(...;...;...){`
	`  ...`
	`}`
	
	`for(...;...;...){`
	`  ...`
	`}`
		`while(...){`
		`  ...`
		`}`
		
		`do{`
		`  ...`
		`}while(...)`

10) การดำเนินการเปลี่ยนข้อความการวนซ้ำของ do..while จาก ML_ItDoStCh (METHOD_LEVEL_ITERATE_DO_STRUCTURE_CHANGING)	`do{`
	`  <statements>`
	`}while(...)`
	
	`do{`
	`  <statements>`
	`}while(...)`
		`for(;...;){`
		`  <statements>`
		`}`
		
		`while(...){`
		`  <statements>`
		`}`

11) การดำเนินการเปลี่ยนข้อความการวนซ้ำของ while จาก ML_ItWhStCh (METHOD_LEVEL_ITERATE_WHILE_STRUCTURE_CHANGE)	`while(...){`
	`  <statements>`
	`}`
	
	`while(...){`
	`  <statements>`
	`}`
		`for(;...;){`
		`  <statements>`
		`}`
		
		`do{`
		`  <statements>`
		`}while(...)`

ตารางที่ 4.5 การดำเนินการเปลี่ยนแปลงเพื่อให้เกิดข้อบกพร่องที่เกี่ยวข้องกับข้อความสั่งต่างๆ (ต่อ)

ชนิดข้อบกพร่องที่ดำเนินการ	ตัวอย่างรหัสต้นฉบับก่อนแก้ไข	ตัวอย่างรหัสต้นฉบับหลังแก้ไข
12) การดำเนินการสลับข้อความในML_StSeSw (METHOD_LEVEL_ STATEMENT_ SEQUENCE_ SWAPPING)	`f(a);` `a=b+c;`	`a=b+c;` `f(a);`

13) การดำเนินการล้างการคืนค่าเป็นค่ามาตรฐานของข้อมูลชนิดพื้นฐานจาก ML_RePrRe (METHOD _LEVEL_RETURN_ PRIMITIVE_REMOVING)	`int getInt (){`
	`  return a;`
	`}`
	
	`float getFloat (){`
	`  return b;`
	`}`
		`int getInt (){`
		`  return 0;`
		`}`
		
		`float getFloat (){`
		`  return 0.0f;`
		`}`

14) การดำเนินการล้างการคืนค่าเป็นค่ามาตรฐานของข้อมูลชนิดวัตถุจาก ML_ReNPrRe (METHOD _LEVEL_RETURN_NONP RIMITIVE_REMOVING)	`Object getO (){`
	`  return a;`
	`}`
		`Object getO (){`
		`  return null;`
		`}`

15) การดำเนินการเปลี่ยนตัวแปรในการคืนค่าข้อมูลชนิดพื้นฐานใน ML_RePrCh (METHOD_ LEVEL_RETURN_PRIMITI VE_CHANGING)	`int getInt(){`
	`  ...`
	`  return a;`
	`}`
	
	`float getFloat(){`
	`  return b;`
	`  ...`
	`}`
		`int getInt(){`
		`  ...`
		`  return c;`
		`}`
		
		`float getFloat(){`
		`  return d;`
		`  ...`
		`}`

ตารางที่ 4.5 การดำเนินการเปลี่ยนแปลงเพื่อให้เกิดข้อบกพร่องที่เกี่ยวข้องกับข้อความสั่งต่างๆ (ต่อ)

ชนิดข้อบกพร่องที่ดำเนินการ	ตัวอย่างรหัสต้นฉบับก่อนแก้ไข	ตัวอย่างรหัสต้นฉบับหลังแก้ไข

16) การดำเนินการเปลี่ยนตัวแปรในการคืนค่าข้อมูลชนิดวัตถุใน ML_ReNPrCh (METHOD_LEVEL_ RETURN_NONPRIMITIVE _CHANGING)	`A getObject(){`
	`  return a;`
	`}`
		`A getObject(){`
		`  return childOfA;`
		`}`

17) การดำเนินการลบข้อความการคืนค่าบางอันออกจากเมท็อดใน ML_ReStDe (METHOD_LEVEL_ RETURN_STATEMENT_ DELETION)	`A getObject(){`
	`  ...`
	`  return a1;`
	`  ...`
	`  return a2;`
	`}`
		`A getObject(){`
		`  ...`
		`  ...`
		`  return a2;`
		`}`

18) การดำเนินการลบข้อความการเรียกใช้งานของเมท็อดออกจากการทำงานใน ML_MeInStDe(METHOD _LEVEL_METHOD _INVOCATION_ STATEMENT_DELETION	`<statement1>`
	`method();`
	`<statement2>`
		`<statement1>`
		`<statement2>`

19) การดำเนินการลบข้อความการเรียกใช้งานของเมท็อด super() ใน ML_SuCoDe (METHOD_LEVEL_ SUPER_CONSTRUCTOR_ DELETION)	`class CA extend A{`
	`  CA(){`
	`  super();`
	`  ...`
	`  }`
	`}`
		`class CA extend A{`
		`  CA(){`
		`  ...`
		`  }`
		`}`

4.2.1.6 ความผิดพลาดที่เกี่ยวข้องกับโครงสร้างเมท็อด

ความบกพร่องทั่วไปที่เกี่ยวข้องกับโครงสร้างของเมท็อดเป็นความผิดพลาดที่เกิดจากการแก้ไขโครงสร้างสร้างของเมท็อดทำให้โครงสร้างของเมท็อดเปลี่ยนไปนำมาซึ่งการทำงานที่ผิดพลาด อย่างเช่นการเปลี่ยนชนิดของการคืนค่าของเมท็อดโดยจะมีรายละเอียดการดำเนินการเปลี่ยนแปลงดังตารางที่ 4.6

ตารางที่ 4.6 การดำเนินการเปลี่ยนแปลงเพื่อให้เกิดข้อบกพร่องที่เกี่ยวข้องกับโครงสร้างของเมท็อด

ชนิดข้อบกพร่องที่ดำเนินการ	ตัวอย่างรหัสต้นฉบับก่อนแก้ไข	ตัวอย่างรหัสต้นฉบับหลังแก้ไข

1) การดำเนินการแก้ไขโดยการยกเลิกการคืนค่าของเมท็อดใน ML_ReMeDe (METHOD_LEVEL_RETURN_METHOD_DELETION)	`A getObject(){`
	`  ...`
	`  return a1;`
	`  ...`
	`  return a2;`
	`}`
		`void getObject(){`
		`  ...`
		`  ...`
		`}`

2) การดำเนินการแก้ไขเมท็อดตัวสร้าง(constructor)ให้กลายเป็นเมท็อดทั่วไปที่ไม่มีการคืนค่า ML_CoToMe (METHOD_LEVEL_CONSTRUCTOR_TO_METHOD)	`ClassA(){`
	`  ...`
	`}`
		`void ClassA(){`
		`  ...`
		`}`

3) การดำเนินการสลับพารามิเตอร์ของเมท็อดจาก ML_MeDeArSw (METHOD_LEVEL_METHOD_DECLARATION_ARGUMENTS_SWAPPING)	`int m1(int a,int b){`
	`  ...`
	`}`
		`int m1(int b,int a){`
		`  ...`
		`}`

4.1.2 ข้อบกพร่องทั่วไปในระดับคลาส

ข้อบกพร่องทั่วไปในระดับการทำงานของคลาสเป็นลักษณะข้อบกพร่องที่มักเกิดขึ้นภายในคลาสที่มีความเกี่ยวพันธ์กับโครงสร้างของคลาสและเป็นลักษณะความผิดพลาดในระดับคลาสที่มักเกิดขึ้นบ่อยครั้ง เช่นความผิดพลาดในการใช้คำสำคัญที่ใช้ควบคุมการเข้าถึงข้อมูลเช่น public, protect คำสำคัญ static หรือเรื่องอื่นๆในส่วนที่เป็นสมาชิกตัวแปรและสมาชิกเมท็อดของคลาส โดยจะเป็นการเพิ่มเติม ตัดข้อความบางข้อความหรือแก้ไขเปลี่ยนแปลงข้อความต่างๆ เพื่อให้ส่งผลกระทบกับโครงสร้างการทำงานของคลาส โดยจะมีรายละเอียดการดำเนินการเปลี่ยนแปลงดังตารางที่ 4.7

ตารางที่ 4.7 การดำเนินการเปลี่ยนแปลงเพื่อให้เกิดข้อบกพร่องในระดับคลาส

ชนิดข้อบกพร่องที่ดำเนินการ	ตัวอย่างรหัสต้นฉบับก่อนแก้ไข	ตัวอย่างรหัสต้นฉบับหลังแก้ไข
1) การดำเนินการแก้ไขคำสำคัญควบคุมการเข้าถึงข้อมูลใน CL_FiAcMoCh (CLASS_LEVEL_FIELD_ ACCESS_MODIFIER_ CHANGING)	`public int a;` `protected int b;`	`private int a;` `public int b;`

2) การดำเนินการลบค่าเริ่มต้นของตัวแปร CL_FiInDe (CLASS_ LEVEL_FIELD_ INITIALIZE_DELETION)	`public int a=5;`
	`float c=2.0f;`
	`double d=3.5;`
		`public int a;`
		`float c;`
		`double d;`

3) การดำเนินการแก้ไขคำสำคัญควบคุมการเข้าถึงเมท็อดใน CL_MeAcMoCh (CLASS_LEVEL_ METHOD_ACCESS_ MODIFIER_CHANGING)	`public int m1(){`
	`...`
	`}`
	`protected int m2{`
	`...`
	`}`
		`private int m1(){`
		`...`
		`}`
		`public int m2{`
		`…`
		`}`

ตารางที่ 4.7 การดำเนินการเปลี่ยนแปลงเพื่อให้เกิดข้อบกพร่องในระดับคลาส (ต่อ)

ชนิดข้อบกพร่องที่ดำเนินการ	ตัวอย่างรหัสต้นฉบับก่อนแก้ไข	ตัวอย่างรหัสต้นฉบับหลังแก้ไข
4) การดำเนินการลบคำสำคัญ static ออกจากข้อมูลของคลาสใน CL_FiStMoDe (CLASS_LEVEL_FIELD_ STATIC_MODIFIER_ DELETION)	`static int a;` `static Class b;`	`int a;` `Class b;`
5) การดำเนินการเพิ่มคำสำคัญ static ออกจากข้อมูลของคลาสใน CL_FiStMoIn(CLASS_ LEVEL_FIELD_STATIC _MODIFIER_INSERTION)	`int a;` `Class b;`	`static int a;` `static Class b;`

6) การดำเนินการในการตัดส่วนต่อประสานของคลาสใน CL_MiExIn (CLASS_ LEVEL_EXTEND_ INTERFACE_DELETION)	`class a`
	`implements In1,In2{`
	`...`
	`}`
		`class a`
		`implements In2 {`
		`...`
		`}`

4.1.3 ข้อบกพร่องพิเศษจากโครงสร้างของภาษาจาวา

ภาษาจาวาเป็นภาษาคอมพิวเตอร์เชิงวัตถุที่มีวิธีการในการดำเนินการทำงานของโปรแกรมบางอย่างที่เป็นเอกลักษณ์เช่นการเปรียบเทียบความเท่ากันของวัตถุ การเรียกของความยาวของสายอักขระ หรือการใช้งานเมท็อดของคลาสมาตรฐานของจาวา การเข้าใจเอกลักษณ์เหล่านี้ผิดย่อมนำมาซึ่งความผิดพลาดในแบบต่างๆ โดยจะมีรายละเอียดการดำเนินการเปลี่ยนแปลงดังตารางที่ 4.8

ตารางที่ 4.8 การดำเนินการเปลี่ยนแปลงเพื่อให้เกิดข้อบกพร่องในลักษณะพิเศษ

ชนิดข้อบกพร่องที่ดำเนินการ	ตัวอย่างรหัสต้นฉบับก่อนแก้ไข	ตัวอย่างรหัสต้นฉบับหลังแก้ไข
1) การดำเนินการแก้ไขการเรียกใช้เมท็อด length()ของ SP_MiCaLeMe (SPECIAL_MISTAKE_CALLING_LENGTH_METHOD)	`a.length();`	`a.length;`

2) การดำเนินการตัดการเรียกเมท็อด add()ของคลาสที่สืบทอดคุณสมบัติมาจากคลาส Collection ใน SP_FoCaAdMe (SPECIAL_FORGOT_CALLING_ADD_METHOD)	`List c=new C();`
	`int a=2;`
	`c.add(a);`
	…
		`List c=new C();`
		`int a=2;`
		…

3) การดำเนินการตัดการเรียกเมท็อด start() ของคลาสที่สืบทอดคุณสมบัติมาจากคลาส Thread ใน SP_FoCaStMe (SPECIAL_FORGOT_CALLING_START_METHOD)	`Thread a=new T(b);` `a. start();`	`Thread a=new T(b);`
4) การดำเนินการแก้ไขการเรียกเมท็อด equals() เพื่อการเปรียบเทียบ ใน SP_ObCoCh (SPECIAL_OBJECT_COMPARISON_CHANGING)	`c=a.equals(b);` `d=a==b;`	`c=a==b;` `d=a.equals(b);`

4.2 กลุ่มความบกพร่อง

กลุ่มของความผิดพลาดนี้จะเป็นการรวบรวมลักษณะของความผิดพลาดแต่ละลักษณะที่มักเกิดขึ้นด้วยกัน หรือ มีความคล้ายคลึงกัน เข้ามารวมกันไว้เป็นกลุ่มๆเพื่อให้ง่ายแก่การใช้งาน ซึ่งได้ทำการรวบรวมไว้ทั้งหมด 35 ลักษณะ โดยมีรายละเอียดดังนี้คือ

4.2.1) METHOD_LEVEL_PRIMITIVE_NUMBER_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_PrNuMo เป็นกลุ่มของข้อผิดพลาดที่เกิดจากการแก้ไขรูปแบบของตัวเลขเพื่อให้เกิดข้อผิดพลาด โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_InLiMiFo : METHOD_LEVEL_INTEGER_LITERAL_MISSING_FORMAT
2) ML_FlLiMiFo : METHOD_LEVEL_FLOAT_LITERAL_MISSING_FORMAT

4.2.2) METHOD_LEVEL_PRIMITIVE_CONSTANT_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_PrCoMo เป็นกลุ่มของข้อผิดพลาดที่เกิดจากการแก้ไขข้อมูลชนิดพื้นฐานต่างๆเพื่อให้เกิดข้อผิดพลาด โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_InLiMiFo : METHOD_LEVEL_INTEGER_LITERAL_MISSING_FORMAT
2) ML_FlLiMiFo : METHOD_LEVEL_FLOAT_LITERAL_MISSING_FORMAT
3) ML_CoVaCh : METHOD_LEVEL_CONSTANT_VALUE_CHANGING

4.2.3) METHOD_LEVEL_PRIMITIVE_INITIALIZE_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_PrInMo เป็นกลุ่มของข้อผิดพลาดที่เกิดจากการแก้ไขการกำหนดค่าตั้งต้นให้กับตัวแปรพื้นฐาน โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_PrCoInIn : METHOD_LEVEL_PRIMITIVE_CONSTANT_INITIALIZE_INSERTION
2) ML_PrCoInDe : METHOD_LEVEL_PRIMITIVE_CONSTANT_INITIALIZE_DELETION

4.2.4) METHOD_LEVEL_PRIMITIVE_VARIABLE_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_PrVaMo เป็นกลุ่มของข้อผิดพลาดที่เกิดกับตัวแปรชนิดพื้นฐานทั้งจากการแก้ไขค่าตั้งต้นของข้อมูลแก้ไขชนิดของข้อมูล หรือการสับเปลี่ยนตัวแปรชนิดพื้นฐาน โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_PrCoInIn : METHOD_LEVEL_PRIMITIVE_CONSTANT_INITIALIZE_INSERTION
2) ML_PrCoInDe : METHOD_LEVEL_PRIMITIVE_CONSTANT_INITIALIZE_DELETION
3) ML_VaDeTyCh : METHOD_LEVEL_VARIABLE_DECLARATION_TYPE_CHANGING

4) ML_PrVaCh: METHOD_LEVEL_PRIMITIVE_VARIABLE_CHANGING

4.2.5) METHOD_LEVEL_PRIMITIVE_DATA_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_PrDaMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นจากการแก้ไขเปลี่ยนแปลงข้อมูลหรือตัวแปรชนิดพื้นฐาน โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_InLiMiFo : METHOD_LEVEL_INTEGER_LITERAL_MISSING_FORMAT
2) ML_FlLiMiFo : METHOD_LEVEL_FLOAT_LITERAL_MISSING_FORMAT
3) ML_CoVaCh : METHOD_LEVEL_CONSTANT_VALUE_CHANGING
4) ML_PrCoInIn : METHOD_LEVEL_PRIMITIVE_CONSTANT_INITIALIZE_INSERTION
5) ML_PrCoInDe : METHOD_LEVEL_PRIMITIVE_CONSTANT_INITIALIZE_DELETION
6) ML_VaDeTyCh : METHOD_LEVEL_VARIABLE_DECLARATION_TYPE_CHANGING
7) ML_PrVaCh : METHOD_LEVEL_PRIMITIVE_VARIABLE_CHANGING

4.2.6) METHOD_LEVEL_VARIABLE_DATA_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_VaDaMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นกับการใช้งานตัวแปรในลักษณะต่างๆ โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_PrCoInIn : METHOD_LEVEL_PRIMITIVE_CONSTANT_INITIALIZE_INSERTION
2) ML_PrCoInDe : METHOD_LEVEL_PRIMITIVE_CONSTANT_INITIALIZE_DELETION
3) ML_NPrCoInDe : METHOD_LEVEL_NONPRIMITIVE_CONSTANT_INITIALIZE_ DELETION
4) ML_VaDeTyCh : METHOD_LEVEL_VARIABLE_DECLARATION_TYPE_CHANGING
5) ML_PrVaCh : METHOD_LEVEL_PRIMITIVE_VARIABLE_CHANGING

4.2.7) METHOD_LEVEL_ARITHMETIC_OPERATOR_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_ArOpMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นจากการแก้ไขเปลี่ยนแปลงตัวดำเนินการทางคณิตศาสตร์ได้แก่ตัวดำเนินการ +, -, * ,/ ,++ และ— โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_ArUnOp : METHOD_LEVEL_ARITHMETIC_UNARY_OPERATOR
2) ML_ArBiOp : METHOD_LEVEL_ARITHMETIC_BINARY_OPERATOR
3) Ml_ArShOp : METHOD_LEVEL_ARTHMETIC_SHORT_OPERATOR

4.2.8) METHOD_LEVEL_ARITHMETIC_EXPRESSION_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_ArExMo เป็นกลุ่มข้อผิดพลาดที่เกิดจากการแก้ไข

นิพจน์การประเมินค่าทางเลขคณิตซึ่งจะรวมทั้งตัวดำเนินการทางคณิตศาสตร์ โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_ArUnOp : METHOD_LEVEL_ARITHMETIC_UNARY_OPERATOR
2) ML_ArBiOp : METHOD_LEVEL_ARITHMETIC_BINARY_OPERATOR
3) ML_ArShOp : METHOD_LEVEL_ARTHMETIC_SHORT_OPERATOR
4) ML_AsArOp : METHOD_LEVEL_ASSIGN_ARITHMETIC_OPERATOR
5) ML_UpExCh : MEHTOD_LEVEL_UPDATE_EXPRESSION_CHANGING

4.2.9) METHOD_LEVEL_LOGICAL_CONDITIONAL_OPERATOR สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_LoCoMo เป็นกลุ่มของข้อผิดพลาดที่เกิดจากการแก้ไขเปลี่ยนแปลงตัวดำเนินการทางตรรกะโดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_LoCoBiOp: METHOD_LEVEL_LOGICAL_CONDITIONAL_BINARY_OPERATOR
2) ML_LoCoUnOp: METHOD_LEVEL_LOGICAL_CONDITIONAL_UNARY_OPERATOR

4.2.10) METHOD_LEVEL_LOGICAL_EXPRESSION_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_LoExMo เป็นกลุ่มของข้อผิดพลาดที่เกิดจากการแก้ไขเปลี่ยนแปลงนิพจน์ทางตรรกะโดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_LoCoBiOp: METHOD_LEVEL_LOGICAL_CONDITIONAL_BINARY_OPERATOR
2) ML_LoCoUnOp: METHOD_LEVEL_LOGICAL_CONDITIONAL_UNARY_OPERATOR
3) ML_AsLoOp: METHOD_LEVEL_ASSIGN_LOGICAL_OPERATOR

4.2.11) METHOD_LEVEL_BOOLEAN_EXPRESSION_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_BoExMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นบนนิพนจ์ที่มีการประเมินค่าความจริง โดยรวมถึงการประเมินค่าความสัมพันธ์ในนิพนจ์ด้วย โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_LoCoBiOp: METHOD_LEVEL_LOGICAL_CONDITIONAL_BINARY_OPERATOR
2) ML_LoCoUnOp: METHOD_LEVEL_LOGICAL_CONDITIONAL_UNARY_OPERATOR
3) ML_AsLoOp: METHOD_LEVEL_ASSIGN_LOGICAL_OPERATOR
4) ML_ReOp : METHOD_LEVEL_RELATIONAL_OPERATOR

4.2.12) METHOD_LEVEL_ASSIGNMENT_OPERATOR สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_AsOp เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นกับตัวดำเนินการในการกำหนดค่า

รูปแบบต่างๆทั้งการกำหนดค่าทางเลขคณิต การกำหนดค่าทางตรรกะ การกำหนด่าเลื่อนบิต โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_AsArOp : METHOD_LEVEL_ASSIGN_ARITHMETIC_OPERATOR
2) ML_AsLoOp : METHOD_LEVEL_ASSIGN_LOGICAL_OPERATOR
3) ML_AsShOp : METHOD_LEVEL_ASSIGN_SHIFT_OPERATOR

4.2.13) METHOD_LEVEL_TYPE_EXPRESSION_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_TyExMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นเนื่องจากการแก้ไขเปลี่ยนแปลงที่ส่งผลกระทบต่อชนิดของข้อมูลในนิพจน์ เช่นการลบการเปลี่ยนชนิดของข้อมูล (casting) หรือ การเปลี่ยนชนิดข้อมูลของตัวแปร โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_PrCaDe : METHOD_LEVEL_PRIMITIVE_CASTING_DELETION
2) ML_VaDeTyCh : METHOD_LEVEL_VARIABLE_DECLARATION_TYPE_CHANGING

4.2.14) METHOD_LEVEL_THIS_EXPRESSION_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_ThExMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นเนื่องจากการแก้ไขนิพจน์ที่มีการเรียกใช้คำสำคัญ this ในการเข้าถึงสมาชิกข้อมูลของคลาสจากการลบและเพิ่มคำสำคัญนี้ลงในนิพจน์ โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_ThFiAcDe : METHOD_LEVEL_THIS_FIELD_ACCESS_DELETION
2) Ml_ThFiAcIn : METHOD_LEVEL_FIELD_ACCESS_INSERTION

4.2.15) METHOD_LEVEL_SUPER_EXPESSION_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_SuExMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นเนื่องจากการแก้ไขเปลี่ยนแปลงนิพจน์ที่มีการใช้คำพิเศษ super ในการเข้าถึงสมาชิกข้อมูลและสมาชิกเมท็อดของคลาสแม่ โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_SuFiAcDe : METHOD_LEVEL_SUPER_FIELD_ACCESS_DELETION
2) ML_SuFiAcIn : METHOE_LEVEL_SUPER_FIELD_ACCESS_INSERTION

4.2.16) METHOD_LEVEL_CALLING_METHOD_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_CaMeMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นเนื่องจากการแก้ไขการเรียกใช้งานเมท็อดในลักษณะต่างๆเช่น การสลับพารามิเตอร์ของเมท็อด หรือลบตัวแปรที่จะรับค่าจากเมท็อดออก โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_MeInPaSw : METHOD_LEVEL_METHOD_INVOCATION_PARAMETER_SWAPPING

2) ML_FIMeIn : METHOD_LEVEL_FLOAT_METHOD_INVOCATION

4.2.17) METHOD_LEVEL_ARRAY_ACCESS_EXPRESSION_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_ArAcExMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นเนื่องจากการแก้ไขนิพจน์ในส่วนที่มีการเข้าถึงข้อมูลของอาร์เรย์โดยการเปลี่ยนค่านิพจน์ทางคณิตศาสตร์ หรือการสลับดัชนีสำหรับอาร์เรย์หลายมิติ ซึ่งประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_ArAcCh : METHOD_LEVEL_ARRAY_ACCESS_CHANGING
2) ML_ArAcSw : METHOD_LEVEL_ARRAY_ACCESS_SWAPPING

4.2.18) METHOD_LEVEL_UPDATE_EXPRESSION_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_UpExMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นเนื่องจากการแก้ไขนิพจน์การปรับค่าให้เป็นปัจจุบัน โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_ArShOp : METHOD_LEVEL_ARTHMETIC_SHORT_OPERATOR
2) ML_UpExCh : METHOD_LEVEL_UPDATE_EXPRESSION_CHANGING

4.2.19) METHOD_LEVEL _EXPRESSION_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_ExMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นเนื่องจากการแก้ไขเปลี่ยนแปลงนิพจน์ต่างในทุกๆลักษณะซึ่งประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_PrCaDe : METHOD_LEVEL_PRIMITIVE_CASTING_DELETION
2) ML_ThFiAcDe : METHOD_LEVEL_THIS_FIELD_ACCESS_DELETION
3) Ml_ThFiAcIn : METHOD_LEVEL_FIELD_ACCESS_INSERTION
4) ML_SuFiAcDe : METHOD_LEVEL_SUPER_FIELD_ACCESS_DELETION
5) ML_SuFiAcIn : METHOE_LEVEL_SUPER_FIELD_ACCESS_INSERTION
6) ML_MeInPaSw : METHOD_LEVEL_METHOD_INVOCATION_PARAMETER_ SWAPPING
7) ML_FIMeIn : METHOD_LEVEL_FLOAT_METHOD_INVOCATION
8) ML_ArAcCh : METHOD_LEVEL_ARRAY_ACCESS_CHANGING
9) ML_ArAcSw : METHOD_LEVEL_ARRAY_ACCESS_SWAPPING
10) ML_UpExCh : METHOD_LEVEL_UPDATE_EXPRESSION_CHANGING

4.2.20) METHOD_LEVEL_SWITCH_STATEMENT_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_SwStMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นเนื่องจากการแก้ไข

เปลี่ยนแปลงโครงสร้างของข้อความสั่งที่เกี่ยวข้องกับข้อความ switch ซึ่งประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_SwCaSw : METHOD_LEVEL_SWITCH_CASE_SWAPPING
2) ML_SwCaDe : METHOD_LEVEL_SWITCH_CASE_DELETION
3) ML_SwBrDe : METHOD_LEVEL_SWITCH_BREAK_DELETION

4.2.21) METHOD_LEVEL_IF_STATEMENT_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_IfStMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นเนื่องจากแก้ไขเปลี่ยนแปลงโครงสร้างของข้อความสั่งที่เกี่ยวข้องกับข้อความ if และ if…else โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_If1StCh : METHOD_LEVEL_IF1_STATEMENT_CHANGING
2) ML_If2StCh : METHOD_LEVEL_IF2_STATEMENT_CHANGING
3) ML_If3StCh : METHOD_LEVEL_IF3_STATEMENT_CHANGING

4.2.22) METHOD_LEVEL_DECISION_MODIFICAITON สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_DeMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นเนื่องจากการแก้ไขเปลี่ยนแปลงโครงสร้างของข้อความที่เกี่ยวกับข้อความการตัดสินใจทั้งจากข้อความ switch, if และ if… else โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_SwCaSw : METHOD_LEVEL_SWITCH_CASE_SWAPPING
2) ML_SwCaDe : METHOD_LEVEL_SWITCH_CASE_DELETION
3) ML_SwBrDe : METHOD_LEVEL_SWITCH_BREAK_DELETION
4) ML_If1StCh : METHOD_LEVEL_IF1_STATEMENT_CHANGING
5) ML_If2StCh : METHOD_LEVEL_IF2_STATEMENT_CHANGING
6) ML_If3StCh : METHOD_LEVEL_IF3_STATEMENT_CHANGING

4.2.23) METHOD_LEVEL_ITERATION_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_ItMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นเนื่องจากการแก้ไขเปลี่ยนแปลงข้อความสั่งที่เกี่ยวข้องกับข้อความคำสั่งการวนซ้ำทั้งหมาด โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_ItBrDe : METHOD_LEVEL_ITERATION_BREAK_DELETION
2) ML_ItExDe : METHOD_LEVEL_ITERATION_EXPRESSION_DELETION
3) ML_ItFoStCh : METHOD_LEVEL_ITERATION_FOR_STATEMENT_CHANGING

4) ML_ItDoStch : METHOD_LEVEL_ITERATION_DO_STATEMENT_CHANGING

5) ML_ItWhStCh : METHOD_LEVEL_ITERATION_WHILE_STATEMENT_CHANGING

4.2.24) METHOD_LEVEL_RETURN_PRIMITIVE_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_RePrMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นเนื่องจากการแก้ไขเปลี่ยนแปลงข้อความการคืนค่าของเมท็อดแบบพื้นฐานในลักษณะต่างๆ โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_RePrRe : METHOD_LEVEL_RETURN_PRIMITIVE_REMOVING

2) ML_RePrCh : METHOD_LEVEL_RETURN_PRIMITIVE_CHANGING

4.2.25) METHOD_LEVEL_RETURN_NONPRIMITIVE_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_NRePrMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นเนื่องจากการแก้ไขเปลี่ยนแปลงข้อความการคืนค่าของเมท็อดแบบวัตถุในลักษณะต่างๆ โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_ReNPrRe : METHOD_LEVEL_RETURN_NONPRIMITIVE_REMOVING

2) ML_ReNPrCh : METHOD_LEVEL_RETURN_NONPRIMITIVE_CHANGING

4.2.26) METHOD_LEVEL_RETURN_STATEMENT_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_ReStMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นเนื่องจากแก้ไข ลบออก หรือเปลี่ยนแปลงตัวแปรของข้อความสั่งที่เกี่ยวข้องกับข้อความการคืนค่าของเมท็อดในลักษณะต่างๆ ซึ่งประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_ReStDe : METHOD_LEVEL_RETURN_STATEMENT_DELETION

2) ML_RePrRe : METHOD_LEVEL_RETURN_PRIMITIVE_REMOVING

3) ML_RePrCh : METHOD_LEVEL_RETURN_PRIMITIVE_CHANGING

4) ML_ReNPrRe : METHOD_LEVEL_RETURN_NONPRIMITIVE_REMOVING

5) ML_ReNPrCh : METHOD_LEVEL_RETURN_NONPRIMITIVE_CHANGING

4.2.27) METHOD_LEVEL_STATEMENET_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_StMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นเนื่องจากการแก้ไขเปลี่ยนแปลงข้อความสั่งในระดับเมท็อดทั้งหมด โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_SwCaSw : METHOD_LEVEL_SWITCH_CASE_SWAPPING

2) ML_SwCaDe : METHOD_LEVEL_SWITCH_CASE_DELETION

3) ML_SwBrDe : METHOD_LEVEL_SWITCH_BREAK_DELETION

4) ML_If1StCh : METHOD_LEVEL_IF1_STATEMENT_CHANGING
5) ML_If2StCh : METHOD_LEVEL_IF2_STATEMENT_CHANGING
6) ML_If3StCh : METHOD_LEVEL_IF3_STATEMENT_CHANGING
7) ML_ItBrDe : METHOD_LEVEL_ITERATION_BREAK_DELETION
8) ML_ItExDe : METHOD_LEVEL_ITERATION_EXPRESSION_DELETION
9) ML_ItFoStCh : METHOD_LEVEL_ITERATION_FOR_STATEMENT_CHANGING
10) ML_ItDoStch : METHOD_LEVEL_ITERATION_DO_STATEMENT_CHANGING
11) ML_ItWhStCh : METHOD_LEVEL_ITERATION_WHILE_STATEMENT_CHANGING
12) ML_StSeSw : METHOD_LEVEL_STATEMENT_SEQUENCE_SWAPPING
13) ML_ReStDe : METHOD_LEVEL_RETURN_STATEMENT_DELETION
14) ML_RePrRe : METHOD_LEVEL_RETURN_PRIMITIVE_REMOVING
15) ML_RePrCh : METHOD_LEVEL_RETURN_PRIMITIVE_CHANGING
16) ML_ReNPrRe : METHOD_LEVEL_RETURN_NONPRIMITIVE_REMOVING
17) ML_ReNPrCh : METHOD_LEVEL_RETURN_NONPRIMITIVE_CHANGING
18) ML_MeInStDe : METHOD_LEVEL_METHOD_INVOCATION_STATEMENT_ DELETION
19) ML_SuInStDe : METHOD_LEVEL_SUPER_INVOCATION_STATEMENT_DELETION

4.2.28) METHOD_LEVEL_METHOD_STRUCTURE_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML_MeStMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นเนื่องจากการแก้ไขเปลี่ยนแปลงโครงสร้างของเมท็อด โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_ReMeDe : METHOD_LEVEL_RETURN_METHOD_DELETION
2) ML_CoToMe : METHOD_LEVEL_CONSTRUCTOR_TO_METHOD
3) ML_MeDeArSw : METHOD_LEVEL_METHOD_DECLARATION_ARGUMENT_ SWAPPING

4.2.29) CLASS_LEVEL_FIELD_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น CL_FiMo เป็นกลุ่มของข้อผิดพลาดในระดับโครงสร้างของคลาสที่เกิดขึ้นจากการแก้ไขคำสำคัญด้านหน้าการระกาศสมาชิกตัวแปรที่ใช้ในการเก็บข้อมูลของคลาส โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) CL_FiAcMoCh : CLASS_LEVEL_FIELD_ACCESS_MODIFICATION_CHANGING

2) CL_FiInDe : CLASS_LEVEL_FIELD_INITIALIZE_DELETION
3) CL_FiStMoDe : CLASS_LEVEL_FIELD_STATIC_MODIFICATION_DELETION
4) CL_FiStMoIn :CLASS_LEVEL_FIELD_STATIC_MODIFICATION_INSERTION

4.2.30) CLASS_LEVEL_ACCESS_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น CL_AcMo เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นเนื่องจากการแก้ไขเปลี่ยนแปลงคำสำคัญที่ใช้ระบุในการเข้าถึงสมาชิกของคลาสทั้งสมาชิกเมท็อดและสมาชิกตัวแปรของคลาส โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) CL_FiAcMoCh : CLASS_LEVEL_FIELD_ACCESS_MODIFICATION_CHANGING
2) CL_MeAcMoCh : CLASS_LEVEL_METHOD_ACCESS_MODIFICAITON_ CHANGING

4.2.31) CLASS_LEVEL_STATIC_MODIFICATION สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น CL_StMo เป็นกลุ่มของข้อผิดพลาดในระดับคลาสที่เกิดขึ้นเนื่องจากการแก้ไขโดยการเพิ่มหรือลบคำสำคัญ staticให้กับสมาชิกตัวแปรของคลาส โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) CL_FiStMoDe : CLASS_LEVEL_FIELD_STATIC_MODIFICATION_DELETION
2) CL_FiStMoIn :CLASS_LEVEL_FIELD_STATIC_MODIFICATION_INSERTION

4.2.32) METHOD_LEVEL สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ML เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นในระดับเมท็อดทั้งหมดซึ่งเป็นข้อบกพร่องที่มักพบเป็นประจำในการเรียนการสอนการเขียนโปรแกรมคอมพิวเตอร์ระดับพื้นฐาน โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) ML_InLiMiFo : METHOD_LEVEL_INTEGER_LITERAL_MISSING_FORMAT
2) ML_FlLiMiFo : METHOD_LEVEL_FLOAT_LITERAL_MISSING_FORMAT
3) ML_CoVaCh : METHOD_LEVEL_CONSTANT_VALUE_CHANGING
4) ML_PrCoInIn : METHOD_LEVEL_PRIMITIVE_CONSTANT_INITIALIZE_INSERTION
5) ML_PrCoInDe : METHOD_LEVEL_PRIMITIVE_CONSTANT_INITIALIZE_DELETION
6) ML_VaDeTyCh : METHOD_LEVEL_VARIABLE_DECLARATION_TYPE_CHANGING
7) ML_PrVaCh : METHOD_LEVEL_PRIMITIVE_VARIABLE_CHANGING
8) ML_NPrCoInDe : METHOD_LEVEL_NONPRIMITIVE_CONSTANT_INITIALIZE_ DELETION
9) ML_ArUnOp : METHOD_LEVEL_ARITHMETIC_UNARY_OPERATOR

10) ML_ArBiOp : METHOD_LEVEL_ARITHMETIC_BINARY_OPERATOR

11) ML_ArShOp : METHOD_LEVEL_ARTHMETIC_SHORT_OPERATOR

12) ML_AsArOp : METHOD_LEVEL_ASSIGN_ARITHMETIC_OPERATOR

13) ML_AsShOp : MEHTOD_LEVEL_ASSIGN_SHIFT_OPERATOR

14) ML_ShOp : METHOD_LEVEL_SHIFT_OPERATOR

15) ML_LoCoBiOp: METHOD_LEVEL_LOGICAL_CONDITIONAL_BINARY_OPERATOR

16) ML_LoCoUnOp: METHOD_LEVEL_LOGICAL_CONDITIONAL_UNARY_OPERATOR

17) ML_AsLoOp : METHOD_LEVEL_ASSIGN_LOGICAL_OPERATOR

18) ML_ReOp : METHOD_LEVEL_RELATIONAL_OPERATOR

19) ML_PrCaDe : METHOD_LEVEL_PRIMITIVE_CASTING_DELETION

20) ML_ThFiAcDe : METHOD_LEVEL_THIS_FIELD_ACCESS_DELETION

21) Ml_ThFiAcIn : METHOD_LEVEL_FIELD_ACCESS_INSERTION

22) ML_SuFiAcDe : METHOD_LEVEL_SUPER_FIELD_ACCESS_DELETION

23) ML_SuMeAcDe : METHOD_LEVEL_SUPER_METHOD_ACCESS_DELETION

24) ML_MeInPaSw : METHOD_LEVEL_METHOD_INVOCATION_PARAMETER_ SWAPPING

25) ML_FlMeIn : METHOD_LEVEL_FLOAT_METHOD_INVOCATION

26) ML_ArAcCh : METHOD_LEVEL_ARRAY_ACCESS_CHANGING

27) ML_ArAcSw : METHOD_LEVEL_ARRAY_ACCESS_SWAPPING

28) ML_UpExCh : METHOD_LEVEL_UPDATE_EXPRESSION_CHANGING

29) ML_SwCaSw : METHOD_LEVEL_SWITCH_CASE_SWAPPING

30) ML_SwCaDe : METHOD_LEVEL_SWITCH_CASE_DELETION

31) ML_SwBrDe : METHOD_LEVEL_SWITCH_BREAK_DELETION

32) ML_If1StCh : METHOD_LEVEL_IF1_STATEMENT_CHANGING

33) ML_If2StCh : METHOD_LEVEL_IF2_STATEMENT_CHANGING

34) ML_If3StCh : METHOD_LEVEL_IF3_STATEMENT_CHANGING

35) ML_ItBrDe : METHOD_LEVEL_ITERATION_BREAK_DELETION

36) ML_ItExDe : METHOD_LEVEL_ITERATION_EXPRESSION_DELETION

37) ML_ItFoStCh : METHOD_LEVEL_ITERATION_FOR_STATEMENT_CHANGING

38) ML_ItDoStch : METHOD_LEVEL_ITERATION_DO_STATEMENT_CHANGING

39) ML_ItWhStCh : METHOD_LEVEL_ITERATION_WHILE_STATEMENT_CHANGING

40) ML_StSeSw : METHOD_LEVEL_STATEMENT_SEQUENCE_SWAPPING

41) ML_ReStDe : METHOD_LEVEL_RETURN_STATEMENT_DELETION

42) ML_RePrRe : METHOD_LEVEL_RETURN_PRIMITIVE_REMOVING

43) ML_RePrCh : METHOD_LEVEL_RETURN_PRIMITIVE_CHANGING

44) ML_ReNPrRe : METHOD_LEVEL_RETURN_NONPRIMITIVE_REMOVING

45) ML_ReNPrCh : METHOD_LEVEL_RETURN_NONPRIMITIVE_CHANGING

46) ML_MeInStDe : METHOD_LEVEL_METHOD_INVOCATION_STATEMENT_ DELETION

47) ML_SuInStDe : METHOD_LEVEL_SUPER_INVOCATION_STATEMENT_DELETION

48) ML_ReMeDe : METHOD_LEVEL_RETURN_METHOD_DELETION

49) ML_CoToMe : METHOD_LEVEL_CONSTRUCTOR_TO_METHOD

50) ML_MeDeArSw : METHOD_LEVEL_METHOD_DECLARATION_ARGUMENT_ SWAPPING

4.2.33) CLASS_LEVEL สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น CL เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นในระดับคลาส โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) CL_FiAcMoCh : CLASS_LEVEL_FIELD_ACCESS_MODIFICATION_CHANGING
2) CL_FiInDe : CLASS_LEVEL_FIELD_INITIALIZE_DELETION
3) CL_FiStMoDe : CLASS_LEVEL_FIELD_STATIC_MODIFICATION_DELETION
4) CL_FiStMoIn :CLASS_LEVEL_FIELD_STATIC_MODIFICATION_INSERTION
5) CL_MeAcMoCh : CLASS_LEVEL_METHOD_ACCESS_MODIFICAITON_ CHANGING
6) CL_ExInDe : CLASS_LEVEL_EXTEND_INTERFACE_DELETION

4.2.34) SPECIAL สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น SP เป็นกลุ่มของข้อผิดพลาดที่เกิดขึ้นกับลักษณะเฉพาะของโปรแกรมคอมพิวเตอร์ภาษาจาวา โดยประกอบด้วยลักษณะข้อผิดพลาดคือ

1) SP_MiCaLeMe : SPECIAL_MISTAKE_CALLING_LENGTH_METHOD
2) SP_FoCaAdMe : SPECIAL_FORGOT_CALLING_ADD_METHOD

3) SP_FoCaStMe: SPECIAL_FORGOT_CALLING_START_METHOD

4) SP_ObCoCh : SPECIAL_OBJECT_COMPARISON_CHANGING

4.2.35) ALL สามารถเรียกใช้โดยระบุลงในข้อผิดพลาดเป็น ALL เป็นกลุ่มของข้อผิดพลาดที่มีทั้งหมดในระบบ ซึ่งประกอบไปด้วยข้อบกพร่องทั่วไปในระดับเมท็อด ข้อบกพร่องในระดับคลาส และข้อบกพร่องพิเศษจากโครงสร้างของภาษาจาวา รวม 60 ลักษณะ

4.3 ไฟล์ข้อกำหนดความผิดพลาด

เป็นไฟล์สำหรับใช้ในการระบุขอบเขต ชนิดของข้อบกพร่อง และจำนวนตำแหน่งของข้อบกพร่องแต่ละชนิดที่จะทำการเติมลงในรหัสต้นฉบับ ซึ่งจะมีโครงสร้างเป็น XML เพื่อให้ง่ายในการนำระบบไปใช้งาน ซึ่งโครงสร้างของไฟล์ข้อกำหนดความผิดพลาดจะประกอบด้วยองค์ประกอบและคุณสมบัติต่างคือ

1. <modconf> เป็นองค์ประกอบหลักที่ใช้ในการทำงานของระบบ องค์ประกอบนี้ไม่มีคุณสมบัติ เพราะเป็นเพียงการกำหนดการเริ่มต้นการทำงานเท่านั้น

2. <scope> เป็นองค์ประกอบย่อยของ <modconf> เป็นองค์ประกอบที่ใช้ในการกำหนดขอบเขตของรหัสต้นฉบับที่จะทำการแก้ไขเพิ่มเติมข้อบกพร่อง ประกอบด้วย คุณสมบัติสองอย่างคือ
   2.1) className เป็นคุณสมบัติที่ใช้ในการระบุชื่อคลาสในรหัสต้นฉบับของโปรแกรมที่จะทำการแก้ไขเพิ่มเติมความผิดพลาด
   2.2) methodName เป็นคุณสมบัติที่ใช้ในการระบุชื่อของเมท็อดของคลาสที่ต้องการทำการแก้ไขเพิ่มเติมข้อบกพร่อง โดยจะต้องระบุชื่อของเมท็อดแบบเต็มรูปแบบเนื่องจากในโปรแกรมภาษาจาวายอมให้เมท็อดสามารถมีชื่อเหมือนกันได้แต่จะต้องมีพารามิเตอร์ที่แตกต่างกัน

3. <modify> เป็นองค์ประกอบย่อยของ <scope> เป็นองค์ประกอบที่ใช้ในการกำหนดชนิดและจำนวนของข้อบกพร่องที่ต้องการเติมลงในรหัสต้นฉบับของโปรแกรม ประกอบด้วยคุณสมบัติ

3.1) modType เป็นคุณสมบัติที่ใช้ในการระบุชนิดของข้อบกพร่องลักษณะต่างๆที่ต้องการเติมลงในรหัสต้นฉบับเพื่อให้เป็นไปตามจุดประสงค์ในการเรียนการสอน โดยสามารถระบุชนิดของข้อบกพร่องได้ทั้งข้อบกพร่องแบบกลุ่มและข้อบกพร่องแต่ละลักษณะ

3.2) modNum เป็นคุณสมบัติที่ใช้ในการระบุจำนวนของข้อบกพร่องที่ต้องการเติมลงในรหัสต้นฉบับของโปรแกรม ในแต่ละลักษณะที่ถูกกำหนดขึ้นในคุณสมบัติ modType

บทที่ 5

การทดสอบการทำงาน

การทดสอบการทำงานของระบบผู้วิจัยได้เลือกตัวอย่างรหัสของโปรแกรมจากแบบฝึกหัดสำหรับการฝึกปฏิบัติในการเรียนการสอนวิชาการทำโปรแกรมคอมพิวเตอร์ของภาควิชาวิศวกรรมศาสตร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย และตัวอย่างรหัสต้นฉบับจากหนังสือ Java Examples in a Nutshell Third Edition ของ David Flanagan ในการทดสอบ

5.1 การทดสอบการเติมความผิดพลาดลงในโปรแกรม (ตัวอย่างที่ 1)

การทดสอบการเติมความผิดพลาดนี้เป็นตัวอย่างการเติมความผิดพลาดลงในรหัสต้นฉบับจากหนังสือ Java Examples in a Nutshell โดยจะได้แสดงไฟล์ข้อมูลต่างๆ ได้แก่ไฟล์รหัสต้นฉบับก่อนทำการแก้ไขเพิ่มเติมข้อบกพร่องประกอบด้วย รหัสต้นฉบับและหมายเหตุประกอบการทำงานของโปรแกรม ไฟล์ข้อกำหนดความผิดพลาด และไฟล์รหัสต้นฉบับหลังจากการแก้ไขเพิ่มเติมข้อบกพร่องตามข้อกำหนด

5.1.1 ไฟล์รหัสต้นฉบับก่อนทำการแก้ไขเพิ่มเติมข้อบกพร่อง โปรแกรมนี้เป็นโปรแกรมแสดงเลขลำดับฟีโบนักชี 20 จำนวนแรกซึ่งได้มาจากการหาผลรวมของจำนวนก่อนหน้าสองจำนวนเข้าด้วยกัน โปรแกรมจะทำงานโดยการใช้ข้อความการวนซ้ำเพื่อบวกตัวเลขก่อนหน้าสองตัวเข้าด้วยกัน และจดจำไว้และจะวนซ้ำไปเรื่อยๆจนได้ลำดับฟีโบนักชีที่ต้องการ ซึ่งมีรายละเอียดดังรหัสตัวอย่างในรูปที่ 5.1.ก

```
/**
 * This program prints out the first 20 numbers in the
 * Fibonacci sequence. Each number is formed by adding
 * together the previous two numbers in the
 * sequence, starting with the numbers 0 and 1.
**/
public class Fibonacci {
  public static void main(String[] args) {
    int current, prev = 1, prevprev = 0;// Initialize some variables
    for(int i = 0; i < 20; i++) {       // Loop exactly 20 times
      current = prev + prevprev;        // Next number is sum of previous two
      System.out.print(current + " ");  // Print it out
      prevprev = prev;                  // First previous becomes 2nd previous
```

```
          prev = current;                    // And current number becomes previous
        }
        System.out.println();                // Terminate the line, and flush output
      }
    }
```

รูปที่ 5.1.ก ไฟล์รหัสต้นฉบับซึ่งประกอบด้วย และหมายเหตุประกอบการทำงานของตัวอย่างที่ 5.1

5.1.2 ไฟล์ข้อกำหนดความผิดพลาดที่ใช้ในการเติมข้อบกพร่องลักษณะต่างๆ ไฟล์ข้อกำหนดความผิดพลาดนี้เป็นไฟล์ xml โดยมีรายละเอียดดังรหัสตัวอย่างในรูปที่ 5.1.ข ซึ่งโครงสร้างสมาชิกที่ประกอบด้วย

- scope ที่ระบุถึงบริเวณที่จะทำการเติมความผิดพลาด โดยมีการระบุคุณลักษณะเป็นคลาส Fibonacci เมท็อด main
- modify เป็นส่วนที่ใช้ระบุถึงคุณลักษณะของข้อบกพร่องต่างๆและจำนวนจุดของความของข้อบกพร่องที่ต้องการเติมลงในรหัสต้นฉบับ เช่น ต้องการชนิดของข้อบกพร่องเป็น ML_ArBiOp จำนวน 1จุด

```
<?xml version="1.0" encoding="UTF-8"?>
<modconf>
  <scope className="Fibonacci" methodName=” public static void main(String[] args)”>
    <modify modType="ML_ArBiOp" modNum="1"/>
    <modify modType="ML_ReOp" modNum="1"/>
    <modify modType="ML_AsArOp" modNum="1"/>
    <modify modType="ML_InLiMiFo" modNum="1"/>
    <modify modType="ML_StSeSw" modNum="1"/>
  </scope>
</modconf>
```

รูปที่ 5.1.ข ไฟล์ข้อกำหนดความผิดพลาดของตัวอย่างที่ 5.1

5.1.3 ไฟล์รหัสต้นฉบับหลังการแก้ไขเพิ่มเติมข้อบกพร่องซึ่งจะพบเป็นรหัสต้นฉบับที่เปลี่ยนแปลงไป ดังรหัสตัวอย่างในรูปที่ 5.1ค

```
/**
 * This program prints out the first 20 numbers in the
 * Fibonacci sequence.Each number is formed by adding
 * together the previous two numbers in the
 *  sequence, starting with the numbers 0 and 1.
**/
```

```
public class Fibonacci {
  public static void main(String[] args) {
    int current, prev = 1, prevprev = 0; // Initialize some variables
    for(int i = 0; i <= 020; i++) {      // Loop exactly 20 times
      current = prev % prevprev;         // Next number is sum of previous two
      System.out.print(current + " ");   // Print it out
      prev += current;                   // First previous becomes 2nd previous
      prevprev = prev;                   // And current number becomes previous
    }
    System.out.println();                // Terminate the line, and flush output
  }
}
```

1
2
3
4

รูปที่ 5.1.ค ไฟล์รหัสต้นฉบับหลังการแก้ไขและเลขแสดงตำแหน่งที่เปลี่ยนแปลงของตัวอย่างที่ 5.1

หลังการแก้ไขเพิ่มเติมข้อบกพร่องภายในขอบเขตของรหัสต้นฉบับที่ได้กำหนดไว้จะพบการเปลี่ยนแปลงในรหัสต้นฉบับตามตำแหน่งต่างๆดังต่อไปนี้

1. การเปลี่ยนแปลงตัวดำเนินการในการเปรียบเทียบความสัมพันธ์และรูปแบบของการแสดงเลขจำนวนเต็มจากข้อกำหนดความผิดพลาด ชนิด ML_ReOp และ ML_InLiMiFo
2. การเปลี่ยนแปลงตัวดำเนินการทางเลขคณิตแบบทวิภาคจากข้อกำหนดความผิดพลาดชนิด ML_ArBiOp
3. การเปลี่ยนแปลงตัวดำเนินการการกำหนดค่าทางเลขคณิต จากข้อกำหนดความผิดพลาดชนิด ML_AsArOp
4. การเปลี่ยนแปลงสลับลำดับของข้อความกำหนดค่าและข้อความอื่น จากข้อกำหนด ML_StSeSw

5.2 การทดสอบการเติมความผิดพลาดลงในโปรแกรม (ตัวอย่างที่ 2)

การทดสอบการเติมความผิดพลาดนี้เป็นตัวอย่างการเติมความผิดพลาดลงในรหัสต้นฉบับจากแบบฝึกหัดสำหรับการฝึกปฏิบัติในการเรียนการสอนวิชาการทำโปรแกรมคอมพิวเตอร์ โดยจะได้แสดงไฟล์ข้อมูลต่างๆ ได้แก่ไฟล์รหัสต้นฉบับก่อนทำการแก้ไขเพิ่มเติมข้อบกพร่อง ประกอบด้วย รหัสต้นฉบับและหมายเหตุประกอบการทำงานของโปรแกรม ไฟล์ข้อกำหนดความผิดพลาด และไฟล์รหัสต้นฉบับหลังจากการแก้ไขเพิ่มเติมข้อบกพร่องตามข้อกำหนด

5.2.1 ไฟล์รหัสต้นฉบับก่อนทำการแก้ไขเพิ่มเติมข้อบกพร่อง รหัสต้นฉบับของโปรแกรมนี้เป็นรหัสต้นฉบับของโปรแกรมโปรแกรมที่ใช้ในการคำนวณจำนวนเวลาเป็น ปี เดือน วัน ชั่วโมง นาที และวินาที จากจำนวนวินาทีที่ได้กำหนด โปรแกรมจะทำงานโดยอาศัยการคำนวณค่าทางเลขคณิตแล้วตอบคำตอบของเวลาที่คำนวณได้ออกมาทางหน้าจอ โดยมีรายละเอียดดังรหัสตัวอย่างในรูปที่ 5.2.ก

```
import java.util.Scanner;

public class Seconds {
  // เขียนโปรแกรมเปลี่ยนจำนวนวินาทีเป็นจำนวนปีเดือนวันชั่วโมง
  // ตัวอย่างผลการทำงานของโปรแกรม
  //      จำนวนวินาที = 1000000
  //      1000000 วินาที = 0 ปี 0 เดือน 11 วัน 13 ชั่วโมง 46 นาที 40 วินาที
   //    หมายเหตุ
  //      - เพื่อความง่ายให้ 1 เดือนมี 30 วัน
  //      - ใช้รูปแบบการแสดงผลดังตัวอย่างข้างบนนี้ อย่าพิมพ์ขาดหรือเกินจากที่กำหนด
  public static void main(String[] args) {
    Scanner kb = new Scanner(System.in);
    System.out.print("จำนวนวินาที = ");
    int second = kb.nextInt();

    int tSec = second;
    int year = tSec / (12 * 30 * 24 * 3600);
    tSec %= (12 * 30 * 24 * 3600);
    int month  = tSec /(30 * 24 * 3600);
    tSec %= (30 * 24 * 3600);
    int day = tSec /(24 * 3600);
    tSec %= (24 * 3600);
    int hr = tSec / 3600;
    tSec %= 3600;
    int min = tSec / 60;
    tSec %= 60;

    System.out.println(second + " วินาที = " + year + " ปี " + month + " เดือน " +
                       day + " วัน " + hr + " ชั่วโมง " + min + " นาที " + tSec + " วินาที");
  }

}
```

รูปที่ 5.2.ก ไฟล์รหัสต้นฉบับซึ่งประกอบด้วย และหมายเหตุประกอบการทำงานของตัวอย่างที่ 5.2

5.2.2 ไฟล์ข้อกำหนดความผิดพลาดที่ใช้ในการเติมข้อบกพร่องลักษณะต่างๆ โดยมีโครงสร้างสมาชิกที่ประกอบด้วย สมาชิก scope ที่ระบุถึงบริเวณที่จะทำการเติมข้อบกพร่องซึ่งระบุเป็นคลาส Seconds ในตำแหน่งเมท็อด main และมีสมาชิก modify เป็นส่วนที่ใช้กำหนดคุณลักษณะของข้อบกพร่องชนิดต่างๆและจำนวนจุดของข้อบกพร่องที่ต้องการเติมลงในรหัสต้นฉบับดังรหัสตัวอย่างในรูปที่ 5.2.ข

```
<modconf>
  <scope className="Seconds" methodName="public static void main(String[] args)">
    <modify modType="ML_ArExMo" modNum="5"/>
    <modify modType="ML_PrCoMo" modNum="3"/>
    <modify modType="ML_PrVaCh" modNum="2"/>
  </scope>
</modconf>
```

รูปที่ 5.2.ข ไฟล์ข้อกำหนดความผิดพลาดของตัวอย่างที่ 5.2

5.2.3 ไฟล์รหัสต้นฉบับหลังการแก้ไขเพิ่มเติมข้อบกพร่องซึ่งจะพบเป็นรหัสต้นฉบับที่เปลี่ยนแปลงไป ดังรหัสตัวอย่างในรูปที่ 5.2.ค

```
import java.util.Scanner;

public class Seconds {
  // เขียนโปรแกรมเปลี่ยนจำนวนวินาทีเป็นจำนวนปีเดือนวันชั่วโมง
  // ตัวอย่างผลการทำงานของโปรแกรม
  //      จำนวนวินาที = 1000000
  //      1000000 วินาที = 0 ปี 0 เดือน 11 วัน 13 ชั่วโมง 46 นาที 40 วินาที
   //    หมายเหตุ
  //      - เพื่อความง่ายให้ 1 เดือนมี 30 วัน
  //      - ใช้รูปแบบการแสดงผลดังตัวอย่างข้างบนนี้ อย่าพิมพ์ขาดหรือเกินจากที่กำหนด

  public static void main(String[] args) {
    Scanner kb = new Scanner(System.in);
    System.out.print("จำนวนวินาที = ");
    int second = kb.nextInt();

    int tSec = second;
    int year = tSec / (12 + 30 * 25 * 3600);
    tSec -= (12 * 30 * 24 * 3600);
    int month  = tSec /-(30 * 24 * 3600);
    month %= (30 * 24 * 3600);
    int day = tSec /(24 * 3600);
    tSec %= (24 + 3600);
    int hr = tSec / 3600;
    tSec %= -3600;
    int min = tSec / 060;
```

1 2 3 4 5 6 7

```
    tSec /= 60;                                                    8

    System.out.println(second + " วินาที = " + year + " ปี " + month + " เดือน " +
                       day + " วัน " + hr + " ชั่วโมง " + day + " นาที " + tSec + " วินาที");
  }                                                                          9
}
```

รูปที่ 5.2.ค ไฟล์รหัสต้นฉบับหลังการแก้ไขและเลขแสดงตำแหน่งที่มีการเปลี่ยนแปลง

หลังการแก้ไขเพิ่มเติมข้อบกพร่องภายในขอบเขตของรหัสต้นฉบับที่ได้กำหนดไว้จะพบการเปลี่ยนแปลงในรหัสต้นฉบับตามตำแหน่งต่างๆดังต่อไปนี้

1. การเปลี่ยนแปลงตัวดำเนินการทางเลขคณิต และ ตัวดำเนินการเปลี่ยนค่าคงที่ของตัวเลขจากข้อกำหนดความผิดพลาด ชนิด ML_ArExMo และ ML_PrCoMo สองจุด
2. การเปลี่ยนแปลงตัวดำเนินการการกำหนดค่าทางเลขคณิต จากข้อกำหนดความผิดพลาดชนิด ML_ArExMo
3. การเปลี่ยนแปลงตัวดำเนินการทางเลขคณิตแบบเอกภาค จากข้อกำหนดความผิดพลาด ML_ArExMo
4. การดำเนินการเปลี่ยนตัวแปรด้วยตัวแปรอื่นที่มีชนิดเดียวกัน จากข้อกำหนดความผิดพลาด ML_PrVaCh
5. การเปลี่ยนแปลงตัวดำเนินการทางเลขคณิตแบบทวิภาค จากข้อกำหนดความผิดพลาด ML_ArExMo
6. การดำเนินการเปลี่ยนค่าคงที่ของตัวเลขจากข้อกำหนดความผิดพลาด ML_PrCoMo
7. การดำเนินการเปลี่ยนแปรรูปแบบของตัวเลขให้อยู่ในระบบเลขฐานแปดจากข้อกำหนดความผิดพลาด ML_PrCoMo
8. การเปลี่ยนแปลงตัวดำเนินการการกำหนดค่าทางเลขคณิต จากข้อกำหนดความผิดพลาดชนิด ML_ArExMo
9. การดำเนินการเปลี่ยนตัวแปรด้วยตัวแปรอื่นที่มีชนิดเดียวกัน จากข้อกำหนดความผิดพลาด ML_PrVaCh

5.3 การทดสอบการเติมความผิดพลาดลงในโปรแกรม (ตัวอย่างที่ 3)

การทดสอบการเติมความผิดพลาดนี้เป็นตัวอย่างการเติมความผิดพลาดลงในรหัสต้นฉบับจากหนังสือ Java Examples in a Nutshell โดยในตัวอย่างนี้จะได้แสดงไฟล์ข้อมูลต่างๆ ได้แก่ไฟล์รหัสต้นฉบับก่อนทำการแก้ไขเพิ่มเติมข้อบกพร่องประกอบด้วย รหัสต้นฉบับและหมาย

เหตุประกอบการทำงานของโปรแกรม ไฟล์ข้อกำหนดความผิดพลาด และไฟล์รหัสต้นฉบับหลังจากการแก้ไขเพิ่มเติมข้อบกพร่อง ตามข้อกำหนด

5.3.1 ไฟล์รหัสต้นฉบับก่อนทำการแก้ไขเพิ่มเติมข้อบกพร่อง ในโปรแกรมนี้เป็นโปรแกรมแสดงผลลัพธ์ของตัวเลขที่ได้จากการคำนวณหาค่าแฟกทอเรียล โดยวิธีการค่าแฟกทอเรียลที่คำนวณได้ใส่ลงในตารางขนาด 21 ช่องเพื่อรองรับการคำนวณค่าแฟกทอเรียลที่ไม่เกิน 20! แล้วทำการตรวจสอบว่าเคยคำนวณค่าแฟกทอเรียลดังกล่าวไปแล้วหรือยังถ้าเคยคำนวณมาแล้วจะนำค่าที่เก็บไว้ในตารางมาตอบ

```
/**
 * This class computes factorials and caches the results in a table for reuse.
 * 20! is as high as we can go using the long data type, so check the argument
 * passed and "throw an exception" if it is too big or too small.
 **/
public class Factorial {
  // Create an array to cache values 0! through 20!.
  static long[] table = new long[21];
  // A "static initializer": initialize the first value in the array
  static { table[0] = 1; }  // factorial of 0 is 1.
  // Remember the highest initialized value in the array
  static int last = 0;
  public static long factorial(int x) throws IllegalArgumentException {
    // Check if x is too big or too small.  Throw an exception if so.
    if (x >= table.length)   // ".length" returns length of any array
      throw new IllegalArgumentException("Overflow; x is too large.");
    if (x < 0) throw new IllegalArgumentException("x must be non-negative.");
    // Compute and cache any values that are not yet cached.
    while(last < x) {
      table[last + 1] = table[last] * (last + 1);
      last++;
    }
    // Now return the cached factorial of x.
    return table[x];
  }
}
```

รูปที่ 5.3.ก ไฟล์รหัสต้นฉบับซึ่งประกอบด้วย และหมายเหตุประกอบการทำงานของตัวอย่างที่ 5.3

5.3.2 ไฟล์ข้อกำหนดความผิดพลาดที่ใช้ในการเติมข้อบกพร่องลักษณะต่างๆ โดยมีโครงสร้างสมาชิกที่ประกอบด้วย สมาชิก scope ที่ระบุถึงบริเวณที่จะทำการเติมข้อบกพร่องซึ่งระบุเป็นคลาส Factorial ในตำแหน่งเมท็อด factorial และมีสมาชิก modify เป็นส่วนใช้ที่กำหนดคุณลักษณะของข้อบกพร่องต่างๆและจำนวนจุดของข้อบกพร่องที่ต้องการเติมลงในรหัสต้นฉบับ ดังรหัสตัวอย่างในรูปที่ 5.3.ข

```
<modconf>
  <scope className="Factorial" methodName="public static int factorial(int x)">
    <modify modType="ML_ArOpMo" modNum="2"/>
    <modify modType="ML_ReOp" modNum="2"/>
    <modify modType="ML_ItWhStCh" modNum="1"/>
    <modify modType="ML_ArAcMo" modNum="2"/>
  </scope>
</modconf>
```

รูปที่ 5.3.ข ไฟล์ข้อกำหนดความผิดพลาดของตัวอย่างที่ 5.3

5.3.3 ไฟล์รหัสต้นฉบับหลังการแก้ไขเพิ่มเติมข้อบกพร่องซึ่งจะพบเป็นรหัสต้นฉบับที่เปลี่ยนแปลงไป ดังรหัสตัวอย่างในรูปที่ 5.3.ค

```
public class Factorial {
  // Create an array to cache values 0! through 20!.
  static long[] table = new long[21];
  // A "static initializer": initialize the first value in the array
  static { table[0] = 1; }  // factorial of 0 is 1.
  // Remember the highest initialized value in the array
  static int last = 0;
  public static long factorial(int x) throws IllegalArgumentException {
    // Check if x is too big or too small.  Throw an exception if so.
    if (x > table.length)   // 1  th" returns length of any array
      throw new IllegalArgumentException("Overflow; x is too large.");
    if (x < 0) throw new IllegalArgumentException("x must be non-negative.");
    // Compute and cache any values that are not yet cached.
    do{                                                          2
      last++;
      table[last - 1] = table[last+1] + (last + 1);                    3
    } while(last <= x)                                   4
    // Now return the cached factorial of x.
    return table[x-1];                                   5
  }
}
```

รูปที่ 5.3.ค ไฟล์รหัสต้นฉบับหลังการแก้ไขและเลขแสดงตำแหน่งที่มีการเปลี่ยนแปลง

หลังการแก้ไขเพิ่มเติมข้อบกพร่องภายในขอบเขตของรหัสต้นฉบับที่ได้กำหนดไว้จะพบการเปลี่ยนแปลงในรหัสต้นฉบับตามตำแหน่งต่างๆดังต่อไปนี้

1. การเปลี่ยนแปลงตัวดำเนินการเปรียบเทียบความสัมพันธ์จากข้อกำหนดความผิดพลาดชนิด ML_ReOp เป็นจุดที่หนึ่งจากทั้งหมดสองจุดที่ต้องการเติม
2. การเปลี่ยนแปลงข้อความการวนซ้ำแบบ while ไปเป็นข้อความวนซ้ำในลักษณะอื่นๆ โดยที่ในตัวอย่างจะเป็นข้อความการวนซ้ำแบบ do…while จากข้อกำหนดความผิดพลาดชนิด ML_ItWhStCh
3. การเปลี่ยนแปลงตัวดำเนินการทางเลขคณิตจำนวนสองจุด และการเปลี่ยนแปลงตำแหน่งในการเข้าถึงข้อมูลของอาร์เรย์ จากข้อกำหนดความผิดพลาดชนิด ML_ArOpMo และ ML_ArAcMo
4. การเปลี่ยนแปลงตัวดำเนินการเปรียบเทียบความสัมพันธ์จากข้อกำหนดความผิดพลาดชนิด ML_ReOp ในจุดที่สองจากทั้งหมดสองจุดที่ต้องการเติม
5. การเปลี่ยนแปลงตำแหน่งในการเข้าถึงข้อมูลของอาร์เรย์ จากข้อกำหนดความผิดพลาดชนิด ML_ArAcMo

5.4 การทดสอบการเติมความผิดพลาดลงในโปรแกรม (ตัวอย่างที่ 4)

การทดสอบการเติมความผิดพลาดนี้เป็นตัวอย่างการเติมความผิดพลาดลงในรหัสต้นฉบับจากแบบฝึกหัดสำหรับการฝึกปฏิบัติในการเรียนการสอนวิชาการทำโปรแกรมคอมพิวเตอร์ โดยในตัวอย่างนี้จะได้แสดงไฟล์ข้อมูลต่างๆ ได้แก่ไฟล์รหัสต้นฉบับก่อนทำการแก้ไขเพิ่มเติมข้อบกพร่องประกอบด้วย รหัสต้นฉบับและหมายเหตุประกอบการทำงานของโปรแกรม ไฟล์ข้อกำหนดความผิดพลาด และไฟล์รหัสต้นฉบับหลังจากการแก้ไขเพิ่มเติมข้อบกพร่อง ตามข้อกำหนด

5.4.1 ไฟล์รหัสต้นฉบับก่อนทำการแก้ไขเพิ่มเติมข้อบกพร่อง รหัสต้นฉบับของโปรแกรมนี้เป็นรหัสต้นฉบับของโปรแกรมโปรแกรมที่ใช้ในการหาเครื่องหมายที่สามารถนำไปประยุกต์ต่อตัวเลขสองตัวแรกที่กำหนดให้แล้วมีค่าเท่ากับตัวเลขจำนวนที่สามที่ได้กำหนดให้ ดังรหัสตัวอย่างในรูปที่ 5.4.ก

```
import java.util.Scanner;

public class SignFinder {

  public static void main(String[] args) {
    // การสร้าง Scanner ข้างล่างนี้ทำให้ผู้ใช้สามารถป้อนจำนวนแต่ละตัว คั่นด้วย , ได้
    Scanner kb = new Scanner(System.in).useDelimiter("\\s*[,\\s]\\s*");
    System.out.print("จำนวนทั้งสาม = ");
    int a = kb.nextInt();
    int b = kb.nextInt();
    int c = kb.nextInt();

    String sign = "";    // ใช้เครื่องหมาย + - x และ /
    if (a + b == c) {
      sign+="+";
    } else if (a - b == c) {
      sign+="-";
    } else if (a * b == c) {
      sign+="x";
    } else if (a / b == c) {
      sign+="/";
    }
    System.out.println("เครื่องหมายที่ต้องการคือ " + sign);
  }
}
```

รูปที่ 5.4.ก ไฟล์รหัสต้นฉบับซึ่งประกอบด้วย และหมายเหตุประกอบการทำงานของตัวอย่างที่ 5.4

5.4.2 ไฟล์ข้อกำหนดความผิดพลาดที่ใช้ในการเติมข้อบกพร่องลักษณะต่างๆ โดยมีโครงสร้างสมาชิกที่ประกอบด้วย สมาชิก scope ที่ระบุถึงบริเวณที่จะทำการเติมซึ่งระบุเป็นคลาส SignFinder ในตำแหน่งเมท็อด main และมีสมาชิก modify เป็นส่วนใช้ที่กำหนดคุณลักษณะของข้อบกพร่องชนิดต่างๆและจำนวนจุดของข้อบกพร่องที่ต้องการเติมลงในรหัสต้นฉบับ ดังรหัสตัวอย่างในรูปที่5.4.ข

```
<modconf>
  <scope className="SignFinder" methodName="public static void main(String[] args)">
    <modify modType="ML_PrInMo" modNum="1"/>
    <modify modType="ML_ReOp" modNum="2"/>
    <modify modType="ML_IfStMo" modNum="1"/>
    <modify modType="ML_PrVaCh" modNum="1"/>
  </scope>
</modconf>
```

รูปที่ 5.4.ข ไฟล์ข้อกำหนดความผิดพลาดของตัวอย่างที่ 5.4.1

5.4.3 ไฟล์รหัสต้นฉบับหลังการแก้ไขเพิ่มเติมข้อบกพร่องซึ่งจะพบเป็นรหัสต้นฉบับที่เปลี่ยนแปลงไปดังรหัสตัวอย่างในรูปที่ 5.4.ค

```
import java.util.Scanner;

public class SignFinder {

  public static void main(String[] args) {
    // การสร้าง Scanner ข้างล่างนี้ทำให้ผู้ใช้สามารถป้อนจำนวนแต่ละตัว คั่นด้วย , ได้
    Scanner kb = new Scanner(System.in).useDelimiter("\\s*[,\\s]\\s*");
    System.out.print("จำนวนทั้งสาม = ");
    int a = kb.nextInt();
    int b;
    int c = kb.nextInt();

    String sign = "";    // ใช้เครื่องหมาย + - x และ /

    if (a + b != c) {
      sign+="+";
    } else if (a - b == c) {
      sign+="-";
    }
    if (a * b >= c) {
      sign+="x";
    } else if (c / b == c) {
      sign+="/";
    }
    System.out.println("เครื่องหมายที่ต้องการคือ " + sign);
  }
}
```

1
2
3
4

รูปที่ 5.4.ค ไฟล์รหัสต้นฉบับหลังการแก้ไขและเลขแสดงตำแหน่งที่มีการเปลี่ยนแปลง

หลังการแก้ไขเพิ่มเติมข้อบกพร่องภายในขอบเขตของรหัสต้นฉบับที่ได้กำหนดไว้จะพบการเปลี่ยนแปลงในรหัสต้นฉบับตามตำแหน่งต่างๆดังต่อไปนี้

1. การตัดการกำหนดค่าให้กับตัวแปรชนิดพื้นฐานจากข้อกำหนดความผิดพลาด ML_PrInMo
2. การเปลี่ยนแปลงตัวดำเนินการทางเปรียบเทียบความสัมพันธ์จากข้อกำหนดความผิดพลาด ML_ReOp
3. การเปลี่ยนแปลงตัวดำเนินการทางเปรียบเทียบความสัมพันธ์ และการเปลี่ยนแปลงประโยคข้อความ if โดยการตัด else ออกจากข้อกำหนดความผิดพลาด ML_ReOp และ ML_IfStMo
4. การเปลี่ยนแปลงตัวแปรเป็นตัวแปรตัวอื่นที่มีชนิดเดียวกันจากข้อกำหนดความผิดพลาด ML_PrVaCh

5.5 การทดสอบการเติมความผิดพลาดลงในโปรแกรม (ตัวอย่างที่ 5)

การทดสอบการเติมความผิดพลาดนี้เป็นตัวอย่างการเติมความผิดพลาดลงในรหัสต้นฉบับจากหนังสือ Java Examples in a Nutshell โดยในตัวอย่างนี้จะได้แสดงไฟล์ข้อมูลต่างๆ ได้แก่ไฟล์รหัสต้นฉบับก่อนทำการแก้ไขเพิ่มเติมข้อบกพร่องประกอบด้วย รหัสต้นฉบับและหมายเหตุประกอบการทำงานของโปรแกรม ไฟล์ข้อกำหนดความผิดพลาด และไฟล์รหัสต้นฉบับหลังจากการแก้ไขเพิ่มเติมข้อบกพร่อง ตามข้อกำหนด

5.5.1 ไฟล์รหัสต้นฉบับก่อนทำการแก้ไขเพิ่มเติมข้อบกพร่อง โปรแกรมนี้เป็นโปรแกรมแสดงการนับเลขโดยจะมีเงื่อนไขว่าเมื่อแสดงตัวเลขถึงเลขที่เป็นเลขที่หารด้วย 5 ลงตัวโปรแกรมจะแสดงคำว่า “fizz” แทน เมื่อแสดงถึงตัวเลขที่หารด้วยเจ็ดลงตัวจะแสดงคำว่า “buzz” แทน และเมื่อแสดงถึงตัวเลขที่หารด้วยห้าและเจ็ดลงตัวแล้วให้แสดงคำว่า “fizzbuzz” โดยจะอาศัยประโยคการตัดสินใจแบบ if…else if เพื่อตัดสินใจว่าตัวเลขที่จะแสดงนั้นอยู่ในกรณีใด

```
/** This program plays the game "Fizzbuzz".  It counts to 100, replacing each
 * multiple of 5 with the word "fizz", each multiple of 7 with the word "buzz",
 * and each multiple of both with the word "fizzbuzz".  It uses the modulo
 * operator (%) to determine if a number is divisible by another.
 **/
public class FizzBuzz {                          // Everything in Java is a class
  public static void main(String[] args) {   // Every program must have main()
    for(int i = 1; i <= 100; i++) {                    // count from 1 to 100
      if (((i % 5) == 0) && ((i % 7) == 0))            // A multiple of both?
        System.out.print("fizzbuzz");
      else if ((i % 5) == 0) System.out.print("fizz"); // else a multiple of 5?
      else if ((i % 7) == 0) System.out.print("buzz"); // else a multiple of 7?
      else System.out.print(i);                        // else just print it
      System.out.print(" ");
    }
    System.out.println();
  }
}
```

รูปที่ 5.5.ก ไฟล์รหัสต้นฉบับซึ่งประกอบด้วย และหมายเหตุประกอบการทำงานของตัวอย่างที่ 5.5

5.4.2 ไฟล์ข้อกำหนดความผิดพลาดที่ใช้ในการเติมข้อบกพร่องลักษณะต่างๆ โดยมีโครงสร้างสมาชิกที่ประกอบด้วย สมาชิก scope ที่ระบุถึงบริเวณที่จะทำการเติมข้อบกพร่องซึ่งระบุเป็นคลาส FizzBuzz ในตำแหน่งเมท็อด main และมีสมาชิก modify เป็นส่วนที่ใช้กำหนดคุณลักษณะของข้อบกพร่องต่างๆและจำนวนจุดของข้อบกพร่องที่ต้องการเติมลงในรหัสต้นฉบับ ดังรหัสตัวอย่างในรูปที่ 5.5.ข

```
<modconf>
  <scope className="FizzBuzz" methodName=”public static void main(String[] args)”>
    <modify modType="ML_ArOpMo" modNum="2"/>
    <modify modType="ML_ReOp" modNum="2"/>
    <modify modType="ML_LoCoBiOp" modNum="1"/>
    <modify modType="ML_LoCoUnOp" modNum="1"/>
    <modify modType="ML_Fi2StCh" modNum="1"/>
    <modify modType="ML_Fi3StCh" modNum="1"/>
  </scope>
</modconf>
```

รูปที่ 5.5.ข ไฟล์ข้อกำหนดความผิดพลาดของตัวอย่างที่ 5.4.1

5.5.3 ไฟล์รหัสต้นฉบับหลังการแก้ไขเพิ่มเติมข้อบกพร่องซึ่งจะพบเป็นรหัสต้นฉบับที่เปลี่ยนแปลงไป ดังรหัสตัวอย่างในรูปที่ 5.5.ค

```
/** This program plays the game "Fizzbuzz".  It counts to 100, replacing each
 *  multiple of 5 with the word "fizz", each multiple of 7 with the word "buzz",
 *  and each multiple of both with the word "fizzbuzz".  It uses the modulo
 *  operator (%) to determine if a number is divisible by another.
 **/
public class FizzBuzz {                    // Everything in Java is a class
  public static void main(String[] args) {  // Every program must have main()
    for(int i = 1; i < 100; i++) {          // count from 1 to 100
      if (!((i % 5) >= 0) || ((i % 7) == 0))  // A multiple of both?
        System.out.print("fizzbuzz");
      else if ((i % 5) == 0) System.out.print("fizz"); // ch    multiple of 5?
      if ((i % 7) == 0)  System.out.print(i); // else    ltiple of 7?
      else System.out.print("buzz");                    // else just print it
      System.out.print(" ");
    }
    System.out.println();
  }
}
```

1 2 3 4

รูปที่ 5.5.ค ไฟล์รหัสต้นฉบับหลังการแก้ไขและเลขแสดงตำแหน่งที่มีการเปลี่ยนแปลง

หลังการแก้ไขเพิ่มเติมข้อบกพร่องภายในขอบเขตของรหัสต้นฉบับที่ได้กำหนดไว้จะพบการเปลี่ยนแปลงในรหัสต้นฉบับตามตำแหน่งต่างๆดังต่อไปนี้

1. การเปลี่ยนแปลงตัวดำเนินการเปรียบเทียบความสัมพันธ์จากข้อกำหนดความผิดพลาดชนิด ML_ReOp เป็นจุดที่หนึ่งจากทั้งหมดสองจุดที่ต้องการเติม
2. การเปลี่ยนแปลงตัวดำเนินการทางเลขคณิต ตัวดำเนินการทางตรรกะแบบทวิภาค ตัวดำเนินการทางตรรกะแบบเอกภาค และตัวดำเนินการเปรียบเทียบความสัมพันธ์ จากข้อกำหนดความผิดพลาดชนิด ML_ArOpMo, ML_LoCoBiOp, ML_LoCoUnOp และ ML_ReOp ชนิดละหนึ่งจุด
3. การเปลี่ยนแปลงข้อความการตัดสินใจแบบ if ในลักษณะที่สามโดยการตัดข้อความ else ออกจากรหัสต้นฉบับ จากข้อกำหนดความผิดพลาดชนิด ML_If3StCh
4. การเปลี่ยนแปลงตัวดำเนินการทางเลขคณิต และการเปลี่ยนแปลงข้อความการตัดสินใจ if ในลักษณะที่สองโดยเป็นการสลับลำดับกลุ่มข้อความภายในประโยค if และประโยค else จากข้อกำหนดความผิดพลาดชนิด ML_ArOpMo และ ML_If3StCh

5.6 การทดสอบการเติมความผิดพลาดลงในโปรแกรม (ตัวอย่างที่ 6)

การทดสอบการเติมความผิดพลาดนี้การทดสอบการเติมความผิดพลาดนี้เป็นตัวอย่างการเติมความผิดพลาดลงในรหัสต้นฉบับจากหนังสือ Java Examples in a Nutshell โดยในตัวอย่างนี้จะได้แสดงไฟล์ข้อมูลต่างๆ ได้แก่ไฟล์รหัสต้นฉบับก่อนทำการแก้ไขเพิ่มเติมข้อบกพร่อง ประกอบด้วย รหัสต้นฉบับและหมายเหตุประกอบการทำงานของโปรแกรม ไฟล์ข้อกำหนดความผิดพลาด และไฟล์รหัสต้นฉบับหลังจากการแก้ไขเพิ่มเติมข้อบกพร่องตามข้อกำหนด

5.6.1 ไฟล์รหัสต้นฉบับก่อนทำการแก้ไขเพิ่มเติมข้อบกพร่อง เช่นเดียวกับโปรแกรมก่อนหน้านี้ในโปรแกรมนี้เป็นโปรแกรมแสดงการนับเลขโดยจะมีเงื่อนไขว่าเมื่อแสดงตัวเลขถึงเลขที่เป็นเลขที่หารด้วย 5 ลงตัวโปรแกรมจะแสดงคำว่า “fizz” แทน เมื่อแสดงถึงตัวเลขที่หารด้วยเจ็ดลงตัวจะแสดงคำว่า “buzz” แทน และเมื่อแสดงถึงตัวเลขที่หารด้วยห้าและเจ็ดลงตัวแล้วให้แสดงคำว่า “fizzbuzz” แต่จะแตกต่างกันที่ไวยากรณ์ของประโยคที่ใช้ในการตัดสินในในโปรแกรมนี้จะใช้ประโยค switch…case ในการตัดสินใจว่าจะเข้าเงื่อนไขของเหตุการณ์ใด

```
/**
 * This class is much like the FizzBuzz class, but uses a switch statement
 * instead of repeated if/else statements
```

```
 **/
public class FizzBuzz2 {
  public static void main(String[] args) {
    for(int i = 1; i <= 100; i++) { // count from 1 to 100
      switch(i % 35) {              // What's the remainder when divided by 35?
      case 0:                               // For multiples of 35...
        System.out.print("fizzbuzz ");     // print "fizzbuzz"
        break;                              // Don't forget this statement!
      case 5: case 10: case 15:             // If the remainder is any of these
      case 20: case 25: case 30:            // then the number is a multiple of 5
        System.out.print("fizz ");         // so print "fizz"
        break;
      case 7: case 14: case 21: case 28:  // For any multiple of 7...
        System.out.print("buzz ");         // print "buzz"
        break;
      default:                              // For any other number...
        System.out.print(i + " ");         // print the number
        break;
      }
    }
    System.out.println();
  }
}
```

รูปที่ 5.6.ก ไฟล์รหัสต้นฉบับซึ่งประกอบด้วย และหมายเหตุประกอบการทำงานของตัวอย่างที่ 5.6

5.6.2 ไฟล์ข้อกำหนดความผิดพลาดที่ใช้ในการเติมข้อบกพร่องลักษณะต่างๆ โดยมีโครงสร้างสมาชิกที่ประกอบด้วย สมาชิก scope ที่ระบุถึงบริเวณที่จะทำการเติมข้อบกพร่อง และสมาชิก modify เป็นส่วนใช้ที่กำหนดคุณลักษณะของข้อบกพร่องต่างๆและจำนวนจุดของข้อบกพร่องที่ต้องการเติมลงในรหัสต้นฉบับ

```
<modconf>
  <scope className="FizzBuzz2" methodName=”public static void main(String[] args)”>
    <modify modType="ML_InLiMiFo" modNum="1"/>
    <modify modType="ML_ReOp" modNum="1"/>
    <modify modType="ML_SwCaSw" modNum="1"/>
    <modify modType="ML_SwCaDe" modNum="1"/>
    <modify modType="ML_SwBrDe" modNum="1"/>
    <modify modType="ML_ItFoStCh" modNum="1"/>
  </scope>
</modconf>
```

รูปที่ 5.6.ข ไฟล์ข้อกำหนดความผิดพลาดของตัวอย่างที่ 5.6.1

5.6.3 ไฟล์รหัสต้นฉบับหลังการแก้ไขเพิ่มเติมข้อบกพร่องซึ่งจะพบเป็นรหัสต้นฉบับที่เปลี่ยนแปลงไป ดังรหัสตัวอย่างในรูปที่ 5.6.ค

```
public class FizzBuzz2 {
  public static void main(String[] args) {
    int i = 1;
    do{                                                    1
      i++;
      switch(i % 035) {                                    2  // What's the remainder when divided by 35?
      case 10:                                             3   // For multiples of 35...
        System.out.print("fizzbuzz ");                       // print "fizzbuzz"
        break;                                               // Don't forget this statement!
      case 5: case 0: case 15:                              // If the remainder is any of these
      case 20: case 25: case 30:                            // then the number is a multiple of 5
        System.out.print("fizz ");                          4  print "fizz"
      case 7: case 21: case 28:                             5  any multiple of 7...
        System.out.print("buzz ");                          // print "buzz"
        break;
      default:                                              // For any other number...
        System.out.print(i + " ");                          // print the number
        break;
      }
    }while(i == 100;)                                                6
    System.out.println();
  }
}
```

รูปที่ 5.6.ค ไฟล์รหัสต้นฉบับหลังการแก้ไขและเลขแสดงตำแหน่งที่มีการเปลี่ยนแปลง

หลังการแก้ไขเพิ่มเติมข้อบกพร่องภายในขอบเขตของรหัสต้นฉบับที่ได้กำหนดไว้จะพบการเปลี่ยนแปลงในรหัสต้นฉบับตามตำแหน่งต่างๆดังต่อไปนี้

1. การเปลี่ยนแปลงข้อความการวนซ้ำแบบ for เป็นวงวนลักษณะอื่นๆโดยในการเปลี่ยนแปลงนี้ถูกเปลี่ยนเป็นข้อความการวนซ้ำแบบ do…while จากจากข้อกำหนดความผิดพลาดชนิด ML_ItFoStCh
2. การเปลี่ยนแปลงรูปแบบการแสดงของตัวเลขจากข้อกำหนดความผิดพลาดชนิด ML_InLiFoCh
3. การเปลี่ยนแปลงในข้อความการตัดสินใจแบบ switch…case โดยการสลับกรณีในการตัดสินใจของข้อความ จากข้อกำหนดความผิดพลาดชนิด ML_SwCaSw

4. การเปลี่ยนแปลงในข้อความการตัดสินใจแบบ switch…case โดยการตัดข้อความ break ออก จากข้อกำหนดความผิดพลาดชนิด ML_SwBrDe
5. การเปลี่ยนแปลงในข้อความการตัดสินใจแบบ switch…case โดยการตัดกรณีในการทำงานบางกรณีออกไป ข้อกำหนดความผิดพลาดชนิด ML_SwCaDe
6. การเปลี่ยนแปลงตัวดำเนินการเปรียบเทียบความสัมพันธ์จากข้อกำหนดความผิดพลาดชนิด ML_ReOp

5.7 การทดสอบการเติมความผิดพลาดลงในโปรแกรม (ตัวอย่างที่ 7)

การทดสอบการเติมความผิดพลาดนี้เป็นตัวอย่างการเติมความผิดพลาดลงในรหัสต้นฉบับจากแบบฝึกหัดสำหรับการฝึกปฏิบัติในการเรียนการสอนวิชาการทำโปรแกรมคอมพิวเตอร์ โดยในตัวอย่างนี้จะได้แสดงไฟล์ข้อมูลต่างๆ ได้แก่ไฟล์รหัสต้นฉบับก่อนทำการแก้ไขเพิ่มเติมข้อบกพร่องประกอบด้วย รหัสต้นฉบับและหมายเหตุประกอบการทำงานของโปรแกรม ไฟล์ข้อกำหนดความผิดพลาด และไฟล์รหัสต้นฉบับหลังจากการแก้ไขเพิ่มเติมข้อบกพร่องตามข้อกำหนด

5.7.1 ไฟล์รหัสต้นฉบับก่อนทำการแก้ไขเพิ่มเติมข้อบกพร่อง ไฟล์รหัสต้นฉบับของโปรแกรมภาษาจาวาโปรแกรมนี้เป็นโปรแกรมสำหรับเปลี่ยนคำเอกพจน์ในภาษาอังกฤษให้เป็นคำพหูพจน์โดยการเติม s, es หรือเปลี่ยนตัวอักษร y ที่ลงท้ายเป็น ies แล้วแต่กรณีของคำ โดยในโปรแกรมจะทำการแยกกรณีต่างๆออกจากกันโดยอาศัยประโยค switch ในการตัดสินใจซึ่งมีรายละเอียดดังรหัสตัวอย่างในรูปที่ 5.7.ก

```
import java.util.Scanner;

public class Plural {
  public static void main(String[] args) {
    String sNoun, pNoun;
    Scanner kb = new Scanner(System.in);
    System.out.print("noun = ");
    sNoun = kb.nextLine();

    pNoun=sNoun;
    switch (sNoun.charAt(sNoun.length()-1)){
    case 'c':
    case 's':
    case 'x':
      pNoun+="es";
      break;
    case 'y':
      if((sNoun.charAt(sNoun.length()-2)!='a')&&
```

```
          (sNoun.charAt(sNoun.length()-2)!='e')&&
          (sNoun.charAt(sNoun.length()-2)!='i')&&
          (sNoun.charAt(sNoun.length()-2)!='o')&&
          (sNoun.charAt(sNoun.length()-2)!='u')
          ){
        char[] pNoun2 = pNoun.toCharArray();
        pNoun2[pNoun2.length-1]='i';
        pNoun=new String(pNoun2);
        pNoun+="es";
      }
      break;
    default:
      pNoun+="s";
      break;
    }

    System.out.println(pNoun);
  }
}
```

รูปที่ 5.7.ก ไฟล์รหัสต้นฉบับซึ่งประกอบด้วย และหมายเหตุประกอบการทำงานของตัวอย่างที่ 5.7

5.7.2 ไฟล์ข้อกำหนดความผิดพลาดที่ใช้ในการเติมข้อบกพร่องลักษณะต่างๆ โดยมีโครงสร้างสมาชิกที่ประกอบด้วย สมาชิก scope ที่ระบุถึงบริเวณที่จะทำการเติมข้อบกพร่อง และสมาชิก modify เป็นส่วนใช้ที่กำหนดคุณลักษณะของข้อบกพร่องต่างๆและจำนวนจุดของข้อบกพร่องที่ต้องการเติมลงในรหัสต้นฉบับ ดังรายละเอียดตัวอย่างในรูปที่ 5.7.ข

```
<modconf>
  <scope className="Plural" methodName=”public static void main(String[] args)”>
    <modify modType="ML_PrCoMo" modNum="1"/>
    <modify modType="ML_ReOp" modNum="2"/>
    <modify modType="ML_DeMo" modNum="3"/>
    <modify modType="SP_MiCaLeMe" modNum="1"/>
  </scope>
</modconf>
```

รูปที่ 5.7.ข ไฟล์ข้อกำหนดความผิดพลาด

5.7.3 ไฟล์รหัสต้นฉบับหลังการแก้ไขเพิ่มเติมข้อบกพร่องซึ่งจะพบเป็นรหัสต้นฉบับที่เปลี่ยนแปลงไป ดังรหัสตัวอย่างในรูปที่ 5.7.ค

```
import java.util.Scanner;

public class Plural {
  public static void main(String[] args) {
    String sNoun, pNoun;
    Scanner kb = new Scanner(System.in);
    System.out.print("noun = ");
    sNoun = kb.nextLine();
```

```
    pNoun=sNoun;
    switch (sNoun.charAt(sNoun.length()-1)){
    case 'c':
    case 'x':
      pNoun+="es";
    case 'y':
      if((sNoun.charAt(sNoun.length()-2)!='b')||
         (sNoun.charAt(sNoun.length()-2)!='e')&&
         (sNoun.charAt(sNoun.length()-2)!='i')&&
         (sNoun.charAt(sNoun.length-2)!='o')^
         (sNoun.charAt(sNoun.length()-2)!='u')
         )
        char[] pNoun2 = pNoun.toCharArray();
      pNoun2[pNoun2.length-1]='i';
      pNoun=new String(pNoun2);
      pNoun+="es";
      break;
    default:
      pNoun+="s";
      break;
    }

    System.out.println(pNoun);
  }
}
```

1 2 3 4

รูปที่ 5.7.ค ไฟล์รหัสต้นฉบับหลังการแก้ไขและเลขแสดงตำแหน่งที่มีการเปลี่ยนแปลง

หลังการแก้ไขเพิ่มเติมข้อบกพร่องภายในขอบเขตของรหัสต้นฉบับที่ได้กำหนดไว้จะพบการเปลี่ยนแปลงในรหัสต้นฉบับตามตำแหน่งต่างๆดังต่อไปนี้

1. การเปลี่ยนแปลงโครงสร้างของข้อความ switch จากการตัดกรณีเงื่อนไขและการตัด break ออกจากข้อกำหนดความผิดพลาด ML_DeMo
2. การเปลี่ยนแปลงรูปแบบการแสดงของตัวอักษร และการเปลี่ยนเครื่องหมายตัวดำเนินการทางตรรกะจากข้อกำหนดความผิดพลาดชนิด ML_PrVaCh, ML_ReOp
3. การเปลี่ยนเครื่องหมายตัวดำเนินการทางตรรกะ และการเปลี่ยนแปลงการเรียกใช้เมท็อด length เป็นการเรียกใช้ข้อมูล จากข้อกำหนดความผิดพลาดชนิด ML_ReOpและ SP_MiCaLeMe
4. การเปลี่ยนแปลงโครงสร้างของข้อความ if จากการตัดวงเล็บปีกกาออก จากข้อกำหนดความผิดพลาด ML_DeMo

5.8 การทดสอบการเติมความผิดพลาดลงในโปรแกรม (ตัวอย่างที่ 8)

การทดสอบการเติมความผิดพลาดนี้เป็นตัวอย่างการเติมความผิดพลาดลงในรหัสต้นฉบับจากหนังสือ Java Examples in a Nutshell โดยในตัวอย่างนี้จะได้แสดงไฟล์ข้อมูลต่างๆ ได้แก่ไฟล์รหัสต้นฉบับก่อนทำการแก้ไขเพิ่มเติมข้อบกพร่องประกอบด้วย รหัสต้นฉบับและหมายเหตุประกอบการทำงานของโปรแกรม ไฟล์ข้อกำหนดความผิดพลาด และไฟล์รหัสต้นฉบับหลังจากการแก้ไขเพิ่มเติมข้อบกพร่องตามข้อกำหนด

5.8.1 ไฟล์รหัสต้นฉบับก่อนทำการแก้ไขเพิ่มเติมข้อบกพร่อง โปรแกรมนี้เป็นโปรแกรมที่ใช้ในการคำนวณเลขจำนวนเฉพาะโดยใช้ขั้นตอนวิธีของ Sieve of Eratosthenes ซึ่งอาศัยหลักการว่าเลขจำนวนเต็มใดที่มีค่าเป็นจำนวนเท่าของเลขจำนวนเต็มที่มีค่าน้อยกว่าแล้วจำนวนนั้นไม่ใช่จำนวนเฉพาะ และส่วนที่เหลือคือจำนวนเฉพาะ โดยโปรแกรมนี้จะแสดงจำนวนเฉพาะที่มากที่สุดที่มีค่าน้อยกว่า 100 หรือตามที่กำหนดออกมาหลังจากประมวลผลเพื่อหาจำนวนเฉพาะเสร็จ

```
/**
 * This program computes prime numbers using the Sieve of Eratosthenes
 * algorithm: rule out multiples of all lower prime numbers, and anything
 * remaining is a prime.  It prints out the largest prime number less than
 * or equal to the supplied command-line argument
 **/
public class Sieve {
  public static void main(String[] args) {
    // We will compute all primes less than the supplied command line argument
    // Or, if no argument, all primes less than 100
    int max = 100;                          // Assign a default value
    try { max = Integer.parseInt(args[0]); } // Try to parse user-supplied arg
    catch (Exception e) {}                  // Silently ignore exceptions.
    // Create an array that specifies whether each number is prime or not.
    boolean[] isprime = new boolean[max+1];
    // Assume that all numbers are primes, until proven otherwise.
    for(int i = 0; i <= max; i++) isprime[i] = true;
    // However, we know that 0 and 1 are not primes.  Make a note of it.
    isprime[0] = isprime[1] = false;
    // To compute all primes less than max, we need to rule out
    // multiples of all integers less than the square root of max.
    int n = (int) Math.ceil(Math.sqrt(max));  // See java.lang.Math class
    // Now, for each integer i from 0 to n:
```

```
        //   If i is a prime, then none of its multiples are primes, so
        //   indicate this in the array.
        //   If i is not a prime, then its multiples have already been
        //   ruled out by one of the prime factors of i, so we can skip this case.
        for(int i = 0; i <= n; i++) {
          if (isprime[i])                          // If i is a prime,
            for(int j = 2*i; j <= max; j = j + i)  // loop through its multiples
              isprime[j] = false;                  // noting they are not prime.
        }
        // Now go look for the largest prime:
        int largest;
        for(largest = max; !isprime[largest]; largest--) ;  // empty loop body
        // Output the result
        System.out.println("The largest prime less than or equal to " + max +
                           " is " + largest);
      }
    }
```

รูปที่ 5.8.ก ไฟล์รหัสต้นฉบับซึ่งประกอบด้วย และหมายเหตุประกอบการทำงานของตัวอย่างที่ 5.8

5.8.2 ไฟล์ข้อกำหนดความผิดพลาดที่ใช้ในการเติมข้อบกพร่องลักษณะต่างๆ โดยมีโครงสร้างสมาชิกที่ประกอบด้วย สมาชิก scope ที่ระบุถึงบริเวณที่จะทำการเติมข้อบกพร่อง และสมาชิก modify เป็นส่วนใช้ที่กำหนดคุณลักษณะของข้อบกพร่องต่างๆและจำนวนจุดของข้อบกพร่องที่ต้องการเติมลงในรหัสต้นฉบับ ดังรายละเอียดในตัวอย่าง 5.8.ข

```
<modconf>
  <scope className="Sieve" methodName=”public static void main(String[] args)”>
    <modify modType="ML_PrCaDe" modNum="1"/>
    <modify modType="ML_ItFoStCh" modNum="2"/>
    <modify modType="ML_AsOp" modNum="1"/>
    <modify modType="ML_ArOpMo" modNum="3"/>
    <modify modType="ML_LoCoOp" modNum="1"/>
  </scope>
</modconf>
```

รูปที่ 5.8.ข ไฟล์ข้อกำหนดความผิดพลาด

5.8.3 ไฟล์รหัสต้นฉบับหลังการแก้ไขเพิ่มเติมข้อบกพร่องซึ่งจะพบเป็นรหัสต้นฉบับที่เปลี่ยนแปลงไป ดังรูปที่ 5.8.ค

```
/**
 * This program computes prime numbers using the Sieve of Eratosthenes
 * algorithm: rule out multiples of all lower prime numbers, and anything
 * remaining is a prime.  It prints out the largest prime number less than
 * or equal to the supplied command-line argument
 **/
public class Sieve {
  public static void main(String[] args) {
    // We will compute all primes less than the supplied command line argument
    // Or, if no argument, all primes less than 100
    int max = 100;                          // Assign a default value
    try { max = Integer.parseInt(args[0]); } // Try to parse user-supplied arg
    catch (Exception e) {}                  // Silently ignore exceptions.
    // Create an array that specifies whether each number is prime or not.
    boolean[] isprime = new boolean[max+1];
    // Assume that all numbers are primes, until proven otherwise.
    int i = 0;
    while(i <= max){
      isprime[i] = true;
      --i;
    }
    // However, we know that 0 and 1 are not primes.  Make a note of it.
    isprime[0] = isprime[1] = false;
    // To compute all primes less than max, we need to rule out
    // multiples of all integers less than the square root of max.
    int n = Math.ceil(Math.sqrt(max));  // See ang.Math class
    // Now, for each integer i from 0 to n:
    //   If i is a prime, then none of its multiples are primes, so
    //   indicate this in the array.
    //   If i is not a prime, then its multiples have already been
    //   ruled out by one of the prime factors of i, so we can skip this case.
    for(int i = 0; i <= n; i++) {
      if (isprime[i])                       // If i is a prime,
        for(int j = 2+i; j <= max; j = j * i)  // loop gh its multiples
          isprime[j] |= false;              // noting they are not prime.
    }
    // Now go look for the largest prime:
    int largest;
    largest = max;
    do{
      largest--;
    }while(isprime[largest]) // empty loop y
    // Output the result
```

```
        System.out.println("The largest prime less than or equal to " + max +
                           " is " + largest);
    }
}
```

รูปที่ 5.8.ค ไฟล์รหัสต้นฉบับหลังการแก้ไขและเลขแสดงตำแหน่งที่มีการเปลี่ยนแปลง

หลังการแก้ไขเพิ่มเติมข้อบกพร่องภายในขอบเขตของรหัสต้นฉบับที่ได้กำหนดไว้จะพบการเปลี่ยนแปลงในรหัสต้นฉบับตามตำแหน่งต่างๆดังต่อไปนี้

1. การเปลี่ยนแปลงข้อความการวนซ้ำแบบ for เป็นวงวนลักษณะอื่นๆโดยในการเปลี่ยนแปลงนี้ถูกเปลี่ยนเป็นข้อความการวนซ้ำแบบ while จากจากข้อกำหนดความผิดพลาดชนิด ML_ItFoStCh
2. การเปลี่ยนแปลงตัวดำเนินการทางเลขคณิตจำนวนหนึ่งจุดจากข้อกำหนดความผิดพลาดชนิด ML_ArOpMo;
3. การเปลี่ยนแปลงจากการลบข้อความการเปลี่ยนชนิดของข้อมูลพื้นฐานออกจากรหัสต้นฉบับ จากข้อกำหนดความผิดพลาดชนิด ML_PrCaDe
4. การเปลี่ยนแปลงตัวดำเนินการทางเลขคณิตจำนวนสองจุดจากข้อกำหนดความผิดพลาดชนิด ML_ArOpMo;
5. การเปลี่ยนแปลงตัวดำเนินการการกำหนดค่า จากข้อกำหนดความผิดพลาดชนิด ML_AsOpMo
6. การเปลี่ยนแปลงข้อความการวนซ้ำแบบ for เป็นวงวนลักษณะอื่นๆโดยในการเปลี่ยนแปลงนี้ถูกเปลี่ยนเป็นข้อความการวนซ้ำแบบ do…while จากข้อกำหนดความผิดพลาดชนิด ML_ItFoStCh
7. การเปลี่ยนแปลงตัวดำเนินการตรรกะ จากข้อกำหนดความผิดพลาดชนิด ML_LoCoOp

5.9 การทดสอบการเติมความผิดพลาดลงในโปรแกรม (ตัวอย่างที่ 9)

การทดสอบการเติมความผิดพลาดนี้เป็นตัวอย่างการเติมความผิดพลาดลงในรหัสต้นฉบับจากแบบฝึกหัดสำหรับการฝึกปฏิบัติในการเรียนการสอนวิชาการทำโปรแกรมคอมพิวเตอร์ โดยในตัวอย่างนี้จะได้แสดงไฟล์ข้อมูลต่างๆ ได้แก่ไฟล์รหัสต้นฉบับก่อนทำการแก้ไขเพิ่มเติมข้อบกพร่องประกอบด้วย รหัสต้นฉบับและหมายเหตุประกอบการทำงานของโปรแกรม ไฟล์

ข้อกำหนดความผิดพลาด และไฟล์รหัสต้นฉบับหลังจากการแก้ไขเพิ่มเติมข้อบกพร่องตามข้อกำหนด

5.9.1 ไฟล์รหัสต้นฉบับก่อนทำการแก้ไขเพิ่มเติมข้อบกพร่อง ไฟล์รหัสต้นฉบับของโปรแกรมภาษาจาวาโปรแกรมนี้เป็นโปรแกรมสำหรับใช้ในการตรวจสอบตาราง Sudoku ว่าตารางที่ได้รับมานั้นถูกต้องตามกฎหรือไม่ กล่าวคือในแถว หลัก หรือตารางย่อยของตาราง Sudoku จะต้องไม่มีตัวเลขใดเลยที่ซ้ำกัน ซึ่งมีรายละเอียดดังรหัสตัวอย่างในรูปที่ 5.9.ก

```
import java.util.Arrays;

public class Sudoku {
  public static void main(String[] a) {
    int[][] t = {{ 9, 6, 3, 1, 7, 4, 2, 5, 8 },
                 { 1, 7, 8, 3, 2, 5, 6, 4, 9 },
                 { 2, 5, 4, 6, 8, 9, 7, 3, 1 },
                 { 8, 2, 1, 4, 3, 7, 5, 9, 6 },
                 { 4, 9, 6, 8, 5, 2, 3, 1, 7 },
                 { 7, 3, 5, 9, 6, 1, 8, 2, 4 },
                 { 5, 8, 9, 7, 1, 3, 4, 6, 2 },
                 { 3, 1, 7, 2, 4, 6, 9, 8, 5 },
                 { 6, 4, 2, 5, 9, 8, 1, 7, 3 }};

    System.out.println(isSudoku(t));
  }
  public static boolean isSudoku(int[][] table) {
   if(checkRow(table)&& checkColumn(table)&& checkGrid(table)){
       return true;
    }
    return false;
  }
  // add any additional methods here

  public static boolean isRepeat(int n, int[] check) {
    if(check[n]==0){
      return true;
    }else{
      check[n] = 0;
      return false;
    }
  }

  public static int[] constructChecker(){
    int[] check=new int[10];
    for(int i=0;i<10;i++){
      check[i]=i;
    }
    return check;
  }

  public static boolean checkRow(int[][] arr){
```

```
    for(int i=0;i<9;i++){
      int[] checker=constructChecker();
      for(int j=0;j<9;j++){
        if(isRepeat(arr[i][j],checker)){
          return false;
        }
      }
    }
    return true;
  }

  public static boolean checkColumn(int[][] arr){
    for(int i=0;i<9;i++){
      int[] checker=constructChecker();
      for(int j=0;j<9;j++){
        if(isRepeat(arr[j][i],checker)){
          return false;
        }
      }
    }
    return true;
  }

  public static boolean checkGrid(int[][] arr){
    int[] checker;
    for(int row=0;row<7;row+=3){
      for(int col=0;col<7;col+=3){
        checker=constructChecker();
        for(int i=0;i<3;i++){
          for(int j=0;j<3;j++){
            if(isRepeat(arr[row+i][col+j],checker)){
              return false;
            }
          }
        }
      }
    }
    return true;
  }
}
```

รูปที่ 5.9.ก ไฟล์รหัสต้นฉบับซึ่งประกอบด้วย และหมายเหตุประกอบการทำงานของตัวอย่างที่ 5.9

5.9.2 ไฟล์ข้อกำหนดความผิดพลาดที่ใช้ในการเติมข้อบกพร่องลักษณะต่างๆ โดยมีโครงสร้างสมาชิกที่ประกอบด้วย สมาชิก scope ที่ระบุถึงบริเวณที่จะทำการเติมข้อบกพร่อง และสมาชิก modify เป็นส่วนใช้ที่กำหนดคุณลักษณะของข้อบกพร่องต่างๆและจำนวนจุดของข้อบกพร่องที่ต้องการเติมลงในรหัสต้นฉบับ

```
<modconf>
  <scope className="Sudoku">
    <modify modType="ML" modNum="15"/>
  </scope>
</modconf>
```

รูปที่ 5.9.ข ไฟล์ข้อกำหนดความผิดพลาด

5.9.3 ไฟล์รหัสต้นฉบับหลังการแก้ไขเพิ่มเติมข้อบกพร่องซึ่งจะพบเป็นรหัสต้นฉบับที่เปลี่ยนแปลงไป ดังรูปที่ 5.9.ค

```
import java.util.Arrays;

public class Sudoku {
  public static void main(String[] a) {
    int[][] t = {{ 9, 6, 3, 1, 7, 4, 2, 5, 8 },
                 { 1, 7, 8, 3, 2, 5, 6, 4, 9 },
                 { 2, 5, 4, 6, 8, 9, 7, 3, 1 },
                 { 8, 2, 1, 4, 3, 7, 5, 9, 6 },
                 { 4, 9, 6, 8, 5, 2, 3, 1, 7 },
                 { 7, 3, 5, 9, 6, 1, 8, 2, 4 },
                 { 5, 8, 9, 7, 1, 3, 4, 6, 2 },
                 { 3, 1, 7, 2, 4, 6, 9, 8, 5 },
                 { 6, 4, 2, 5, 9, 8, 1, 7, 3 }};

    System.out.println(isSudoku(t));
  }
  public static boolean isSudoku(int[][] table) {
    if(checkRow(table)|| checkColumn(table)&& checkGrid(table)){    }
    return false;
  }
  // add any additional methods here

  public static boolean isRepeat(int[] check, int n) {
    if(check[n]==0){
      return true;
    }else{
      check[n] = 1;
      return false;
    }
  }

  public static int[] constructChecker(){
    int[] check=new int[10];
    for(int i=0;i>=10;){
      check[i]=i;
    }
    return check;
  }

  public static boolean checkRow(int[][] arr){
    for(int i=0;i<9;--i){
```

1

2

3

4

5

```
      int[] checker=constructChecker();
      for(int j=0;j<9;j++){
        if(isRepeat(checker, arr[i][j])){
          return false;
        }
      }
    }
    return true;
  }

  public static boolean checkColumn(int[][] arr){
    for(int i=0;i<9;i++){
      int[] checker=constructChecker();
      int j=0;
      do{
         j++;
         if(isRepeat(arr[i][j],checker)){
          return false;
         }
      }while(j<9)
    }
    return true;
  }

  public static boolean checkGrid(int[][] arr){
    int[] checker;
    int row=0;
    while(row<7){
      row+=3;
      for(int col=0;col<7;col+=3){
        constructChecker();
        for(int i=0;i<3;i++){
          for(int j=0;j<3;j++){
            if(isRepeat(arr[row-i][col+i],checker)){
              return flase;
            }
          }
        }
      }
    }
    return false;
  }
}
```

6 7 8 9 10 11 12

รูปที่ 5.9.ค ไฟล์รหัสต้นฉบับหลังการแก้ไขและเลขแสดงตำแหน่งที่มีการเปลี่ยนแปลง

หลังการแก้ไขเพิ่มเติมข้อบกพร่องภายในขอบเขตของรหัสต้นฉบับที่ได้กำหนดไว้จะพบการเปลี่ยนแปลงในรหัสต้นฉบับตามตำแหน่งต่างๆดังต่อไปนี้

1. การเปลี่ยนแปลงตัวดำเนินการทางตรรกะและการเปลี่ยนแปลงจากการตัดข้อความการคืนค่าออก
2. การเปลี่ยนแปลงโครงสร้างของเมท็อดจากการสลับพารามิเตอร์
3. การเปลี่ยนแปลงค่าคงที่ให้มีค่าผิดไปเล็กน้อย
4. การเปลี่ยนแปลงตัวดำเนินการเปรียบเทียบความสัมพันธ์และการลบนิพจน์การเพิ่มค่าของข้อความการวนซ้ำออก
5. การเปลี่ยนแปลงตัวดำเนินการทางเลขคณิตอย่างย่อ
6. การสลับลำดับของอาร์กิวเมนต์ในการเรียกใช้งานเมท็อด
7. การเปลี่ยนแปลงโครงของข้อความวนซ้ำจากการวนซ้ำแบบ for เป็นแบบ do…while
8. การสลับการเรียกตำแหน่งในการเข้าถึงของอาร์เรย์หลายมิติ
9. การเปลี่ยนแปลงโครงของข้อความวนซ้ำจากการวนซ้ำแบบ for เป็นแบบ while
10. การเปลี่ยนแปลงจากการลบตัวแปรออกจากการกำหนดค่าตัวแปรด้วยค่าที่ได้รับจากเมท็อดให้เป็นการเรียกใช้งานเมท็อดลอยๆ
11. การเปลี่ยนแปลงตัวดำเนินการทางเลขคณิตแบบทวิภาคและการเปลี่ยนตัวแปรด้วยตัวแปรอื่นที่มีชนิดของตัวแปรเหมือนกัน
12. การแก้ไขเมท็อดการคืนค่าให้เป็นค่ามาตรฐาน

5.10 สรุปผลการทดสอบการทำงาน

ในการเติมความผิดพลาดลงในรหัสต้นฉบับผู้วิจัยได้ทำการทดสอบและรวบรวมข้อมูลตำแหน่งที่มีความเหมาะสมในการแก้ไขที่ถูกตรวจพบโดย Visitor และผลจากการแก้ไขเพื่อให้เกิดความผิดพลาดในแต่ละตำแหน่ง โดยจะทำการทดสอบความผิดพลาดที่เกิดขึ้นทุกครั้งหลังจากการแก้ไขในแต่ละตำแหน่งของรหัสต้นฉบับที่ได้เสนอมาในตัวอย่างการทดสอบการทำงาน ซึ่งลักษณะความผิดพลาดที่ได้รวบรวมนี้จะเป็นเฉพาะชนิดความผิดพลาดทั่วไปซึ่งผลการทำงานที่ได้นั้นเป็นดังตารางที่ 5.1

ตารางที่ 5.1 ตารางสรุปผลการเติมความผิดพลาดทั่วไปลงในตัวอย่าง

ตัวอย่างที่	ชนิดข้อบกพร่อง	ตำแหน่งเหมาะสมที่พบ	จำนวนที่ทดสอบพบข้อบกพร่อง
ตัวอย่างที่ 1	ML_ReOp	1	1

ตารางที่ 5.1 ตารางสรุปผลการเติมความผิดพลาดทั่วไปลงในตัวอย่าง(ต่อ)

ตัวอย่างที่	ชนิดข้อบกพร่อง	ตำแหน่งเหมาะสมที่พบ	จำนวนที่ทดสอบพบข้อบกพร่อง
ตัวอย่างที่ 1	ML_InLiMiFo	4	1
	ML_ArBiOp	1	1
	ML_AsArOp	3	3
	ML_StSeSw	1	1
ตัวอย่างที่ 2	ML_PrVaCh	17	16
ตัวอย่างที่ 3	ML_ReOp	3	3
	ML_ItWhSt	1	1
	ML_ArAcCh	3	3
ตัวอย่างที่ 4	ML_ReOp	4	4
	ML_PrVaCh	7	7
	ML_If2StCh	3	3
ตัวอย่างที่ 5	ML_ReOp	5	5
	ML_LoCoBi	1	1
	ML_LoCoUn	5	5
	ML_If2StCh	2	2
	ML_If3StCh	3	3
ตัวอย่างที่ 6	ML_InLiMiFo	13	10
	ML_SwCaSw	10	8
	ML_SwCaDe	11	11
	ML_SwBrDe	4	3
	ML_ItFoCh	1	1

ตารางที่ 5.1 ตารางสรุปผลการเติมความผิดพลาดทั่วไปลงในตัวอย่าง(ต่อ)

ตัวอย่างที่	ชนิดข้อบกพร่อง	ตำแหน่งเหมาะสมที่พบ	จำนวนที่ทดสอบพบข้อบกพร่อง
ตัวอย่างที่ 7	SP_MiCaLeMe	6	6
ตัวอย่างที่ 8	ML_PrCaDe ML_ItFoStCh	1 4	1 4
ตัวอย่างที่ 9	ไม่ได้เติมลักษณะข้อบกพร่องทั่วไป (เติมแต่ลักษณะกลุ่ม)	-	-

จากข้อมูลในตาราง 5.1ทำให้ทราบว่าในโจทย์ปัญหาอย่างง่ายทั่วไปนั้นในการแก้ไขเพื่อเติมความผิดพลาดเนื่องจากการเติมข้อบกพร่องลงในแต่ละตำแหน่งของรหัสต้นฉบับที่มีโครงสร้างการทำงานเหมาะสมที่ตรวจพบจากขั้นตอนการค้นหาโครงสร้างนั้นสามารถใช้โปรแกรมการตรวจสอบการทำงานทั่วไปก็สามารถตรวจพบความผิดพลาดที่เกิดขึ้นได้ โดยมีเพียงส่วนน้อยเท่านั้นที่ไม่สามารถตรวจสอบความผิดพลาดที่เกิดขึ้นได้

บทที่ 6

สรุปผลการวิจัย และข้อเสนอแนะ

สรุปผลการวิจัย

สำหรับงานวิจัยนี้ผู้วิจัยได้ทำการออกแบบและพัฒนาซอฟต์แวร์โปรแกรมการเติมข้อบกพร่องอัตโนมัติจากแนวความคิดการกลายรหัสต้นฉบับของโปรแกรมภาษาจาวาโดยอาศัยการกลายโครงสร้างต้นไม้ของรหัสต้นฉบับเพื่อให้ได้โครงสร้างของต้นไม้ที่มีการทำงานที่ผิดพลาดเชิงความหมาย โปรแกรมจะนำรหัสต้นฉบับที่ได้รับมาทำการแปลงให้อยู่ในรูปแบบของโครงสร้างต้นไม้ไวยากรณ์เชิงนามธรรม และใช้ข้อมูลจากไฟล์ข้อกำหนดความผิดพลาดเป็นข้อมูลในการกำหนดขอบเขตในการค้นหาและข้อมูลสำหรับเติมข้อผิดพลาดลงในโครงสร้างต้นไม้ของรหัสต้นฉบับ แล้วทำการสุ่มตำแหน่งที่จะทำการกลายโครงสร้างต้นไม้จากตำแหน่งทั้งหมดในขอบเขตที่มีรูปแบบของโครงสร้างที่เป็นไปตามเงื่อนไข ภายหลังการกลายรหัสจะใช้โปรแกรมทดสอบในการตรวจสอบความผิดพลาดที่เกิดขึ้นของโปรแกรม

จากการกลายรหัสต้นฉบับของโปรแกรมภาษาจาวาในระดับของโครงสร้างต้นไม้ไวยากรณ์ทำให้สามารถเติมข้อบกพร่องลักษณะต่างๆลงในรหัสต้นฉบับได้ ทั้งข้อบกพร่องที่ทำให้เกิดความผิดพลาดจากไวยากรณ์ของภาษาหรือความผิดพลาดเชิงความหมาย โดยซอฟต์แวร์โปรแกรมการเติมข้อบกพร่องอัตโนมัตินี้มีการกลายรหัสสำหรับลักษณะความผิดพลาดระดับเมท็อดในลักษณะย่อยๆรวม 50 ลักษณะ ระดับคลาสในลักษณะย่อยๆ 6 ลักษณะ และในข้อบกพร่องพิเศษจากโครงสร้างเชิงวัตถุของภาษาจาวาเองอีก 4 ลักษณะ รวมเป็นลักษณะย่อยๆทั้งสิ้น 60 ลักษณะ ซึ่งได้รวมรวมไว้เป็นกลุ่มของความผิดพลาดเพื่อให้สะดวกในการใช้งานในการกลายรหัสอีก 35 ลักษณะ เพื่อความสะดวกและความเหมาะสมในการนำไปใช้เป็นโปรแกรมเพื่อสร้างแบบฝึกหัดอัตโนมัติสำหรับระบบการฝึกฝนการแก้ไขข้อบกพร่องในรหัสต้นฉบับของโปรแกรม

ในการทดสอบการทำงานของโปรแกรมกับรหัสต้นฉบับในการเรียนการสอนการเขียนโปรแกรมภาษาจาวาระดับพื้นฐานพบว่าสามารถเติมความผิดพลาดลักษณะต่างๆลงในรหัสต้นฉบับได้ตามจุดประสงค์การเรียนการสอนตามที่ได้ระบุไว้ในไฟล์ข้อกำหนดความผิดพลาด และพบว่าข้อบกร่องที่ได้เติมลงในรหัสต้นฉบับในตำแหน่งต่างๆนั้นมีความหลากหลายเนื่องจากความ

ผิดพลาดที่เติมลงไปนั้นจะมีลักษณะของการสุ่มทั้งตำแหน่งและสุ่มลักษณะความผิดพลาดที่จะทำการเติม

ประโยชน์ที่ได้รับจากการวิจัย

1) ได้ซอฟต์แวร์โปรแกรมการเติมข้อบกพร่องอัตโนมัติสำหรับสร้างแบบฝึกหัดให้แก่ระบบฝึกฝนการแก้ข้อผิดพลาดข้อบกพร่องในรหัสต้นฉบับของโปรแกรมที่มุ่งเน้นข้อผิดพลาดเชิงความหมาย

2) ช่วยส่งเสริมการเรียนการสอนวิชาการเขียนโปรแกรมภาษาจาวาระดับพื้นฐาน

3) ช่วยฝึกฝนนักเรียนให้เกิดความชำนาญในการเขียนโปรแกรม

4) ช่วยนักเรียนในการฝึกฝนทักษะการค้นหา การตรวจสอบ และ การแก้ไขความผิดพลาดที่เกิดขึ้นในโปรแกรม

5) ส่งเสริมความเข้าใจในความผิดพลาดที่เกิดขึ้น เพื่อลดปัญหาการเขียนโปรแกรมที่มีความผิดพลาด

ข้อจำกัดงานวิจัย

1) ข้อกำหนดข้อบกพร่องนั้นจะระบุได้เพียงขอบเขตของรหัสต้นฉบับที่ต้องการทำการเติมข้อบกพร่องเท่านั้น ไม่สามารถระบุในลักษณะที่เป็นข้อยกเว้นได้

2) การเพิ่มเติม แก้ไขเปลี่ยนแปลงและลบลักษณะข้อบกพร่องในลักษณะหรือ กลุ่มของข้อบกพร่องนั้น นักพัฒนาต้องเข้าไปแก้ไขในรายละเอียดของรหัสต้นฉบับ เพื่อให้โปรแกรมรู้จัก Visitor หรือ Modifier ยังไม่มีอินเตอร์เฟสสำหรับการนำงานในส่วนนี้

3) การจัดกลุ่มของข้อบกพร่องนั้นเป็นการจัดตามส่วนใหญ่เป็นเพียงการจัดตามความคล้ายคลึงกันของลักษณะของข้อบกพร่องทำให้ยังไม่มีความสมบูรณ์

4) ในการสร้างไฟล์ข้อกำหนดความผิดพลาดนั้น อาจารย์ผู้สอนต้องเป็นผู้ดำเนินการสร้างขึ้นเองโดยตรงเพราะยังไม่มีอินเตอร์เฟสเพื่อช่วยงานในส่วนนี้

ข้อเสนอแนะและแนวทางในการพัฒนาต่อ

1) เพิ่มเติมลักษณะความผิดพลาดให้หลากหลายมากขึ้น

2) จัดกลุ่มลักษณะความผิดพลาดให้มีความเหมาะสมมากขึ้นตัวอย่างเช่นกลุ่มความผิดพลาดเนื่องจากลักษณะความผิดพลาดจากการคำนวณอยู่ 1 (off-by-one)

3) วิเคราะห์ถึงความสัมพันธ์ระหว่างความผิดพลาดแต่ละลักษณะเพิ่มเติม สำหรับความผิดพลาดที่มักจะเกิดตามกัน

4) ทำการศึกษาถึงลักษณะความถี่ที่จะพบความผิดพลาดในแต่ละลักษณะเพื่อให้การเติมความผิดพลาดมีความเป็นธรรมชาติคล้ายกับรหัสที่ถูกเขียนขึ้นโดยนักเรียนผู้เขียนโปรแกรมจริงๆ

5) จัดทำอินเตอร์เฟสในส่วนต่างๆเพื่อช่วยให้การทำงานสะดวกยิ่งขึ้น

รายการอ้างอิง

[1] ดร.วีรศักดิ์ ซึงถาวร. Java programming Volume I (JavaSE 5.0): บริษัท ซีเอ็ดยูเคชั่น จำกัด (มหาชน), 2549.

[2] รศ.ดร.สมชาย ประสิทธิ์จูตระกูล. เริ่มเรียนเขียนโปรแกรม. พิมพ์ครั้งที่ 1 (มิถุนายน 2552): สำนักพิมพ์แห่งจุฬาลงกรณ์มหาวิทยาลัย, 2552.

[3] A. Ko and B. Myers, Development and Evaluation of a Model of Programming Errors: Human-Computer Interaction Institute, School of Computer Science, Carnegie Mellon University., 2003.

[4] A. Barr, Find The Bug: a book of incorrect programs, Addison Wesley, 2005.

[5] K. Donald., The Errors of TeX p.243 of Literate Programming.: Center for the Study of Language and Information, 1992.

[6] www.eclipse.org/articles/article.php

[7] www.wikipedia.org

[8] P. Arnold, S. Stephen, M. Lauri, M. Linda, A. Elizabeth, B. Jens, D. Marie, and P. James, A survey of literature on the teaching of introductory programming, in Working group reports on ITiCSE on Innovation and technology in computer science education. Dundee, Scotland: ACM, 2007

[9] M. Michael, A. Vicki, D. Danny, G. Mark, H. Dianne, K. Yifat Ben-David, L. Cary, T. Lynda, U. Ian, and W. Tadeusz, A multi-national, multi-institutional study of assessment of programming skills of first-year CS students, in Working group reports from ITiCSE on Innovation and technology in computer science education Canterbury, UK: ACM, 2001.

[10] L. Raymond, S. Elizabeth, F. Sue, F. William, H. John, L. Morten, M. Robert, M. Jan Erik, S. Kate, S. Otto, I, S. Beth, and T. Lynda, A multi-national study of reading and tracing skills in novice programmers, in Working group reports from ITiCSE on Innovation and technology in computer science education Leeds, United Kingdom: ACM, 2004.

[11] G. Lee and J. Wu, Debug It: A debugging practicing system, Computers & Education, vol. 32, pp. 165-179, 1999.

[12] A. Marzieh, E. Dave, et al. (2007). The Impact of Improving Debugging Skill on Programming Ability. Innovation in Teaching and Learning in Information and Computer Sciences Volume 6(Issue 4).

[13] M. C. Jadud, A first look at novice compilation behaviour using BlueJ, Computer Science Education, vol. 15, pp. 1-25, 2005.

[14] P. j. Vipindeep V, List of Common Bugs and Programming Practices to avoid them(Technical Report), 2005.URL: http://www.cse.iitd.ernet.in/~jalote/index.html

[15] M. Yu-Seung, O. Jeff, and K. Yong-Rae, MuJava: a mutation system for java, in Proceedings of the 28th international conference on Software engineering. 2006, ACM: Shanghai, China.

[16] M. Yu-Seung, M. Yu-Seung, K. Yong-Rae, and O. Jeff, Inter-class mutation operators for Java Inter-class mutation operators for Java, in Software Reliability Engineering, 2002. ISSRE 2002. Proceedings. 13th International Symposium on, 2002, pp. 352-363.

[17] O. Jeff, M. Yu-Seung, and K. Yong-Rae, The class-level mutants of MuJava, in Proceedings of the 2006 international workshop on Automation of software test Shanghai, China: ACM, 2006.

[18] S. P. Reiss, Finding unusual code, in Software Maintenance, 2007. ICSM 2007. IEEE International Conference on, 2007, pp. 34-43. [Online]. Available: http://dx.doi.org/10.1109/ICSM.2007.4362616

[19] N. Iulian, S. F. Jeffrey, and H. Michael, Understanding source code evolution using abstract syntax tree matching, SIGSOFT Softw. Eng. Notes, vol. 30, pp. 1-5, 2005.

ภาคผนวก

ภาคผนวก ก

บทความวิจัย

Application of AST for Enbugging on Debugging Training System

Practical training for realistic experience

Tanawut Wattanaprechagit and Somchai Prasitjutrakul
Department of Computer Engineering, Faculty of Engineering
Chulalongkorn University, Bangkok, 10330 Thailand

ABSTRACT

The lack of debugging training forces programming students to acquire the skill by themselves. Therefore, it is necessary and beneficial to invent an effective training system for them. In this paper, we present an application of the abstract syntax tree for adding defective codes into existing programs in order to create debugging training exercises for students. As a result, our system can insert many semantic errors into the programs.

Keywords

enbug, debug, debugging training, computer education, novice programming.

1. INTRODUCTION

Nature of learning any new skills be starting with poorly especially in computer programming. For many students, computer programming is very difficult because they have to learn many aspects of programming simultaneously. Students must learn the syntax of programming language, programming structure, computer process and computer programming design for solving any problems. These skills are hard for the beginners especially with debugging skill. Debugging is a skill for removing any defective codes from computer programs if it does not work correctly according to the programmer's requirement.

Students require debugging skill to cope with defective programs. Frequently students inject high-frequency defect into their programs [1] some defects are easy to find but most of them are hardly detected by students. Syntax errors are defective codes that can be identified by compiler at the compile time but other errors such as semantic errors must be identified by students. They take substantial amount of time to correct defective programs because debugging is difficult skill for them. Although students require debugging skill for coding programs, computer programming courses rarely train the debugging skill. The students must develop this skill by themselves. However, the students can be trained by debugging defective codes with commonly encountered bugs. These can help the students to reduce the occurrences of defects in their programs.

Many papers in computer science education field had reported in past of decades in many topics. A survey of literature on teaching of introductory programming [2] had identified these papers in four of categories curricula, pedagogy, language choice and the tools for teaching. Most of them are tools for teaching which has two main objectives. One objective of these tools is to improve the understanding of students or help them to learn in more easily ways and the other objective is to reduce the workload of the teachers. Typical targets of these tools are computer programming course management, plagiarism detection, automatic assessment or automatic program diagnosis system. However, there are a small number of papers that are interesting in debugging skill or finding way to improve debugging skill. Marzieh's research [3] has shown the relation between good programmers and good debuggers. The debugging skill has an effect to improve programming skill. Moreover, the research of Ryan and Michael [4] has shown a way to improve debugging skill, which needs practical experience. Hence, we propose to develop a system for training debugging skill of students in a practical way.

In this work we applied Abstract Syntax Tree to develop enbugging technique which emphasizes on high-frequency bugs and semantic bugs. These bugs are embedded into the original codes to be the exercises for the students. The students will practice their skill by our debugging training system. In this paper the meaning of enbug is the opposite of the word debug, namely, that to add errors or defects into software or hardware. The rest of paper is organized as follows. Section 2 gives a short background and overview of related works. Section 3 gives an idea of the methodology to construct the system. Section 4 reports the results of implementation in enbugging

and finally in section 5 we draw a conclusion and point to the further work.

2. Background and Related Works

To develop automatic enbugging technique for debugging training system which scopes to the semantic errors, we need to understand the structure of programs and require a tool for manipulate the structure of programs in order to inject the semantic errors which do not violate the language syntax of programs. There are two main components that are used as the tools for implementation the debugging training system: abstract syntax tree and common bugs.

2.1 Abstract syntax tree (AST). AST is the tree representation of abstract syntax program, which independent from concrete syntax of the underlying programming language such as identifiers, operators, conditions, or statements. AST is not only used in the representation of program structures, but is also used in structural matching to understand source code evolution [5], finding unusual source code [6] or mutable class pattern [7]. These benefits, along with its capability to be parsed back to the source code represented by the AST structure, satisfy our requirements. AST is the best choice for using in this development.

2.2 Common bug. Although Shoper and Soloway have reported that many students inject high-frequency defect code into their program until 1986[1], there is very tiny report about common and high-frequency defects, such as a report of Matthew in 2005[8] that is19 years after Shoper and Soloway published. The most of the information about these defects can be searched from computer programming books or web pages in form of cautions and alerts. Some of the information of defects was collected from Vipindeep's technical report [9]. We use this information to inject defect into the original source code.

3. Methodology

We begin our work by designing the system workflow. The resources and the limitations are planned for building the system. This system is developed for basic java programming course and all of systems are implemented in java language. AST, a tool from Eclipse [10], is chosen for our implementation. For describe our method we will explain in two level of view, overview of system and enbugging step to inject bug.

3.1 System work flow. The workflow of debugging training system is designed to train the debugging skill of students. This system must be easy to use and lessen the workload of the teacher. In computer programming course, teacher must give an exercise to students to increase their experience in programming. In general, the teacher will give the description of problems. Students must design and write program to solve these problems correctly. After they finished programming, students must compile and test their programs before submit them to the teacher.

Often, the teacher already has the original solution source code to his test idea. Moreover, some testers are required to test his requirements. These testers were written to reduce the workload of teacher. When the teacher received the exercises from students, these exercises will be tested by these testers. Hence, the teachers always have the problem description, original solution code and tester in exercise package.

The system workflow was designed in Figure 1. From resources in teaching, debugging training system will feed bug into the original source code. The tester will be used to ensure in enbugging and the optional requirements are used to tell the scope of bug type. There are only little extra workload for the teacher in our design.

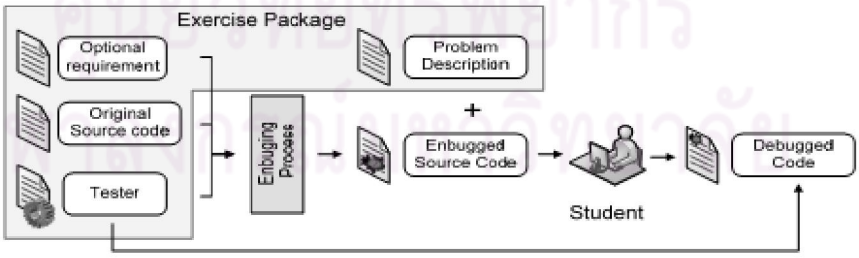


Figure 1 debugging training system work flow

3.2 Enbugging.

AST tool is used to feed the defect into the original source code in following step.

1) We begin by parsing the original source code to AST structure by using AST parser in first step.

2) Second, the AST structure will be modified. All possible enbugging patterns will be searched. Then, a pattern will be randomly selected to modify.

3) After the second step, some structure under the root of the AST will change to defect structure. After that, we parse AST back to source code and run tester to confirm that our modification can inject defect codes.

4) After running the tester, if the modification can cause defect, we remember this modification and start new injection until reach to the number of bugs in requirements.

5) The last two steps, we will combine all defect information into the original AST that were parsed from the original source code and parse it back to the defected source code which can be used as an exercise

These steps are shown in figure 2.

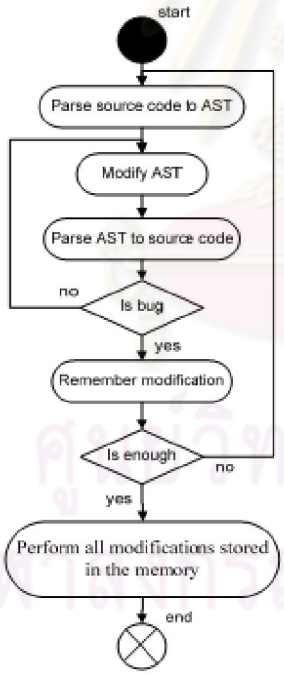


Figure 2 flow of enbugging step

4. RESULTS

In this section we will report 10 examples of bug type from our implementation. They are a variety of simple to complex defects resulting from modifications to the ASTs by three different methods (insert, change, or delete)

4.1 Bug from using false arithmetic operators. Java programming language supports three arithmetic operator unary, binary and short-cut. We modified these codes like table1.

Table 1 Bug from using false arithmetic operators.

Original code:
`a = b+c;`
`d = a/(b-c);`

Enbugged code:
`a =-b+c;`
`d = a*(b-c);`

4.2 Bug from using false relational operators. Java provides relational operators for comparing two values and determines the relation. We modified these codes like table2.

Table 2 Bug from using false relational operators.

Original code: `if(a>b){...}`
Enbugged code: `if(a>=b){...}`

4.3 Bug from using false logical and conditional operators. Java language provides six conditional operators and four logical operators to operate Boolean and bit values. We modified these codes like table3.

Table 3 Bug from using false logical and conditional operator.

Original code:
`if((a&&b)||c){`
`d=e&f;`
`}`

Enbugged code:
`if((!a&&b)^c){`
`d=e|f;`
`}`

4.4 Bug from Dangling else. It is easy to be confused about matching nested if–else statement. It is easy to miss else from the if-else statement. We modified these codes like table4.

Table 4 Bug from dangling else.

Original code

```
if(a){
        ...
}else if(b){
        ...
}
```

Enbugged code

```
if(a){
        ...
}
if(b){
        ...
}
```

4.5 Bug from break in switch case. This bug is frequently encountered. Omitting break in switch statement cause an extra statement to be executed. We modified these codes like table5.

Table 5 Bug from break statement.

Original code

```
switch(a){
        case 0: ... break;
        case 1: ... break;
}
```

Enbugged code

```
switch(a){
        case 0: ...
        case 1: ... break;
}
```

4.6 Bug from missing increment in iteration. This is an important bug Missing increment in iteration can cause infinite loop. We modified these codes like table6.

Table 6 Bug from missing increment in iteration.

Original code

```
While(a<10){
        ...
        a++;
}
```

Enbugged code

```
While(a<10){
        ...
}
```

4.7 Bug from missing initialization of objects. Each object must be initialized before its usage. Without the initial value specified, the object is null and can cause runtime error. We modified these codes like table7.

Table 7 Bug from missing initialization of object.

Original code

```
A a = new A();
B b = new B();
```

Enbugged code

```
A a = null;
B b;
```

4.8 Bug from comparison of object. All java objects are subclasses of java.lang.Object. The java == operator merely compare reference of object. Sometime, student misuse this statement. We modified these codes like table8.

Table 8 Bug from comparison object

Original code

```
objectA1.equals(objectA2);
```

Enbugged code

```
objectA1== objectA2;
```

4.9 Bug from swapping arguments. Same type of arguments in method calls is easily misused. Swapping argument can produce different outcome. We modified these codes like table9.

Table 9 Bug from swapping arguments.

Original code

```
method1(int arg1,int arg2){...}
method2(int arg1,int arg2){...}
...
method1( a,b);
```

Enbugged code

```
method1(int arg1,int arg2){...}
method2(int arg2,int arg1){...}
...
method1( b,a);
```

4.10 Bug from modifier change. Object oriented programming has many modifiers for different purposes, changing, deleting, and inserting them can cause defects. We modified these codes like table10.

Table 10 Bug from modifier change.

Original code `public int method(...){...}`
Enbugged code `private static int method(...){...}`

5. CONCLUSIONS AND FURTURE WORKS

We have presented application of AST for injecting defective codes into programs. In our work we can inject as many semantic bugs as we need and leave minimal extra workload to the teachers. In the future we will extend our type of bugs for training, implement debugging training system to train student and analyze efficiency of system on student.

REFERENCES

[1] J. C. Spohrer and E. Soloway, Analyzing the high frequency bugs in novice programs. In *Papers Presented At the First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers* (Washington, D.C., United States). E. Soloway and S. Iyengar, Eds. Ablex Publishing Corp., Norwood, NJ, 230-251.,1986.

[2] P. Arnold, S. Stephen, M. Lauri, M. Linda, A. Elizabeth, B. Jens, D. Marie, and P. James, "A survey of literature on the teaching of introductory programming," in *Working group reports on ITICSE on Innovation and technology in computer science education*. Dundee, Scotland: ACM, 2007.

[3] A. Marzieh, E. Dave, and H. Colin, "An analysis of patterns of debugging among novice computer science students," in *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*. Caparica, Portugal: ACM, 2005.

[4] C. Ryan and C. L. Michael, "Debugging: from novice to expert," in *Proceedings of the 35th SIGCSE technical symposium on Computer science education*. Norfolk, Virginia, USA: ACM, 2004.

[5] N. Iulian, S. F. Jeffrey, and H. Michael, "Understanding source code evolution using abstract syntax tree matching," *SIGSOFT Softw. Eng. Notes*, vol. 30, pp. 1-5, 2005.

[6] S.P. Reiss, "Finding unusual code," in *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, 2007, pp. 34-43. [Online]. Available: http://dx.doi.org/10.1109/ICSM.2007.4362616

[7] M. Nikolay, "Processing heterogeneous abstract syntax trees with the mutable class pattern," in *Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*. Nashville, TN, USA: ACM, 2008.

[8] M. C. Jadud, "A first look at novice compilation behaviour using BlueJ," *Computer Science Education*, vol. 15, pp. 1-25, 2005.

[9] P. j. Vipindeep V, "List of Common Bugs and Programming Practices to avoid them(Technical Report)," 2005.URL: http://www.cse.iitd.ernet.in/~jalote/index.html

[10] Eclipse.org, Eclipse JDT page, http://www.eclipse.org/jdt/

ประวัติผู้เขียนวิทยานิพนธ์

ว่าที่ร้อยตรี ธนาวุฒิ วัฒนปรีชากิจ เกิดเมื่อวันที่ 17 พฤษภาคม พ.ศ. 2526 ที่จังหวัดกรุงเทพมหานคร สำเร็จการศึกษาหลักสูตรวิทยาศาสตร์บัณฑิต สาขาวิชาวิทยาศาสตร์ฟิสิกส์ จากภาควิชาวิทยาศาสตร์ฟิสิกส์ คณะวิทยาศาสตร์ มหาวิทยาลัยมหิดล ในปีการศึกษา 2548 และเข้าศึกษาต่อในหลักสูตรวิทยาศาสตร์มหาบัณฑิต สาขาวิชาวิทยาศาสตร์คอมพิวเตอร์ ที่ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2549