การพัฒนาเครื่องมือเพื่อช่วยในการกู้คืนซอร์สโค้ดสำหรับภาษาจาวา

นายสมคะเน บาลลา

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

มหาวิทยาลัยเทคโนโลยีสุรนารี

ปีการศึกษา 2555

DEVELOPMENT OF TOOL FOR JAVA CODE RECOVERY

Somka-ne Balla

A Thesis Submitted in Partial Fulfillment of the Requirements for the

Degree of Master of Engineering in Computer Engineering

Suranaree University of Technology

Academic Year 2012

การพัฒนาเครื่องมือเพื่อช่วยในการกู้คืนซอร์สโค้ดสำหรับภาษาจาวา

มหาวิทยาลัยเทคโนโลยีสุรนารี อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

คณะกรรมการสอบวิทยานิพนธ์

(รศ. ดร.กิตติศักดิ์ เกิดประสพ)
ประธานกรรมการ

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

(ผศ. ดร.คะชา ชาญศิลป์)
กรรมการ

(ศ. ดร.ชูกิจ ลิมปิจำนงค์)
รองอธิการบดีฝ่ายวิชาการ

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

นายสมคะเน บาลลา : การพัฒนาเครื่องมือเพื่อช่วยในการกู้คืนซอร์สโค้ดสำหรับภาษาจาวา (DEVELOPMENT OF TOOL FOR JAVA CODE RECOVERY)
อาจารย์ที่ปรึกษา : ผู้ช่วยศาสตราจารย์ ดร.พิชโยทัย มหัทธนาภิวัฒน์, 84 หน้า.

จาวา (JAVA) เป็นภาษาที่นิยมนำมาใช้ในการเขียนโปรแกรมเชิงวัตถุเป็นอย่างมาก เนื่องจากจาวาเป็นภาษาที่มีคุณสมบัติสนับสนุนการเขียนโปรแกรมเชิงวัตถุอย่างสมบูรณ์และมีข้อดีคือมี virtual machine ในระบบปฏิบัติการหลายแบบ ทำให้เขียนเพียงครั้งเดียวสามารถนำไปใช้งานได้ทุกที่โดยไม่ต้องทำการคอมไพล์ใหม่ ทำให้สะดวกเมื่อนำไปใช้ในองค์กรที่มีฮาร์ดแวร์คละกันหลายแบบ อีกทั้งจาวายังเป็นที่นิยมในการนำไปใช้ในการพัฒนาระบบหรือซอฟต์แวร์ต่างๆ แต่เนื่องด้วยการพัฒนาระบบหรือซอฟต์แวร์นั้น จำเป็นที่จะต้องมีการแก้ไขข้อมูลหรือซอร์สโค้ดบ่อยครั้งเพื่อให้ระบบสามารถทำงานตรงตามที่ต้องการหรือทำตามความต้องการของผู้ว่าจ้างจึงทำให้เกิดปัญหาและความยุ่งยากในการจัดการกับซอร์สโค้ดที่มีการเปลี่ยนแปลง

ดังนั้นในงานวิจัยนี้จึงมุ่งเน้นเพื่อศึกษาและพัฒนาเครื่องมือเพื่อช่วยในการกู้คืนซอร์สโค้ดภาษาจาวาเพื่อทำให้กระบวนการในการจัดการกับซอร์สโค้ดที่มีการเปลี่ยนแปลงมีประสิทธิภาพเพิ่มขึ้น งานวิจัยนี้ได้ทำการพัฒนาเครื่องมือเพื่อช่วยในการกู้คืนซอร์สโค้ดโดยเครื่องมือสามารถทำการเปรียบเทียบหาความแตกต่างของซอร์สโค้ดและทำการกู้คืนซอร์สโค้ดที่ได้ทำการเปรียบเทียบโดยใช้เครื่องมือ

สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนักศึกษา _____
ปีการศึกษา 2555 ลายมือชื่ออาจารย์ที่ปรึกษา _____

SOMKA-NE BALLA : DEVELOPMENT OF TOOL FOR JAVA CODE RECOVERY. THESIS ADVISOR : ASST. PROF. PICHAYOTAI MAHATHANAPIWAT, Ph.D., 84 PP.

JAVA CODE/ DIFFERENCING ALGORITHM/ ABSTRACT SYNTAX TREE

The Java language (JAVA) is commonly used in object-oriented programming because it is the language that supports object-oriented programming features and it has a virtual machines for many operating systems. A single application can be used anywhere without the need to recompile making it convenient when used with a variety of hardware assortment. Java is also popular for application development or system software. Because the system or software may be changed frequently so the source code must be changed so that the system can meet the requirement of the employer. The problems and difficulties in dealing with the Java source code is challenging.

Therefore, this research focuses on the study and development of tool to aid in the recovery process in the Java source code and makes Java source code change management more effective. The research has developed a tool to assist in the recovery of source code by comparison the difference of the source code.

School of Computer Engineering Student's Signature _____

Academic Year 2012 Advisor's Signature _____

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จลุล่วงด้วยดี เนื่องจากผู้วิจัยได้รับการสนับสนุนในหลายด้าน จากบุคคลหลายท่าน ผู้วิจัยขอกราบขอบพระคุณ บุคคล และกลุ่มบุคคลหลายๆท่าที่ได้กรุณาให้คำปรึกษาแนะนำ ช่วยเหลืออย่างดียิ่งทั้งในด้านวิชาการและการดำเนินการวิจัย ซึ่งผู้วิจัยขอกราบขอบพระคุณดังนี้

ขอขอบพระคุณ ผู้ช่วยศาสตราจารย์ ดร.พิชโยทัย มหัทธนาภิวัฒน์ อาจารย์ที่ปรึกษาวิทยานิพนธ์

ขอขอบพระคุณ รองศาสตราจารย์ ดร.กิตติศักดิ์ เกิดประสพ หัวหน้าสาขาวิชาวิศวกรรมคอมพิวเตอร์ รองศาสตราจารย์ ดร.นิตยา เกิดประสพ ผู้ช่วยศาสตราจารย์สมพันธุ์ ชาญศิลป์ ผู้ช่วยศาสตราจารย์ ดร.คะชา ชาญศิลป์ ผู้ช่วยศาสตราจารย์ ดร.ปรเมศวร์ ห่อแก้ว และ ดร.ชาญวิทย์ แก้วกสิ อาจารย์ประจำสาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี

ขอขอบพระคุณ คุณกัลญา พับโพธิ์ เลขานุการสาขาวิชาวิศวกรรมคอมพิวเตอร์ ที่ให้ความช่วยเหลือในการประสานงานด้านเอกสารต่างๆ ระหว่างศึกษา

ขอขอบคุณ นายวีรศักดิ์ ช่องงูเหลือม ที่ช่วยให้แนวทางและคำปรึกษาในด้านเทคนิค

ขอขอบคุณ เพื่อน พี่ และน้อง บัณฑิตศึกษา สาขาวิชาวิศวกรรมคอมพิวเตอร์ทุกท่านที่ให้คำปรึกษาและช่วยเหลือ

ท้ายนี้ ขอกราบขอบพระคุณบิดา มารดา ที่ให้กำเนิด อบรม เลี้ยงดูด้วยความรักและส่งเสริมให้การศึกษาเป็นอย่างดี คอยให้ความรักความห่วงใย ให้กำลังใจ และคอยกระตุ้นเตือนสร้างแรงใจในการทำงานมาโดยตลอดจนทำให้งานวิจัยนี้สำเร็จลุล่วงด้วยดี

สมคะเน บาลลา

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

ตารางที่ **หน้า**

สารบัญรูป

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

รูปที่ **หน้า**

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของปัญหาการวิจัย

เนื่องจากในปัจจุบัน อุตสาหกรรมการซอฟต์แวร์และการพัฒนาระบบได้มีการเติบโตและขยายตัวอย่างรวดเร็ว สังเกตได้จากการเกิดขึ้นของบริษัทที่รับพัฒนาระบบต่างๆ Software house ขนาดต่างๆ ได้มีการเกิดขึ้นเป็นจำนวนมาก โดยบริษัทหรือ Software house ที่เกิดขึ้นเหล่านี้ได้มีการนำเทคนิคหรือทฤษฎีต่างๆมาใช้งานเพื่อช่วยในการพัฒนาระบบให้กับลูกค้าหรือผู้ว่าจ้าง ซึ่งเทคนิคหรือทฤษฎีต่างๆเหล่านี้ก็จะเป็นเทคนิคหรือทฤษฎีที่เกี่ยวข้องกับกระบวนการทางด้านวิศวกรรมซอฟต์แวร์ โดยในเทคนิคหรือกระบวนการทางด้านวิศวกรรมซอฟต์แวร์นั้นก็จะประกอบด้วยกระบวนการต่างๆ โดยกระบวนการหลักๆโดยทั่วไปที่นิยมใช้กันก็จะเหมือนๆกัน ซึ่งกระบวนการเหล่านั้นรวมๆแล้วก็จะถูกเรียกว่า วัฏจักรของการพัฒนาซอฟต์แวร์

วัฏจักรของการพัฒนาซอฟต์แวร์นั้นก็จะมีขั้นตอนหรือกระบวนการในการทำงานภายในที่แตกต่างกัน ซึ่งประกอบกันขึ้นเป็นวัฏจักรการพัฒนาซอฟต์แวร์ โดยทั่วไปแล้วกระบวนการต่างๆในวัฏจักรการพัฒนาซอฟต์แวร์นั้นจะแบ่งออกเป็นส่วนๆ หรือที่เรียกว่าเฟส(Phase) ซึ่งในวัฏจักรการพัฒนาซอฟต์แวร์ก็จะมีจำนวนเฟสต่างกันตามแนวทางการทำงานและกระบวนการในการทำงาน แต่โดยส่วนใหญ่แล้วกระบวนการการพัฒนาซอฟต์แวร์หรือที่เรียกว่าเฟส โดยหลักๆแล้วก็จะเหมือนกัน โดยกระบวนการหรือว่าเฟสหลักๆของการพัฒนาซอฟต์แวร์จะประกอบไปด้วย

- การหาความต้องการของระบบ (Requirement)
- การวิเคราะห์และออกแบบระบบ (System Analysis and Design)
- การสร้างและการพัฒนาระบบ (Implementation)
- การทดสอบ และการตรวจสอบระบบ (Testing and Verification)

ซึ่งจะเห็นได้ว่ากระบวนการหรือว่าเฟสหลักๆเหล่านี้จะมีอยู่ทั่วไปในการพัฒนาระบบ ซึ่งไม่ว่าจะเทคโนโลยีที่ใช้ในการพัฒนาระบบจะมีการเปลี่ยนแปลงไป กระบวนการเหล่านี้ก็ยังคงอยู่และก็ยังคงมีการใช้งานโดยนำเทคนิคหรือทฤษฎีต่างๆมาประยุกต์ใช้งานกับกระบวนการเหล่านี้อยู่อย่างต่อเนื่อง

ในกระบวนการสร้างและพัฒนาระบบ (Implementation) เป็นกระบวนการหนึ่งที่มีความสำคัญไม่ยิ่งหย่อนไปกว่ากระบวนการหรือเฟสการทำงานอื่น เนื่องจากเป็นกระบวนการที่ทำให้สามารถมองเห็นระบบที่เป็นรูปเป็นร่างได้อย่างชัดเจน

การเขียนโปรแกรมเชิงวัตถุ (object oriented programming) ในปัจจุบันมีความนิยมเป็นอย่างมาก การเขียนโปรแกรมเชิงวัตถุคือหนึ่งในรูปแบบการเขียนโปรแกรมคอมพิวเตอร์ที่ให้ความสำคัญกับวัตถุซึ่งสามารถนำมาประกอบกันหรือรวมกันทำงานได้ซึ่งแนวคิดที่สำคัญของการเขียนโปรแกรมเชิงวัตถุนั้นจะประกอบด้วย

- คลาส (Class) - ประเภทของวัตถุ เป็นการกำหนดว่า วัตถุ จะประกอบไปด้วย ข้อมูล (data) หรือคุณสมบัติ (property) และ พฤติกรรม (behavior) หรือการกระทำ (method) อะไรบ้าง ซึ่ง คลาส (เช่น มนุษย์) เป็นโครงสร้างพื้นฐานของการเขียนโปรแกรมเชิงวัตถุ

- วัตถุ (Object) - โดยมากจะเรียกว่า อ็อบเจกต์ คือ ตัวตน (instance) ของ คลาส ซึ่งจะเกิดขึ้นระหว่าง run-time โดยแต่ละ อ็อบเจกต์ จะมีข้อมูลเฉพาะของตัวเอง วัตถุในการเขียนโปรแกรมมักจะใช้ในการจำลองวัตถุในโลกของความเป็นจริงที่พบได้ในชีวิตประจำวัน โดยวัตถุจะประกอบด้วยคุณสมบัติซึ่งใช้แสดงสถานะหรือลักษณะของตัววัตถุเอง และวัตถุยังประกอบด้วยพฤติกรรม ซึ่งใช่บ่งบอกความสามารถของวัตถุ

- Encapsulation - การปิดบังข้อมูล เป็นวิธีการป้องกันในการเข้าถึงข้อมูล หรือการกระทำกับ วัตถุ ของ คลาสนั้นๆ ทำให้แน่ใจได้ว่าข้อมูลของวัตถุนั้นจะถูกเปลี่ยนแปลงแก้ไขผ่านทาง methods ที่เข้าถึงข้อมูลที่อนุญาตเท่านั้น

- Inheritance - การสืบทอดคุณสมบัติ เป็นวิธีการสร้าง คลาสย่อย (subclass) ซึ่งจะเป็นกำหนดประเภทของวัตถุให้จำเพาะเจาะจงขึ้น ซึ่ง คลาสย่อย จะได้รับถ่ายทอดคุณสมบัติต่างๆ มาจากคลาสหลักด้วย เช่น คลาส มนุษย์ สืบทอดมาจาก คลาส สิ่งมีชีวิต เป็นต้น

- Abstraction - นามธรรม เป็นการแสดงถึงคุณลักษณะและพฤติกรรมของ object เท่าที่จำเป็นต้องรับรู้และใช้งาน โดยซ่อนส่วนที่เหลือเอาไว้เพื่อไม่ให้เกิดความสับสน

- Polymorphism - ภาวะที่มีหลายรูปแบบ เป็นวิธีการกำหนดรูปแบบการกระทำที่เหมือนกันแต่ได้ผลที่แตกต่างกัน

จาวา (Java) เป็นภาษาที่นิยมนำมาใช้ในการเขียนโปรแกรมเชิงวัตถุเป็นอย่างมากเนื่องจากจาวาเป็นภาษาที่มีคุณสมบัติสนับสนุนการเขียนโปรแกรมเชิงวัตถุอย่างสมบูรณ์และมีข้อดีคือมี virtual machine ในระบบปฏิบัติหลายแบบ ทำให้เขียนเพียงครั้งเดียวสามารถนำไปใช้งานทุกที่ โดยไม่ต้องคอมไพล์ใหม่ ทำให้สะดวกเมื่อนำไปใช้ในองค์กรขนาดใหญ่ ที่มีฮาร์ดแวร์คละกันหลายแบบ อีกทั้งจาวายังเป็นที่นิยมในนำไปใช้ในการพัฒนาระบบหรือซอฟต์แวร์ต่างๆ แต่เนื่องด้วยการพัฒนาระบบหรือซอฟต์แวร์ต่างๆนั้น จำเป็นต้องมีการแก้ไขข้อมูลหรือโค้ดที่ใช้ในการพัฒนาระบบบ่อยครั้ง เพื่อให้ระบบสามารถทำงานตรงตามที่ต้องการ หรือทำให้ระบบทำงานได้ตรงตามที่ผู้ว่าจ้างต้องการจึงทำให้เกิดปัญหาในการแก้ไขข้อมูลหรือการกู้คืนโค้ดทำให้ผู้พัฒนาเกิดความสับสนเนื่องจากไม่มีเครื่องมือที่ช่วยบันทึกการแก้ไขหรือการกู้คืนข้อมูล

ดังนั้นงานวิจัยนี้จึงมุ่งเน้นเพื่อศึกษา และพัฒนาเครื่องมือเพื่อช่วยบันทึกการแก้ไขข้อมูลและการกู้คืนโค้ดเพื่อทำให้กระบวนการการพัฒนาระบบมีประสิทธิภาพเพิ่มขึ้น โดยงานวิจัยนี้ได้ทำการพัฒนาเครื่องมือเพื่อช่วยบันทึกและกู้คืนข้อมูลในระหว่างการพัฒนาซอฟต์แวร์หรือระบบที่พัฒนาด้วยภาษาจาวา โดยที่ตัวเครื่องมือสามารถบันทึกการเปลี่ยนแปลงต่างๆที่เกิดขึ้นกับข้อมูลหรือโค้ดที่อยู่ในระหว่างการพัฒนา และสามารถกู้คืนข้อมูลหรือโค้ดได้โดยการนำเทคนิคเกี่ยวกับ Concurrent Version System มาใช้ในการพัฒนาเครื่องมือนี้

1.2 วัตถุประสงค์การวิจัย

1.2.1 เพื่อศึกษาและพัฒนาเครื่องมือที่สามารถช่วยในกระบวนการการสร้างและพัฒนาระบบ

1.2.2 เพื่อปรับปรุงและพัฒนากระบวนการการจัดเก็บ สำรอง และการกู้คืนโค้ด ในระหว่างกระบวนการการพัฒนาระบบหรือพัฒนาซอฟต์แวร์ เพื่อให้เกิดความสะดวกต่อผู้พัฒนาระบบหรือซอฟต์แวร์

1.2.3 เพื่อพัฒนากระบวนการการพัฒนาระบบหรือซอฟต์แวร์โดยอาศัยเทคนิคและแนวคิด โดยการนำเทคนิคและแนวคิดเกี่ยวกับ Concurrent Version System มาประยุกต์ใช้และช่วยในการพัฒนาเครื่องมือ

1.2.4 เพื่อให้สามารถนำเครื่องมือที่ทำการพัฒนาไปใช้ในกระบวนการการพัฒนาระบบหรือซอฟต์แวร์ได้จริง

1.3 ข้อตกลงเบื้องต้น

1.3.1 เครื่องมือนี้จะนำเข้าซอร์สโค้ดที่มีชนิดเป็น java เท่านั้น

1.3.2 เครื่องมือนี้สามารถทำการเปรียบเทียบหาความแตกต่างของโค้ดภาษาจาวาสองเวอร์ชั่น โดยในการเปรียบเทียบจะสามารถทำการเปรียบเทียบหาความแตกต่างในระดับของโครงสร้างของซอร์สโค้ด โดยสามารถระบุความแตกต่าง การเพิ่ม หรือการลบของซอร์สโค้ดได้เท่านั้น

1.3.3 เครื่องมือนี้สามารถทำกระบวนการย้อนกลับข้อมูลไปยังเวอร์ชั่นที่ต้องการได้ โดยเวอร์ชั่นที่ต้องการนั้นต้องได้มาจากการเปรียบเทียบหาความแตกต่างโดยใช้เครื่องมือที่สร้างขึ้นนี้และซอร์สโค้ดที่นำมาเปรียบเทียบต้องมีความต่อเนื่องกัน

1.4 ขอบเขตของการวิจัย

โครงการวิจัยนี้เป็นการวิจัยและพัฒนาเครื่องมือที่ใช้ในกระบวนการการสร้างและพัฒนาระบบ(Implement Phase) โดยเป็นเครื่องมือที่ใช้ในการจัดเก็บ สำรองโค้ด และผู้ใช้สามารถทำการกู้คืนโดยอ้างอิงจากประวัติการทำงานที่ได้ทำการเก็บหรือบันทึกไว้ โดยเครื่องมือนี้จะพัฒนาด้วยภาษาจาวา

1.5 ประโยชน์ที่คาดว่าจะได้รับ

โครงการวิจัยนี้มีจุดมุ่งหมายที่จะพัฒนาเครื่องมือที่ช่วยในการกู้คืนโค้ดในระหว่างการพัฒนาระบบหรือซอฟต์แวร์โดยภาษาจาวา เพื่อช่วยให้กระบวนการการพัฒนาระบบหรือซอฟต์แวร์มีประสิทธิภาพและเพิ่มความสะดวกให้กับผู้พัฒนาระบบหรือผู้พัฒนาซอฟต์แวร์

บทที่ 2
ปริทัศน์วรรณกรรมและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะเป็นการนำเสนอวรรณกรรมและงานวิจัยที่เกี่ยวข้องกับงานวิจัยนี้ โดยในหัวข้อที่ 2.1 จะเป็นการกล่าวถึงความหมายของวิศวกรรมซอฟต์แวร์ ในหัวข้อที่ 2.2 จะกล่าวถึงกระบวนการพัฒนาซอฟต์แวร์และวัฏจักรในการพัฒนาซอฟต์แวร์ ซึ่งเป็นกระบวนการที่ถูกสร้างขึ้นมาเพื่อจัดการขั้นตอนการทำงานของการพัฒนาซอฟต์แวร์ให้มีการทำงานอย่างเป็นระเบียบมาตรฐาน ในหัวข้อที่ 2.3 จะกล่าวถึง Software Configuration Management ซึ่งเป็นมาตรฐานในการจัดการและควบคุมการเปลี่ยนแปลงที่เกิดขึ้นในระหว่างกระบวนการพัฒนาซอฟต์แวร์ สำหรับในหัวข้อที่ 2.4 จะกล่าวถึง Version Control System ซึ่งเป็นกระบวนการของการบันทึกและรวบรวมการเปลี่ยนแปลง หัวข้อที่ 2.5 จะกล่าวถึง Syntax Tree Matching ซึ่งเป็นเทคนิคหาความแตกต่างโดยการวิเคราะห์ความแตกต่างจากโครงสร้างของไวยากรณ์ภาษา และหัวข้อที่ 2.6 จะกล่าวถึงงานวิจัยที่เกี่ยวข้อง

2.1 วิศวกรรมซอฟต์แวร์ (Software Engineering)

Institute of Electrical and Electronics Engineers ได้ให้คำนิยามคำว่าวิศวกรรมซอฟต์แวร์ว่าหมายถึง การนำแนวทางที่เป็นระบบ มีระเบียบ กฎเกณฑ์ และสามารถวัดผลในเชิงปริมาณได้ มาประยุกต์ใช้ในการพัฒนา ปฏิบัติการ และบำรุงรักษาซอฟต์แวร์ ซึ่งก็คือ เพื่องานด้านวิศวกรรมซอฟต์แวร์ หรือกล่าวอีกนัยหนึ่งคือ เป็นการศึกษาวิธีการผลิตซอฟต์แวร์

The Canadian Standards Association ได้ให้คำนิยามคำว่าวิศวกรรมซอฟต์แวร์ว่า หมายถึงกิจกรรมที่เกี่ยวข้องกับการออกแบบ การสร้างและพัฒนา และการทดสอบซอฟต์แวร์อย่างเป็นระบบเพื่อเพิ่มประสิทธิภาพในการผลิตและสนับสนุนกิจกรรมเหล่านั้น

ในที่นี้ขอสรุปคำจำกัดความของ "วิศวกรรมซอฟต์แวร์" ไว้ดังนี้

วิศวกรรมซอฟต์แวร์ หมายถึง การนำหลักวิชาการด้านวิศวกรรมมาดูแลกระบวนการผลิตซอฟต์แวร์ ตั้งแต่ขั้นตอนแรกจนถึงขั้นตอนการบำรุงรักษา เพื่อให้ซอฟต์แวร์ที่ได้มีคุณภาพสูงสุดภายใต้ข้อจำกัดทางด้านเวลาและงบประมาณ (Timothy C., Lethbridege และ Robert Laganiere, 1992)

ในปัจจุบันการพัฒนาซอฟต์แวร์นั้น จำเป็นต้องใช้กระบวนการของวิศวกรรมซอฟต์แวร์เข้ามาช่วยจัดการและแก้ปัญหาให้กับลูกค้า เพื่อให้การพัฒนาซอฟต์แวร์เป็นไปอย่างมีประสิทธิภาพสามารถติดตามและควบคุมการพัฒนาให้เป็นไปตามระบบ เพื่อช่วยลดค่าใช้จ่ายและระยะเวลาในการพัฒนาและผลิตซอฟต์แวร์ และสุดท้ายเพื่อให้ซอฟต์แวร์ที่ได้มีคุณภาพและสามารถทำการบำรุงรักษาได้โดยง่าย

คุณสมบัติของซอฟต์แวร์ที่มีคุณภาพ (โอภาส เอี่ยมสิริวงศ์, 2548) มีดังนี้

1. มีความถูกต้อง (Correctness) คือ ความถูกต้องของซอฟต์แวร์กับความต้องการของผู้ใช้งานมีความตรงกัน

2. มีความน่าเชื่อถือ (Reliability) คือ ความน่าเชื่อถือในผลลัพธ์และข้อมูลต่างๆ ซึ่งความน่าเชื่อถือในข้อมูลเป็นสิ่งสำคัญต่อการตัดสินใจ

3. ใช้งานง่าย (User friendliness) หมายถึง ซอฟต์แวร์มีลักษณะการใช้งานที่เป็นมิตรต่อผู้ใช้งาน ใช้งานง่าย เรียนรู้ง่าย มีข้อความที่ครบถ้วน

4. มีความง่ายต่อการปรับเปลี่ยน (Adaptability) คือ ความสามารถในการปรับเปลี่ยนการใช้งานเพื่อให้เปิดความสอดคล้องกับความต้องการ หรือเทคโนโลยีที่เปลี่ยนแปลงไป

5. สามารถนำกลับมาใช้งานใหม่ได้ (Reusability) คือ ความสามารถในการนำกลับมาใช้ใหม่ซึ่งมีผลต่อต้นทุนและเวลา ทำให้ลดต้นทุนค่าใช้จ่ายและเวลาในการพัฒนาได้มาก

6. มีความเข้ากันได้กับระบบที่แตกต่าง (Interoperability) คือ คุณสมบัติของซอฟต์แวร์ที่สามารถใช้งานในระบบที่แตกต่าง

7. มีประสิทธิภาพ (Efficiency) คือ ผลของการใช้งานซอฟต์แวร์ ก่อให้เกิดการทำงานที่ดีขึ้นกว่าเดิม ค่าใช้จ่ายลดลง

8. มีความสะดวกในการเคลื่อนย้าย (Portability) คือ ความสะดวกของซอฟต์แวร์ที่สามารถเคลื่อนย้าย เพื่อนำไปใช้งานในสภาวะแวดล้อมใหม่

9. มีความปลอดภัย (Security) คือความปลอดภัยต่อข้อมูลที่อาจถูกปรับเปลี่ยนได้ ซึ่งหมายถึงการจำกัดสิทธิการใช้งานในระบบ เพื่อให้การเข้าถึงข้อมูลเป็นไปตามสิทธิของผู้ใช้งาน

2.2 กระบวนการพัฒนาซอฟต์แวร์และวัฏจักรในการพัฒนาซอฟต์แวร์ (Software Development Processes and Software Development Lift Cycle)

กระบวนการพัฒนาซอฟต์แวร์ คือ กลุ่มของกิจกรรมที่เกี่ยวเนื่องกันในการผลิตซอฟต์แวร์ ขั้นตอนและกิจกรรมต่างๆของกระบวนการการพัฒนาซอฟต์แวร์ถูกสร้างขึ้นมาเพื่อควบคุมและจัดการกับปัญหาต่างๆที่เกิดขึ้นจากการทำงาน (Roger S. Pressman, 1997) โดยสามารถแบ่งเป็นขั้นตอนได้ดังนี้

1) ขั้นตอนของการหาความต้องการ (Requirement) เป็นขั้นตอนของการรวบรวมรายละเอียดต่างๆ เพื่อจุดประสงค์ในการหาข้อสรุปที่ชัดเจนในด้านของความต้องการ (Requirements) ระหว่างผู้พัฒนากับผู้ใช้งาน เพื่อใช้ในขั้นตอนของกระบวนการวิเคราะห์และออกแบบต่อไป โดยการหา requirements ที่ดีมีหลักการดังนี้

- ตรงกับวัตถุประสงค์และหาข้อมูลกับบุคคลที่เกี่ยวข้องโดยตรง
- ควรระบุความต้องการต่างๆ ลงในรูปของเอกสารและเข้าใจทั้งสองฝ่าย ในบางครั้งอาจจะมีการเซ็นกำกับร่วมกันก็ได้ แต่ในกรณีนี้อาจสร้างความกดดันให้กับผู้ใช้งานได้ แต่ก็เป็นผลดีกับผู้พัฒนาระบบงาน ในกรณีที่มีการปรับแก้ในภายภาคหน้า
- Requirements ที่ดีต้องตกลงร่วมกันทั้งสองฝ่าย อย่าคิด วิเคราะห์ หรือออกแบบด้วยตนเองทั้งหมด อาจก่อให้เกิดผลเสียตามมา
- คำจำกัดความบนเอกสารต่างๆ ต้องชัดเจน อย่าเป็นคำจำกัดความที่กำกวมและสามารถตีความได้หลายความหมาย
- เป็นการยากในการหา Requirements ที่สมบูรณ์แบบด้วยการหาข้อมูลเพียงครั้งเดียว ดังนั้นควรมีการยอมรับในการปรับเปลี่ยนในภายภาคหน้าแต่ในการปรับเปลี่ยนที่ดีมิใช่เป็นการปรับเปลี่ยนหรือแก้ไขทั้งหมด เพราะหากเป็นเช่นนี้ถือว่ามีความผิดพลาดในการหาข้อมูลตั้งแต่ขั้นตอนแรก

2) ขั้นตอนการออกแบบระบบ (Architecture or Analysis and Design) เป็นขั้นตอนที่ต่อจากการเก็บรวบรวมความต้องการและนำมาทำการวิเคราะห์เพื่อออกแบบระบบ ในขั้นตอนของการวิเคราะห์ระบบจะทำการนำเอาความต้องการที่ได้มาทำการวิเคราะห์และทำการสร้างแบบจำลองเชิงตรรก (Logical Model) ซึ่งเป็นแผนภาพกระแสข้อมูล (Data Flow Diagram) ที่จะแสดงถึงกระบวนการการทำงานและข้อมูลที่เกี่ยวข้องภายในระบบจากนั้นจึงทำการออกแบบระบบโดยการนำ Logical Model ที่ได้จากการวิเคราะห์มาสร้างและออกแบบให้อยู่ในรูปของการปฏิบัติงานได้จริง (Physical-

Design) โดยผลลัพธ์ที่ได้จะต้องมีความถูกต้องและมีความน่าเชื่อถือเพื่อให้ตรงกับความต้องการของผู้ใช้งาน

3) ขั้นตอนการสร้างและการพัฒนา (Implementation) เป็นขั้นตอนของการนำเอาผลลัพธ์ที่ได้จากการวิเคราะห์มาทำการสร้างและพัฒนาระบบให้สามารถใช้งานได้ซึ่งขั้นตอนการสร้างและการพัฒนานี้จะรวมถึงการเขียนโปรแกรม (Coding) โดยผู้ที่รับหน้าที่ในการเขียนโปรแกรมคือโปรแกรมเมอร์ซึ่งจะมีหน้าที่ในการรับผิดชอบในการสร้างโมดูลต่างๆ ที่ได้รับมอบหมายมาจากนักวิเคราะห์ระบบ

4) ขั้นตอนการทำการทดสอบ (Testing) จะรวมไปถึงการทดสอบระบบในระหว่างการพัฒนาด้วย ขั้นตอนการทำการทดสอบเป็นขั้นตอนที่จะทำการทดสอบโปรแกรมหรือระบบที่ได้ทำการสร้างขึ้นมาว่าสามารถทำงานได้อย่างถูกต้องหรือไม่ ในขั้นตอนของการทดสอบอาจจะมีการจำลองสถานการณ์การดำเนินงานเพื่อทำการบันทึกข้อมูลเข้าสู่ระบบพร้อมทั้งการจำลองข้อมูลขึ้นมาเพื่อใช้งาน

5) ขั้นตอนการทำการส่งมอบและบำรุงรักษา (Deployment and Maintenance) เป็นกระบวนการที่เกิดขึ้นหลังจากที่ซอฟต์แวร์ผ่านการทดสอบและได้รับอนุญาตให้วางจำหน่ายหรือขายหรือกระจายซอฟต์แวร์ไปยังสภาพแวดล้อมการผลิตอื่นๆ ในการส่งมอบนี้จะรวมไปถึงการติดตั้งซอฟต์แวร์และการตั้งค่าต่างๆ การทดสอบในการใช้งานจริงและมีความเป็นไปได้ในการขยายเวลาการประเมิน หลังจากส่งมอบจะเป็นกระบวนการของการดูแลรักษาซึ่งเป็นกิจกรรมที่ต้องทำเมื่อเกิดการค้นพบข้อผิดพลาดของซอฟต์แวร์ ซึ่งอาจเกิดจากความผิดพลาดของกระบวนการหาความต้องการหรืออื่นๆ

สำหรับซอฟต์แวร์ที่มีวัตถุประสงค์ในการนำไปใช้งานแตกต่างกัน ลำดับขั้นตอนในกระบวนการผลิตจะแตกต่างกัน เนื่องจากการกำหนดกระบวนการผลิตซอฟต์แวร์นั้น ต้องพิจารณาถึงวัตถุประสงค์ในการนำไปใช้งาน และข้อจำกัดอื่นๆ ขององค์กรด้วย ขั้นตอนพื้นฐานของกระบวนการผลิตซอฟต์แวร์ที่แต่ละหน่วยงานกำหนดขึ้นจึงแตกต่างกัน วิธีการที่ง่ายที่สุดที่จะแสดงให้เห็นกระบวนการผลิตซอฟต์แวร์ได้ทั้งหมดนั้น คือ การสร้างเป็นแบบจำลองของกระบวนการ เรียกว่าแบบจำลองกระบวนการพัฒนาซอฟต์แวร์ (Software Process Model)

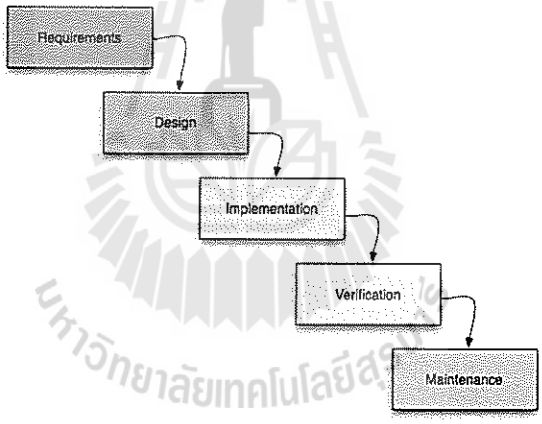
แบบจำลองกระบวนการพัฒนาซอฟต์แวร์ หมายถึง การจำลองภาพของกระบวนการผลิตซอฟต์แวร์เพื่อให้เห็นถึงการจัดการโครงสร้างลำดับขั้นตอนของกระบวนการในรูปแบบที่แตกต่างกันออกไป แบบจำลองกระบวนการพัฒนาซอฟต์แวร์นั้นได้ถูกคิดค้นขึ้นมาอย่างมากมาย เพื่อที่จะ

ตอบสนองต่อปัญหาที่เกิดขึ้นจากการพัฒนาซอฟต์แวร์ โดยที่แบบจำลองของกระบวนการพัฒนาซอฟต์แวร์แต่ละแบบนั้นมีข้อดีและข้อด้อยที่แตกต่างกันออกไปตามแต่ละยุคสมัยของเทคโนโลยี

2.2.1 The linear sequential model

Linear sequential model สำหรับในวิศวกรรมซอฟต์แวร์บางครั้งเรียกว่า classic life cycle หรือ waterfall model (Roger S. Pressman, 1997) เป็น software process model ที่มีความคลอบคลุมกระบวนการในวิศวกรรมซอฟต์แวร์ โดยหลักการเปรียบเสมือนน้ำตกซึ่งไหลจากที่สูงลงสู่ที่ต่ำ โดยกระบวนการหรือกิจกรรมภายใน waterfall model จะประกอบด้วย

- requirement
- design
- implementation
- verification
- maintenance

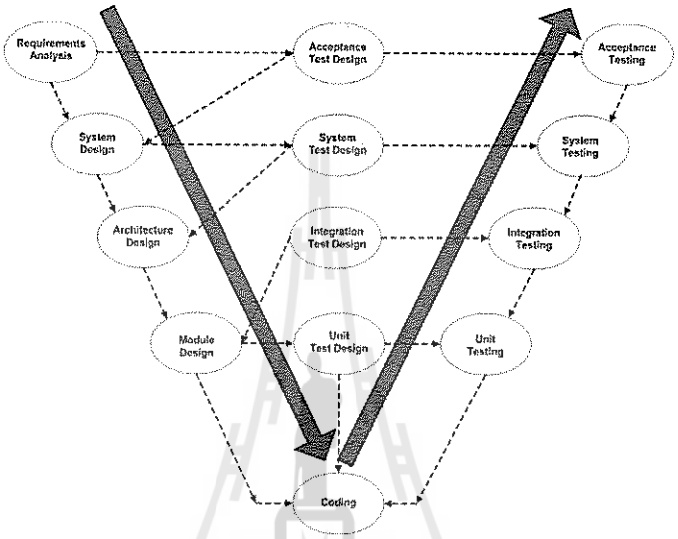


รูปที่ 2.1 กิจกรรมภายใน waterfall model

2.2.2 V-model

V-model หรือรูปแบบการพัฒนารูปตัว V เป็นรูปแบบการพัฒนาซอฟต์แวร์ที่ได้ทำการต่อยอดมาจากรูปแบบการพัฒนาแบบ waterfall model (Roger S. Pressman, 1997) ขั้นตอนกระบวนการพัฒนาจะทำการย้อนกลับขึ้นหลังจากที่การพัฒนาซอฟต์แวร์ดำเนินการมาจนจบขั้นตอน

ของการเขียนโปรแกรม ซึ่งขั้นตอนหลังจากทำการเขียนโปรแกรมจะเป็นการย้อนกลับ โดยเมื่อมองดูกระบวนการแล้วจะเป็นเหมือนรูปตัว V รูปแบบการพัฒนารูปตัว V นี้ เป็นการแสดงให้เห็นถึงความสัมพันธ์ระหว่างขั้นตอนแต่ละขั้นตอนของวงจรการพัฒนาซอฟต์แวร์ โดยความสัมพันธ์สามารถดูได้จากเฟสของการพัฒนาที่อยู่ตรงข้ามกันในแนวนอนโดยดูจากซ้ายไปขวาและลำดับขั้นตอนการดำเนินงานจะสามารถดูได้จากบนลงล่าง



รูปที่ 2.2 รูปแบบการพัฒนาซอฟต์แวร์แบบ V-model

จากรูปที่ 2.2 รูปแบบการพัฒนาซอฟต์แวร์แบบ V-model นั้น จะแบ่งเฟสการทำงานออกเป็นสองฝั่ง โดยฝั่งซ้ายเป็นเฟสการทำงานที่เกี่ยวข้องกับการพัฒนาซอฟต์แวร์ และฝั่งขวาจะเป็นเฟสการทำงานที่เกี่ยวข้องกับการ verification และ validation โดยขั้นตอนการทำงานทางฝั่งซ้ายและฝั่งขวาจะมีความสัมพันธ์กัน เช่น เมื่อทำการออกแบบระบบหรือ System Design จะสามารถทำการออกแบบ System Test ได้ และเมื่อขั้นตอนการทำงานมาถึงขั้นตอนของการทำ System Testing ก็จะใช้ชุดทดสอบที่ได้ออกแบบไว้เมื่อทำการทดสอบ หากเกิดข้อผิดพลาดก็จะสามารถย้อนกลับมายังการทำงานก่อนหน้านั้น ซึ่งก็คือการทำ Integration Testing เพื่อตรวจสอบความถูกต้อง

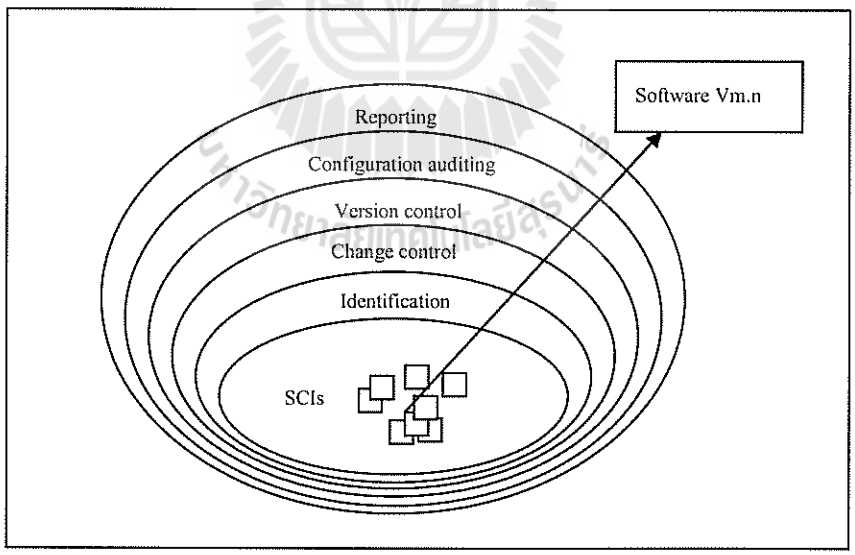
2.3 Software Configuration Management (SCM)

Software Configuration Management (SCM) หมายถึง ข้อกำหนดเพื่อสร้างมาตรฐานในการจัดการและควบคุมการเปลี่ยนแปลงในส่วนของวิวัฒนาการของการพัฒนางานทางด้านซอฟต์แวร์ (Wikipedia, 2552)

วัตถุประสงค์ของ SCM ก็เพื่อสร้างมาตรฐานและสามรถบำรุงรักษาความถูกต้องของผลิตภัณฑ์ทางด้านซอฟต์แวร์ในทุกวัฏจักรการทำงาน โดยส่วนหลักของ SCM นั้นตั้งใจที่จะลดความสับสนข้อผิดพลาดต่างๆที่เกิดขึ้นอันเนื่องมาจากความแตกต่างในแต่ละเวอร์ชั่นของซอฟต์แวร์

Software Configuration Management (SCM) เป็นกิจกรรมที่ครอบคลุมไปตลอดทั้งกระบวนการพัฒนาซอฟต์แวร์เพราะว่าการเปลี่ยนแปลงสามารถเกิดขึ้นได้ทุกเวลา (Roger S. Pressman, 1997) โดยกระบวนการของ SCM จะประกอบไปด้วย

- Identification
- Version control
- Change control
- Configuration auditing
- Reporting



รูปที่ 2.3 ลำดับชั้นของ Software Configuration Management Process

Identification คือ กิจกรรมที่ทำการหาข้อมูลเกี่ยวกับผลิตภัณฑ์เพื่อระบุเอกลักษณ์หรือความสัมพันธ์ของผลิตภัณฑ์กับสิ่งอื่นๆภายนอก Identification เป็นหนึ่งในสิ่งพื้นฐานของกระบวนการ Software Configuration Management ซึ่งจะเป็นไปไม่ได้ถ้าจะทำการควบคุมสิ่งต่างๆ โดยที่ไม่รู้ข้อมูล แต่ละองค์กรจะทำการตั้งข้อตกลงเพื่อใช้ในการระบุเกี่ยวกับผลิตภัณฑ์ของตนเอง เป็นการยากที่จะทำการตั้งข้อตกลงเพื่อใช้ในการระบุสิ่งต่างๆ ควรพิจารณาถึงวัตถุประสงค์ในการระบุและการนำไปใช้ด้วยวิธีที่ไม่ยากจนเกินไป

Version Control คือ กระบวนการของการบันทึกและรวบรวมการเปลี่ยนแปลงที่เกิดขึ้นในตลอดวัฏจักรของการพัฒนาซอฟต์แวร์พร้อมทั้งสามารถระบุและรวบรวมการเปลี่ยนแปลงทั้งหมดที่เกิดขึ้นเข้าด้วยกัน

Change Control คือ การควบคุมการร้องขอการเปลี่ยนแปลงที่เกิดขึ้นกับผลิตภัณฑ์หรือซอฟต์แวร์ และการควบคุมการเปลี่ยนแปลงของการพัฒนาผลิตภัณฑ์ วัตถุประสงค์ของการควบคุมการเปลี่ยนแปลงคือควบคุมและจัดการทุกการเปลี่ยนแปลงที่เกิดขึ้นกับผลิตภัณฑ์หรือเกิดขึ้นในระหว่างการพัฒนา ทุกๆการเปลี่ยนแปลงควรจะสามารถทำการตรวจสอบว่าแต่ละผลิตภัณฑ์หรือซอฟต์แวร์มีการเปลี่ยนแปลงที่ใดบ้าง

Configuration Auditing คือ การตรวจสอบประสิทธิภาพโดยแยกย่อยแบ่งออกเป็นตรวจสอบฟังก์ชั่นและการตรวจสอบทางกายภาพ (physical configuration auditing) โดยการตรวจสอบเพื่อให้แน่ใจว่า ฟังก์ชั่นต่างๆของผลิตภัณฑ์หรือซอฟต์แวร์หรืออุปกรณ์ต่างๆทำงานได้อย่างถูกต้องมีประสิทธิภาพหลังทำการออกวางจำหน่าย ในการตรวจสอบโดยส่วนใหญ่มักจะใช้เทคนิคหรือวิธีการของการประกันคุณภาพ (Quality Assurance) มาใช้ในการตรวจสอบ เช่นการรีวิวและทดสอบ เป็นต้น

Reporting คือ รายงานเกี่ยวกับผลิตภัณฑ์ ซึ่งข้อมูลเป็นสิ่งที่จำเป็นในการบริหารจัดการกับสิ่งต่างๆที่อาจจะมีผลกระทบกับผลิตภัณฑ์หรือซอฟต์แวร์หรือกระทบกับการพัฒนาหรือซ่อมบำรุง ในกิจกรรมส่วนต่างๆของ Software Configuration Management จะทำการส่งข้อมูลมายังส่วนรายงานและส่วนรายงานมีหน้าที่ทำการขยาย เรียบเรียงและจัดรูปแบบให้เป็นไปตามความต้องการ ผลลัพธ์ของรายงานส่วนใหญ่จะออกมาในรูปของเอกสารรายงาน โดยแต่ละองค์กรหรือบริษัทจะเป็นคนกำหนดรูปแบบของรายงาน เป็นสิ่งสำคัญที่จะต้องทำความเข้าใจข้อมูลที่ถูกต้องที่ถูกส่งมาจากกิจกรรมต่างๆ

2.4 Version Control System

Version Control System คือ ระบบของการบันทึกและรวบรวมการเปลี่ยนแปลงที่เกิดขึ้นในโปรเจค Version Control System อนุญาตให้สามารถเรียกข้อมูลของเวอร์ชั่นเก่ากลับมาเพื่อแก้ไขข้อบกพร่อง หรือ เพิ่มเติมความสามารถให้กับซอฟต์แวร์เพื่อให้เกิดความทันสมัย อนุญาตให้ผู้พัฒนาโครงการสามารถทำการพัฒนาได้พร้อมกัน และจะทำการสร้างรายงานการเปลี่ยนแปลงที่เกิดขึ้นในโครงการ และจะอนุญาตให้หัวหน้าโครงการสามารถทำการวิเคราะห์การเปลี่ยนแปลงที่เกิดขึ้นได้ (Jennifer Vesperman, 2003) โดยส่วนใหญ่ Version Control System จะมีความสามารถในการเรียกดูความแตกต่างระหว่างรุ่นของข้อมูลซึ่งเป็นการง่ายในการที่จะระบุข้อผิดพลาดและแก้ไข

ประโยชน์ของ Version Control System เช่น CVS (Concurrent Version System) มีดังนี้

- เก็บรวบรวมไฟล์เวอร์ชั่นต่างๆและสามารถเรียกกลับมาดูหรือแก้ไขได้
- สามารถแสดงความแตกต่างระหว่างไฟล์สองเวอร์ชั่นได้
- มีการสร้างส่วนที่แก้ไขให้อัตโนมัติ
- ผู้พัฒนาหลายๆคนสามารถทำงานได้พร้อมกันในโครงการเดียวกัน โดยไม่มีการสูญหายของข้อมูล
- โครงการสามารถแยกไปพัฒนาได้พร้อมกันพร้อมทั้งสามารถติดตามการเปลี่ยนแปลงได้ ส่วนที่แยกไปพัฒนาสามารถนำกลับเข้ามาสู่การพัฒนาหลักได้
- สนับสนุนการกระจายการพัฒนาในเครือข่ายขนาดเล็กหรือขนาดใหญ่

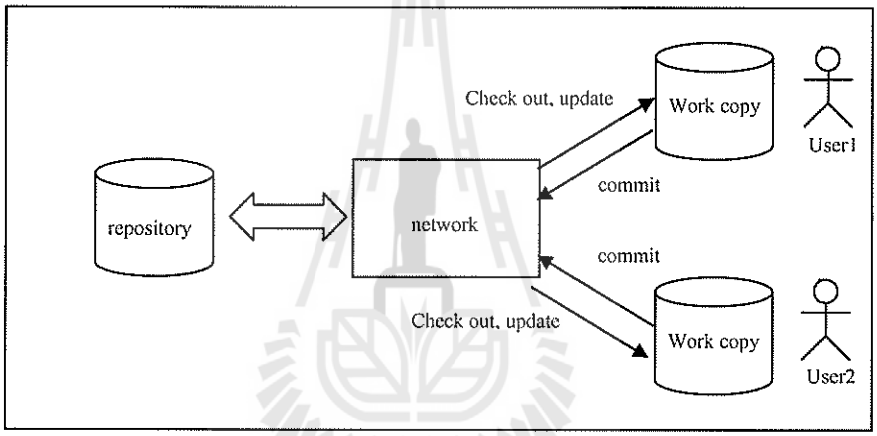
โดยปกติแล้วจะมีการใช้ Version Control System ในการเขียนโปรแกรม แต่ว่า Version Control System ยังมีประโยชน์สำหรับงานเขียน, ผู้ดูแลระบบ และอื่นๆที่ไฟล์มีการเปลี่ยนแปลงบ่อยๆ บางทีในการพัฒนาซอฟต์แวร์อาจจะเป็นการใช้ Version Control System ที่มากที่สุด เพราะหลังจากที่มีการออกวางจำหน่ายซอฟต์แวร์เวอร์ชั่นที่สองของซอฟต์แวร์มักจะมีการปรับปรุงจากเวอร์ชั่นแรก ซึ่งในที่สุดก็จะเป็นเวอร์ชั่นใหม่ซึ่งเป็นเวอร์ชั่นที่ได้มีการแก้ไขข้อผิดพลาดจากเวอร์ชั่นก่อน

ในหลายๆโครงการ ระบบควบคุมเวอร์ชั่นหรือ Version Control System (VCS) และเครื่องมืออื่นๆ เช่น ระบบติดตามเวอร์ชั่น (tracking system) เป็นศูนย์กลาง(Centralized)ในการจัดการวิธีการทำงาน ความท้าทายหลักในการจัดการบริหารการพัฒนาซอฟต์แวร์คืออัตราการเปลี่ยนแปลงที่ขึ้นอยู่กับจำนวนนักพัฒนาที่กระจายตัวโดยปราศจากการลดลงของคุณภาพหรือค่าใช้จ่ายที่เกินควร ในขอบเขตที่กว้างเครื่องมือเหล่านี้ได้กำหนดวิธีการที่บุคคลสามารถมีส่วนร่วมกับโครงการได้อย่างง่ายดาย วิธีการพัฒนาแบบใหม่ที่สามารถทำงานไปพร้อมๆกัน ความถี่ในการแยกและรวมการพัฒนา วิธีการตรวจสอบ

โค้ดและวิธีการสนับสนุนการจัดการกับโค้ดที่พร้อมออกวางจำหน่าย แม้เครื่องมือเหล่านี้จะมีความสำคัญ แต่ผลกระทบที่เกี่ยวข้องที่ยังไม่ได้ศึกษาของเครื่องมือเหล่านี้สามารถมีผลกับการพัฒนาและการแลกเปลี่ยน

2.4.1 Concurrent Version System (CVS)

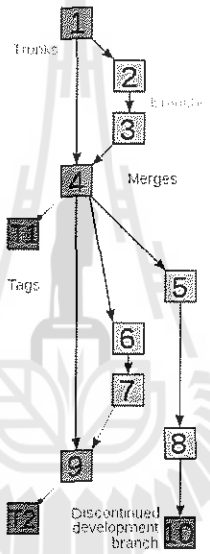
Concurrent Version System (CVS) คือระบบควบคุมเวอร์ชั่นการพัฒนาของงานทางด้านซอฟต์แวร์ โดย CVS จะทำการบันทึกการเปลี่ยนแปลงระหว่างการพัฒนาซอฟต์แวร์หรือโครงการ ซึ่งไฟล์ของการพัฒนาจะถูกเก็บไว้ในศูนย์กลางการเก็บข้อมูล (Repository) ที่ได้สร้างไว้ และผู้พัฒนาจะทำการเรียกไฟล์ออกมาจากศูนย์กลางการเก็บข้อมูลเพื่อนำออกมาพัฒนาและหลังจากพัฒนาก็จะทำการส่งคืนกลับเข้าไปยังศูนย์กลางการเก็บข้อมูล (Jennifer Vesperman, 2003)



รูปที่ 2.4 ศูนย์กลางการเก็บข้อมูลและการทำงานกับสำเนาไฟล์

CVS จะทำงานผ่านระบบ client-server หรือระบบ internet เซิร์ฟเวอร์ทำการเก็บโปรเจคและประวัติการเปลี่ยนแปลง จากนั้นผู้ใช้งานจะทำการ check-in ข้อมูลที่ต้องการออกมาจากเซิร์ฟเวอร์ โดยข้อมูลที่ได้จะเป็นข้อมูลฉบับที่ CVS ทำการสำเนาไว้ เมื่อทำการแก้ไขข้อมูลเสร็จก็จะ check-in กลับเข้าสู่เซิร์ฟเวอร์ โดยเซิร์ฟเวอร์จะทำการอัพเดทการเปลี่ยนแปลงข้อมูลที่ผู้ใช้ทำการ check-in ให้เป็นเวอร์ชั่นล่าสุดอยู่เสมอ นอกจากนี้แล้ว CVS ยังสามารถทำการเปรียบเทียบความแตกต่างของข้อมูลในแต่ละเวอร์ชั่นได้

ในวิศวกรรมซอฟต์แวร์ version control คือการควบคุมและติดตามการเปลี่ยนแปลงที่เกิดขึ้นกับซอร์สโค้ด (Wikipedia, 2012) ผู้พัฒนาซอฟต์แวร์บางครั้งใช้ version control ในการปรับปรุงเอกสารและไฟล์ configuration เหมือนที่ใช้กับซอร์สโค้ด ในการออกแบบ พัฒนาและติดตั้งก็เป็นเรื่องธรรมดาสำหรับซอฟต์แวร์หลายเวอร์ชั่นที่ติดตั้งคนละสถานที่จะแตกต่างกันและสำหรับนักพัฒนาซอฟต์แวร์ที่ทำงานพร้อมกันในการปรับปรุงแก้ไขข้อบกพร่องหรือคุณสมบัติของซอฟต์แวร์ ดังนั้นเพื่อวัตถุประสงค์ในการติดตั้ง และการแก้ไขข้อบกพร่อง version control จึงเป็นสิ่งสำคัญและจำเป็นเพื่อให้สามารถดึงข้อมูลและเรียกใช้งานเวอร์ชั่นที่แตกต่างกันของซอฟต์แวร์เพื่อตรวจสอบในเวอร์ชั่นที่เกิดปัญหา



รูปที่ 2.5 ตัวอย่างของแผนภูมิต้นไม้แสดงประวัติของโปรเจคที่ใช้ version control (Wikipedia, 2012)

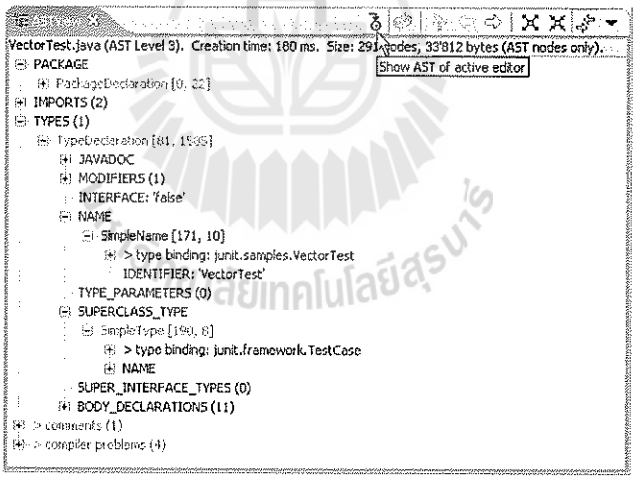
จากรูปที่ 2.5 เป็นการแสดงโครงสร้างแผนภูมิต้นไม้ของประวัติของโปรเจคที่ใช้ version control เข้ามาจัดการ โดย 1 คือ master version ที่เป็นแกนหลักของโปรเจค จากนั้นทำการแยกการพัฒนาเป็นเวอร์ชั่น 2 และ 3 แล้วจึงทำการรวมกันเป็นเวอร์ชั่น 4 ซึ่งเวอร์ชั่น 4 ได้กลายมาเป็นแกนกลางในการพัฒนาในเวอร์ชั่นต่อไป ซึ่งในเวอร์ชั่น 4 นี้ ได้มีการทำการ tags ซึ่งเปรียบเสมือนการวางจำหน่าย หรือพร้อมที่จะนำออกไปใช้งาน หลังจากนั้นมีการพัฒนาต่อ โดยการแยกการพัฒนาเป็นสองสายการพัฒนา ซึ่งก็คือเวอร์ชั่น 5 และ 6 โดยทั้งสองเวอร์ชั่นได้มีการพัฒนาไปพร้อมกัน โดยที่เวอร์

ชั่น 6 ได้มีการพัฒนาจนไปถึงเวอร์ชั่น 7 และทำการรวมกลายเป็นเวอร์ชั่น 9 ซึ่งกลายเป็นแกนกลางและได้มีการ tags ออกไป ส่วนเวอร์ชั่น 5 นั้นได้ทำการพัฒนาไปเป็นเวอร์ชั่น 8 จนถึงเวอร์ชั่น 10 และได้หยุดการพัฒนาลง

ซึ่งจากรูปจะเห็นได้ว่า ในการพัฒนาซอฟต์แวร์โดยการนำ version control มาใช้นั้น มีความสะดวกในการจัดการกับเวอร์ชั่นต่างๆ ซึ่งสามารถทำให้แยกการพัฒนาซอฟต์แวร์ออกไปได้หลากหลาย และยังสามารถทำการพัฒนาในระหว่างที่ทำการวางจำหน่ายในเวอร์ชั่นที่ได้มีการ tags ออกไปได้

2.5 Syntax Tree Matching

Abstract Syntax Tree หรือ Syntax tree เป็นการแสดงโครงสร้างของ source code ในแต่ละภาษาในรูปแบบของโครงสร้างต้นไม้ โดยในแต่ละโหนดของโครงสร้างต้นไม้จะรวบรวมข้อมูลต่างๆ ของ source code ไว้ เช่น ชื่อตัวแปร, ประเภทของตัวแปร เป็นต้น ในการจะหาโครงสร้างของ AST นั้นจำเป็นจะต้องใช้เครื่องมือช่วยวิเคราะห์โครงสร้าง ซึ่งจะแตกต่างกันไปในแต่ละภาษาที่ใช้ตัวอย่างเช่น OCaml framework สำหรับภาษาซี, ASTParser หรือ ANTLR สำหรับภาษาจาวา เป็นต้น



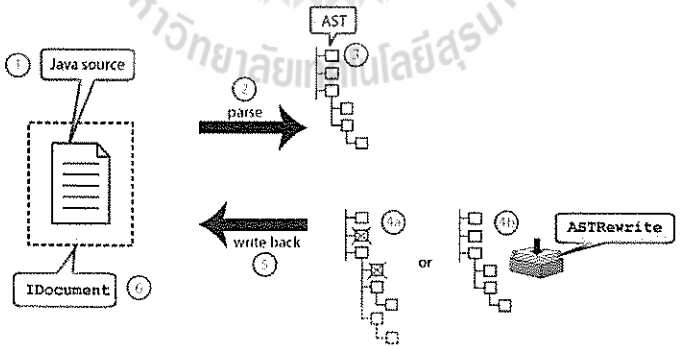
รูปที่ 2.6 ตัวอย่าง Abstract Syntax Tree ที่สร้างโดย ASTParser ของภาษาจาวา

Syntax Tree Matching เป็นเทคนิคในการหาเวอร์ชั่นของซอฟต์แวร์โดยการใช้ Syntax Tree มาช่วยในการวิเคราะห์ความแตกต่างของแต่ละเวอร์ชั่น โดยเทคนิคนี้จะอาศัยความสัมพันธ์ระหว่างโหนดมาช่วยในการวิเคราะห์ ซึ่งในการวิเคราะห์หาเวอร์ชั่นของซอฟต์แวร์นั้นจะใช้อัลกอริทึมที่แตกต่างกันในการหาเวอร์ชั่น โดยอัลกอริทึมที่เป็นที่นิยมใช้ได้แก่ Tree Edit Distance, Bottom-up Maximum Common Subtree Isomorphism, Top-down Maximum Common Subtree Isomorphism, LCS เป็นต้น

2.5.1 Eclipse Abstract Syntax Tree(AST)

Eclipse Abstract Syntax Tree คือกรอบงานพื้นฐาน(base framework) สำหรับเครื่องมือของ Eclipse IDE ซึ่งรวมเอาความสามารถในการ Refactoring, Quick fix และ Quick Assist. โดย Eclipse Abstract Syntax Tree จะจัดรูปแบบของซอร์สโค้ดภาษาจาวาให้อยู่ในรูปแบบของโครงสร้างต้นไม้(Tree) (Thomas Kuhn และ Olivier Thoman, 2006) โดยรูปแบบต้นไม้นี้จะเพิ่มความสะดวกสบายและความน่าเชื่อถือในการวิเคราะห์และการปรับเปลี่ยนโปรแกรมที่มากกว่าในรูปแบบของ text-based

Abstract Syntax Tree เป็นแนวทางในการเข้าถึงซอร์สโค้ดของภาษาจาวาของ Eclipse IDE โดยทุกๆจาวาโค้ดจะอยู่ในรูปแบบของโหนดของ AST โดยแต่ละโหนดจะเป็นคลาสย่อยของ ASTNode โดยในทุกๆคลาสย่อยของ ASTNode จะระบุส่วนประกอบต่างๆของ Java Programming Language ตัวอย่างเช่น โหนดของการประกาศเมธอด(MehtodDeclaration), การประกาศตัวแปร (VariableDeclarationFragment), การระบุค่าและอื่นๆ โดยคลาสของ AST ที่เกี่ยวข้องจะอยู่ในแพ็คเกจ org.eclipse.jdt.core.dom (Thomas Kuhn และ Olivier Thoman, 2006)



รูปที่ 2.7 ขั้นตอนการทำงานปกติของโปรแกรมที่ใช้ Eclipse AST

จากรูปที่ 2.7 เป็นการแสดงขั้นตอนของการนำเอา Eclipse Abstract Syntax Tree ไปใช้งาน โดยขั้นตอนการทำงานจะเริ่มจากการนำเข้าซอร์สโค้ดต้นฉบับ จากนั้นทำการส่งซอร์สโค้ดต้นฉบับไปทำการกระจายนิพจน์โดยการใช้คลาส ASTParser โดยขั้นตอนการใช้งานคลาส ASTParser จะได้อธิบายในหัวข้อถัดไป หลังจากทำการกระจายนิพจน์แล้ว ผลลัพธ์ที่ได้จากการกระจายจะอยู่ในรูปของ Abstract Syntax Tree ในการจัดการกับ Abstract Syntax Tree นั้นหากต้องการเปลี่ยนแปลงโครงสร้างสามารถทำได้ 2 วิธี คือ การแก้ไขในส่วนของ Abstract Syntax Tree โดยตรง หรือจะทำการแก้ไขโดยใช้โปรโตคอลที่แยกต่างหากซึ่งโปรโตคอลนี้จะสามารถควบคุมได้ด้วย instance ของ ASTWriter คลาส เมื่อทำการแก้ไขหรือปรับเปลี่ยนในส่วนของ Abstract Syntax Tree แล้ว หากต้องการทำการบันทึกส่วนที่แก้ไขลงไปยังซอร์สโค้ดสามารถบันทึกการเปลี่ยนแปลงโดยการใช้ instance ของ TextEdit คลาสในการบันทึกลงไปยังซอร์สโค้ด

2.5.2 ASTParser

ASTParser เป็นคลาสสำหรับทำการกระจายนิพจน์ภาษาจาวาเพื่อสร้าง abstract syntax tree (IBM Corporation and other, 2000) โดย ASTParser สืบทอดมาจากคลาส Object โดยในการกระจายนิพจน์จะต้องทำการสร้าง element ของ ASTParser โดยการสร้าง element ของ ASTParser นั้นสามารถสร้างได้จาก java model เช่น ICompilationUnit หรือสร้างจาก String ดังรูปที่ 2.5 จากนั้นสร้างโหนดของซอร์สโค้ดโดยใช้ ASTNode

```
ASTParser parser= ASTParser.newParser(AST.JLS3);
parser.setSource("System.out.println();".toCharArray());
parser.setProject(javaProject);
parser.setKind(ASTParser.K_STATEMENTS);
ASTNode node= parser.createAST(null);
```

Create AST on source string

รูปที่ 2.8 ตัวอย่างการใช้ ASTParser ในการกระจายนิพจน์

จากรูปที่ 2.8 การจะสร้าง Abstract Syntax Tree นั้น จะต้องทำการสร้าง instance ของ ASTParser คลาส ซึ่งคำสั่ง ASTParser.newParser(AST.JLS3) เป็นการสั่งให้ตัวกระจายนิพจน์ทำการกระจายนิพจน์โดยกำหนดให้กระจายนิพจน์ของภาษาจาวาที่กำหนด โดย JSL3 ได้รวมข้อจำกัดของภาษาจาวาทั้งหมดรวมทั้งไวยากรณ์ใหม่ที่มีในภาษาจาวาเวอร์ชั่น 5 จากนั้นจึงทำการกำหนดซอร์สโค้ด

ที่จะนำมาสร้าง Abstract Syntax Tree คำสั่ง parser.setKind(ASTParser.K_STATEMENTS) เป็นคำสั่งที่บอกให้ตัวกระจายนิพจน์ทราบถึงประเภทของซอร์สโค้ดที่จะนำเข้ามาทำการกระจายนิพจน์ โดยประเภทของซอร์สโค้ดที่จะนำเข้ามาทำการกระจายนิพจน์แบ่งออกได้เป็น (Thomas Kuhn และ Olivier Thoman, 2006)

K_COMPILATION_UNIT เป็นส่วนของซอร์สโค้ดภาษาจาวา นำเข้าในรูปแบบของ char[]

K_EXPRESSION เป็นประเภทข้อมูลที่นำเข้าซึ่งประกอบด้วย Expression ของภาษาจาวา ตัวอย่างเช่น new String()

K_STATEMENTS เป็นข้อมูลประเภทที่นำเข้าซึ่งประกอบด้วย Statement ของภาษาจาวา ตัวอย่างเช่น System.out.println("Hello world");

K_CLASS_BODY_DECLARATIONS เป็นข้อมูลที่นำเข้าซึ่งเป็นส่วนประกอบของจาวาคลาส เช่น method declaration, field declaration, static block เป็นต้น

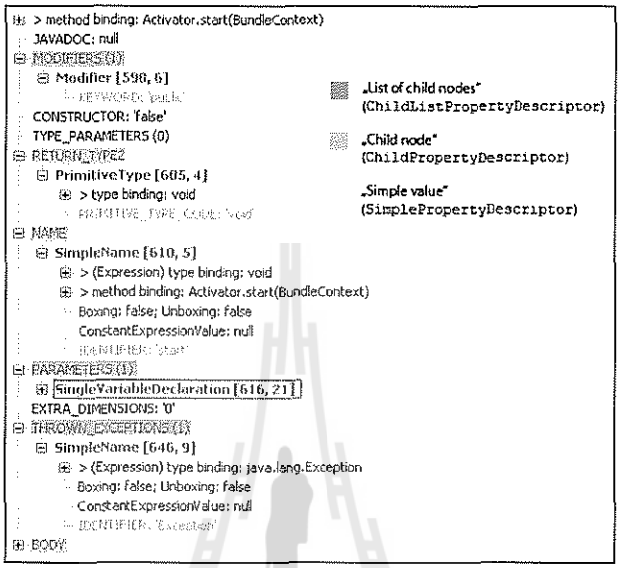
2.5.3 ASTNode

ASTNode เป็นคลาสนามธรรม(abstract class) ที่เป็นคลาสแม่(super class) ของโหนดทุกประเภทใน Abstract Syntax Tree แต่ละโหนดของ AST จะแสดงโครงสร้างของซอร์สโค้ดภาษาจาวา (IBM Corporation and other, 2000) เช่น ชื่อ, ประเภท, expression, statement หรือ declaration ASTNode จะอนุญาตให้เข้าถึงแต่ละโหนดลูก(child node) ได้โดยการใช้ visitor pattern

```
ASTNode root= parser.createAST(null);
root.accept(new ASTVisitor() {

  public boolean visit(CastExpression node) {
    fCastCount++;
    return true;
  }
```

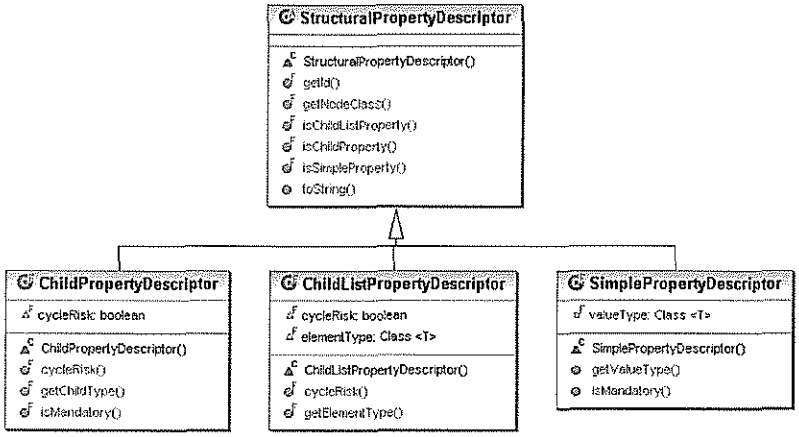
รูปที่ 2.9 ตัวอย่างการใช้ ASTNode และการใช้ ASTVisitor เพื่อเข้าถึงข้อมูลของโหนดในแต่ละโหนด

ในทุกๆคลาสย่อยของ ASTNode จะรวบรวมข้อมูลเฉพาะของแต่ละส่วนประกอบของจาวา เช่น ใน MethodDeclaration จะมีข้อมูลเกี่ยวกับชื่อ return type parameter เป็นต้น ซึ่งข้อมูลของโหนดจะอ้างอิงถึง structural properties



รูปที่ 2.10 Structural properties ของ MethodDeclaration

ในการที่จะเข้าถึงข้อมูลในโหนดของ structural properties สามารถทำได้โดยการใช้ static method หรือ generic method สำหรับ structural properties สามารถทำการจัดกลุ่มได้โดยแบ่งเป็น 3 ประเภท คือ ประเภทที่เก็บค่าพื้นฐาน ประเภทที่เก็บ single child AST node และประเภทที่เก็บรายการของ child AST node (Thomas Kuhn และ Olivier Thoman, 2006)

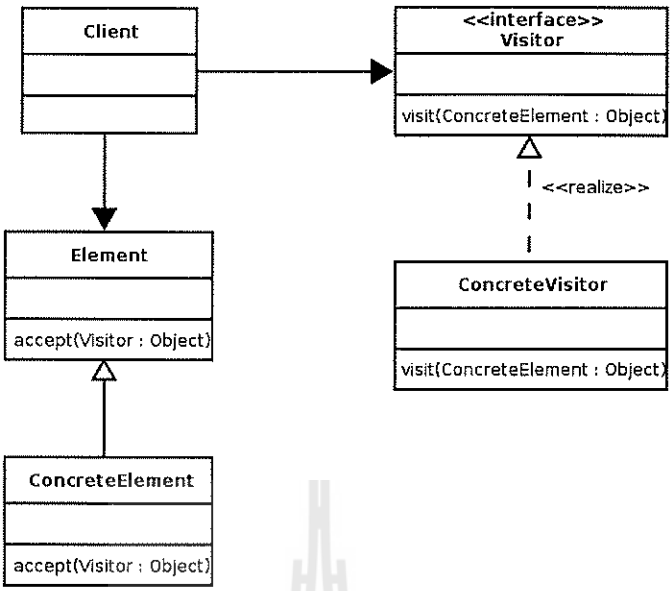


รูปที่ 2.11 คลาสไดอะแกรมแสดงการแยกประเภทของ structural properties

2.5.4 ASTVisitor

ASTVisitor เป็น visitor pattern ที่ใช้สำหรับในการเข้าถึงข้อมูลในแต่ละโหนดของ ASTNode โดยในการเข้าถึงข้อมูลนั้น จะมีการเรียกใช้เมธอดสองเมธอดซึ่งก็คือ visit() และ endVisit() นอกจากนั้นแล้ว ASTVisitor ยังสามารถประกาศเมธอดได้อีกสองเมธอดนั่นก็คือ preVisit(ASTNode node) และ postVisit(ASTNode node) คลาสย่อย(sub class) ของ ASTVisitor จะทำการส่งผ่านโหนดต่างๆของ AST โดยการวนตามลำดับของโครงสร้างต้นไม้และเรียกใช้เมธอดของ ASTVisitor สำหรับทุกๆ ASTNode (IBM Corporation and other, 2000)

ในการเขียนโปรแกรมเชิงวัตถุและวิศวกรรมซอฟต์แวร์นั้นการออกแบบ visitor pattern ถือว่าเป็นแนวทางในการแยกอัลกอริทึมออกจากโครงสร้างของวัตถุที่กำกับการทำงาน ในการแยกอัลกอริทึมออกจากโครงสร้างวัตถุแบบนี้ ส่งผลให้เกิดความสามารถในการเพิ่มการดำเนินการให้แก่โครงสร้างที่มีอยู่แล้วโดยไม่ต้องทำการแก้ไขในส่วนของโครงสร้าง ซึ่งการทำงานในลักษณะนี้ถือว่าเป็นอีกวิธีในการทำงานตามหลักการ open/close ทั้งนี้ visitor อนุญาตให้สามารถทำการเพิ่ม virtual function ให้แก่กลุ่มของคลาสที่มีอยู่โดยที่ไม่ต้องทำการแก้ไขกลุ่มของคลาสนั้นๆ ทั้งนี้ยังสามารถสร้าง visitor คลาสที่สามารถเลือกใช้งานเฉพาะ virtual function ที่มีความเหมาะสมได้ โดย visitor อาจจะต้องใช้ instance ที่อ้างถึงเพื่อเป็นข้อมูลและทำการ implement ผ่านทาง double dispatch เพื่อให้ได้ตรงตามเป้าหมาย



รูปที่ 2.12 UML Diagram ของ visitor pattern

Visitor pattern ต้องการภาษาที่รองรับ single dispatch และการทำ method overloading ภายใต้เงื่อนไข (Wikipedia, 2012) หากพิจารณาวัตถุสองวัตถุ ซึ่งแต่ละวัตถุเป็นวัตถุของคลาสแต่ละประเภท โดยวัตถุหนึ่งเรียกว่า element และอีกวัตถุเรียกว่า visitor วัตถุที่เป็น element มี accept() method ที่สามารถทำให้ visitor เป็น argument เมื่อ accept() method เรียกใช้ visit() method กับ visitor วัตถุที่เป็น element จะสามารถทำให้ตัวเองกลายเป็น argument บน visitor method เพราะฉะนั้น

- เมื่อเรียกใช้ accept() method ในโปรแกรม การ implement จะขึ้นอยู่กับทั้ง dynamic type ของ element และ static type ของ visitor

- เมื่อมีการเรียกใช้ associated visit() method การ implement จะขึ้นอยู่กับทั้ง dynamic type ของ visitor และ static type ของ element ที่ทราบจากการ implement accept() method ซึ่งมีผลเช่นเดียวกับ dynamic type ของ element

ด้วยเหตุนี้ การ implement visit() method จะขึ้นอยู่กับทั้ง

- dynamic type ของ element และ dynamic type ของ visitor

ในภาษาจาวามีการเขียนถึงเทคนิคสองเทคนิคที่ใช้ reflection เพื่อลดความซ้ำซ้อนของกระบวนการ double dispatch simulation ใน visitor pattern เริ่มด้วย JDK 1.2 ความจำเป็นที่ต้องมี

accept() method นั้น สามารถถูกกำจัดด้วยการเพิ่ม getMethod() method ซึ่งจะช่วยให้วัตถุสามารถเข้าถึง method ของอีกวัตถุหนึ่ง โดยขึ้นกับการระบุประเภทของ parameter ทั้งนั้น invoke() method จะสามารถช่วยให้เกิดการเรียกใช้ method เหล่านี้ได้

2.6 งานวิจัยที่เกี่ยวข้อง

(Brian de Alwis และ Jonathan Sillito, 2009) ได้นำเสนอความแตกต่างระหว่าง Centralized Version Control System กับ Decentralized Version Control Systems ซึ่งเป็นระบบควบคุมเวอร์ชั่นโดยได้อธิบายถึงความแตกต่างของระบบควบคุมเวอร์ชั่นทั้งสองระบบและประโยชน์ของการย้ายโปรเจคจากระบบควบคุมเวอร์ชั่นแบบรวมศูนย์กลาง (Centralized) มาเป็นแบบกระจายศูนย์กลาง (Decentralized) ระบบควบคุมเวอร์ชั่นแบบกระจายศูนย์กลางเป็นระบบควบคุมเวอร์ชั่นที่ได้ทำการพัฒนามาจากระบบควบคุมเวอร์ชั่นแบบรวมศูนย์กลาง โดยได้มีการปรับปรุงจากข้อจำกัดบางอย่างของระบบควบคุมเวอร์ชั่นแบบรวมศูนย์กลาง และเพื่อสนับสนุนการทำงานแบบกระจายศูนย์กลางหลักการทำงานของระบบควบคุมเวอร์ชั่นแบบกระจายศูนย์กลางจะแตกต่างจากระบบควบคุมเวอร์ชั่นแบบรวมศูนย์กลางตรงที่เมื่อผู้พัฒนาได้ทำการนำข้อมูลออกมาจากศูนย์กลางการเก็บข้อมูลหลักระบบจะอนุญาตให้ผู้พัฒนาสามารถสร้างศูนย์กลางการเก็บข้อมูลให้เป็นของผู้พัฒนาเองได้เพื่อเป็นการควบคุมการเปลี่ยนแปลงที่เกิดขึ้นในระหว่างการพัฒนาบนคอมพิวเตอร์ของผู้พัฒนา

ในการระบุหาความแตกต่างระหว่างรุ่นของข้อมูลจะมีวิธีการระบุที่แตกต่างกันขึ้นอยู่กับชนิดของซอฟต์แวร์ที่ใช้ ในงานวิจัยของ (Miryung Kim และ David Notkin, 2006) ได้นำเสนองานวิจัยที่รวบรวมและสำรวจเกี่ยวกับความสามารถและคุณสมบัติของ matching technique ซึ่งเป็นเทคนิคที่ใช้ในการวิเคราะห์เวอร์ชั่นของซอฟต์แวร์โดยได้อธิบายเกี่ยวกับผลกระทบของการนำแต่ละ matching technique ไปปรับใช้และการเลือกแต่ละ matching technique มีผลต่อประสิทธิภาพและความถูกต้องอย่างไร โดยสร้าง framework ขึ้นมาเพื่อทดสอบและประเมินผลสำหรับแต่ละ matching technique โดย matching technique ที่ Miryung Kim และ David Notkin ได้ทำการสำรวจมีดังนี้

- Entity name Matching
- String Matching
- Syntax Tree Matching
- Control Flow Graphing Matching
- Program Dependence Graph Matching

- Binary Code Matching
- Clone Detection
- Origin Analysis Tools

Program Representation	Citation	Granularity	Assumed Correspondence	Multiplicity	Heuristics			Application
					N	P	S	
A set of entities	[20, 15, 37]	Module		1:1	√			Fault proneness
	Bevan et al. [11]	File		1:1	√			Instability
	Ying et al. [48]	File		1:1	√			Co-change
	Zimmermann et al. [51]	File Procedure Field		1:1	√			
String	*diff* [25]	Line	File	1:1			√	Merging Clone genealogy [29] Fix inducing code [43]
	bdiff [45]	Line	File	1:n			√	Merging
AST	*cdiff* [47]	AST Node	Procedure	1:1	√			
	Neamtiu et al.[38]	Type, Var		1:1	√	√		Type change
	Hunt, Tichy[24, 35]	Token	File	1:1		√	√	Merging
CFG	*Jdiff* [5]	CFG node		1:1	√		√	Regression testing Impact analysis
Binary	BMAT [46]	Code block		1:1 (procedure) n:1 (block)	√	√	√	Profile propagation Regression testing
Hybrid	Zou, Godfrey [52]	Procedure		1:1 or 1:n or n:1	√		√	Origin analysis
	Kim et al. [30]	Procedure		1:1	√		√	Signature change [31] Renaming analysis

N: Name-based heuristics, P: Position-based heuristics, S: Similarity-based heuristics

ตารางที่ 2.1 ตารางแสดงการเปรียบเทียบ Matching Techniques

ในงานวิจัยของ (Yang, 1991) ได้นำเสนอ AST(Abstract Syntax Tree) difference algorithm เพื่อใช้ในการทำ Software version merging โดยการสร้าง AST ของฟังก์ชั่นสองฟังก์ชั่น จากนั้นนำรูทโหนดของแต่ละ AST มาทำการหาความเหมือนโดยถ้าหากรูทโหนดมีความเหมือนก็จะทำการจัดเรียงโหนดย่อยโดยใช้ LCS algorithm แล้วทำการวนหาความเหมือนของ subtree

(Iulian Neamtiu, Jeffrey S. Foster และ Michail Hicks, 2005) ได้นำเสนองานวิจัยที่ใช้ AST มาช่วยวิเคราะห์หาความแตกต่างใน Dynamic software updating ในงานวิจัยได้ทำการสร้างเครื่องมืออย่างง่ายเพื่อช่วยวิเคราะห์หาความแตกต่างของตัวแปร, ประเภทของตัวแปร, และฟังก์ชั่น โดยอัลกอริทึมที่ใช้จะทำการวนไปในแต่ละโหนดของ AST แบบคู่ขนานและทำการ map แบบ one to one

(Tobias Sager, Abraham Bernstein, Nartin Pinzger และ Chistoph Kiefer, 2006) ได้นำเสนองานวิจัยที่ใช้ Abstract Syntax Tree ร่วมกับ FAMIX ซึ่งเป็น programming language independence model สำหรับแสดง source code แบบ object-oriented เพื่อเปรียบเทียบหาความเหมือนกันระหว่างคลาสสองคลาสในโปรแกรมภาษาจาวาซึ่งเป็นโปรแกรมแบบ object-oriented โดยงานวิจัยนี้ได้ทำการ

สร้างส่วนเสริมของโปรแกรม Eclipse โดยลำดับแรกแปลงจาก Abstract Syntax Tree ที่ได้จาก ASTParser ไปเป็น FAMIX จากนั้นจึงใช้ bottom-up maximum common subtree isomorphism, top-down maximum common subtree isomorphism, tree edit distance ในการหาความเหมือนระหว่างคลาสของภาษาจาวาแล้วประเมินผลเพื่อหาอัลกอริทึมที่ดีที่สุดในการเปรียบเทียบ Abstract Syntax Tree โดยใช้กรณีทดสอบเป็นจาวาโปรเจคที่มีใช้ในงานจริงๆเป็นกรณีทดสอบเพื่อวัดผลการทดลอง

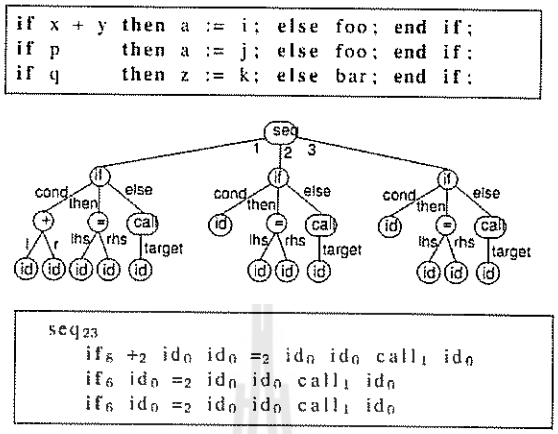
นอกจากนี้แล้วยังมีงานวิจัยที่ได้นำ Abstract Syntax Tree ไปประยุกต์ร่วมกับการวิเคราะห์แบบ Token base เพื่อเป็นการลดข้อด้อยของการ matching ด้วย Abstract Syntax Tree

AST Node	*FAMIX Element*
-	FAMIXInstance
-	Model
PackageDeclaration	Package
TypeDeclaration	Class
-	InheritanceDefinition
FieldDeclaration	Attribute
MethodDeclaration	Method
SingleVariableDeclaration	FormalParameter
SingleVariableDeclaration	LocalVariable
ConstructorInvocation, SuperConstructorInvocation, ClassInstanceCreation, MethodInvocation, SuperMethodInvocation	Invocation
FieldAccess, SuperFieldAccess, SimpleName, QualifiedName	Access

ตารางที่ 2.2 ตารางแสดงความสัมพันธ์ระหว่าง AST node กับ FAMIX element

ในปี 2006 (Rainer Koschke, raimar Falke และ Pierre Fenzel, 2006) ได้นำเสนองานวิจัยที่ใช้ suffix tree มาใช้ในการหา clone ใน Abstract Syntax Tree โดยใช้ suffix tree detection อัลกอริทึมในการหา clone ในงานวิจัยได้กล่าวไว้ว่า suffix tree เป็น String ที่แสดงจาก root node ไปยัง leaf node โดยปกติ suffix tree detection จะเป็นอัลกอริทึมแบบ base on token งานวิจัยนี้ได้ประยุกต์โดยการจัดเรียง AST node ที่ได้จากการ parse source code ให้อยู่ในรูปแบบที่ต้องการดังภาพที่ 2.11 จากนั้นแปลง AST node แต่ละ node ให้อยู่ในรูปของ String base โดยใช้ Ukkonen อัลกอริทึม แล้วจากนั้นจึง

ใช้ Baker อัลกอริทึมซึ่งเป็น suffix tree detection อัลกอริทึมในการหา clone นอกจากนี้แล้วในงานวิจัยนี้ยังได้เปรียบเทียบข้อดีข้อเสียของการวิเคราะห์แบบ Token base กับ AST base



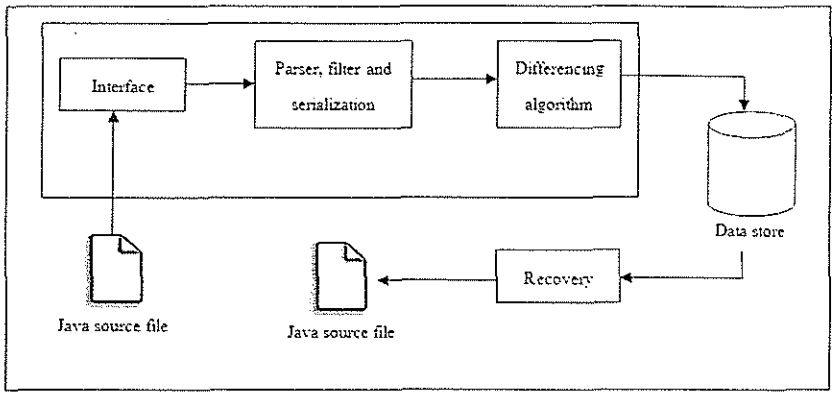
รูปที่ 2.13 ตัวอย่างซอร์สโค้ดและการจัดเรียงเพื่อใช้ในการหาความ clone

บทที่ 3
การออกแบบและพัฒนาเครื่องมือเพื่อช่วยในการกู้คืนโค้ดสำหรับภาษาจาวา

งานวิจัยนี้มุ่งเน้นที่จะพัฒนาเครื่องมือเพื่อช่วยในการกู้คืนโค้ดสำหรับภาษาจาวา เพื่อเป็นการอำนวยความสะดวกและเพิ่มประสิทธิภาพในการจัดเก็บข้อมูลในระหว่างการพัฒนาซอฟต์แวร์ ในงานวิจัยนี้จะพัฒนาเครื่องมือโดยเครื่องมือจะทำการนำเข้าซอร์สโค้ดที่เป็นภาษาจาวา โดยผู้ใช้สามารถเลือกเวอร์ชั่นของซอร์สโค้ดที่จะทำการเปรียบเทียบ หลังจากนั้นเครื่องมือจะทำการวิเคราะห์และเปรียบเทียบซอร์สโค้ดที่ผู้ใช้ได้ทำการนำเข้าหาข้อแตกต่างเพื่อให้ผู้ใช้ทำการพิจารณาว่าจะทำการสำรองโค้ดหรือทำการกู้คืนโค้ด โดยการนำเสนอข้อแตกต่างจะแสดงอยู่ในรูปแบบของโครงสร้างต้นไม้เพื่อความง่ายในการเปรียบเทียบความแตกต่าง

3.1 องค์ประกอบหลักของเครื่องมือ

ในการพัฒนาเครื่องมือเพื่อช่วยในการกู้คืนโค้ดสำหรับภาษาจาวาได้ทำการแบ่งส่วนของการพัฒนาเครื่องมือออกแบบ 5 ส่วนใหญ่ๆคือ ส่วนติดต่อกับผู้ใช้งาน(interface) ส่วนการกระจายนิพจน์ (Parser, filter and serialization) ส่วนการเปรียบเทียบหาความแตกต่างของซอร์สโค้ด(Differencing algorithm) ส่วนของการสำรองและการเก็บข้อมูล(Back up and Data Store) และส่วนของการกู้คืน (Recovery)



รูปที่ 3.1 แผนภาพองค์ประกอบโดยรวมของเครื่องมือ

การพัฒนาเครื่องมือเพื่อช่วยกู้คืนโค้ดสำหรับภาษาจาวา จะถูกพัฒนาโดยภาษาจาวาซึ่งจะอยู่ในรูปแบบของ GUIs (Graphic User Interface) เพื่อให้ง่ายต่อการใช้งานและมีรูปแบบการแสดงผลที่สวยงาม โดยแต่ละส่วนจะมีรายละเอียดในการพัฒนาดังนี้

3.1.1 ส่วนติดต่อผู้ใช้งาน(User Interface)

ในส่วนติดต่อกับผู้ใช้งานนี้จะถูกพัฒนาโดยภาษาจาวาซึ่งจะอยู่ในรูปแบบของ GUIs (Graphic User Interface) เพื่อง่ายต่อการใช้งาน โดยมีส่วนการแบ่งของการทำงานเป็นสองส่วนคือ ในส่วนการรับเข้าซอร์สโค้ดและกู้คืนซอร์สโค้ด โดยในส่วนรับเข้าซอร์สโค้ดนั้นจะเป็นส่วนที่รับเข้าซอร์สโค้ดสำหรับทำการเปรียบเทียบและแสดงความแตกต่างของซอร์สโค้ดโดยความแตกต่างจะแสดงในรูปแบบของโครงสร้างต้นไม้เพื่อให้ง่ายต่อการแสดงผลการเปรียบเทียบ และสำหรับในส่วนของการกู้คืนซอร์สโค้ดจะแสดงแฟ้มข้อมูลของซอร์สโค้ดที่ได้มีการเปรียบเทียบความแตกต่างโดยจะมีการแสดงในรูปแบบของโครงสร้างต้นไม้และมีการระบุวันเวลาที่ทำการเปรียบเทียบเพื่อให้ง่ายต่อการค้นหาซอร์สโค้ดในเวอร์ชั่นที่ผู้ใช้งานต้องการ.

3.1.2 ส่วนการกระจายนิพจน์(Parser, filter and serialization)

ส่วนการกระจายนิพจน์ใช้ ASTParser ซึ่งเป็นคลาสที่ใช้ในการกระจายนิพจน์ของภาษาจาวาในการทำการกระจายนิพจย์ของจาวาคลาสให้อยู่ในรูปแบบโครงสร้างต้นไม้ เพื่อที่จะนำโหนดของโครงสร้างต้นไม้แต่ละส่วนไปทำการเปรียบเทียบ ในการกระจายนิพจน์ของภาษาจาวานั้น โค้ดที่ทำนำเข้ามาต้องเป็นโค้ดที่มีความถูกต้องตามหลักการเขียนของภาษาจาวาจากนั้น ASTparser ซึ่งเป็น

คลาสที่ใช้ทำการกระจายนิพจน์จะทำการสร้างโครงสร้างต้นไม้จากโค้ดที่นำเข้าโดยโค้ดที่ผ่านการกระจายนิพจน์จะอยู่ในรูปแบบของโหนด ซึ่งแต่ละโหนดจะประกอบด้วยข้อมูลที่เกี่ยวข้องกับซอร์สโค้ดภาษาจาวา เช่น ชื่อของตัวแปร ประเภทของตัวแปล การประกาศตัวแปร เป็นต้น

3.1.3 ส่วนการเปรียบเทียบหาความแตกต่างของซอร์สโค้ด(Differencing algorithm)

ในส่วนนี้เป็นการทำการเปรียบเทียบซอร์สโค้ดที่รับเข้ามาเพื่อหาความแตกต่างระหว่างซอร์สโค้ดและทำการแยกส่วนของซอร์สโค้ดที่มีความแตกต่างกันในระหว่างเวอร์ชั่นเพื่อส่งต่อไปยังส่วนของการสำรองและการเก็บข้อมูล

3.1.4 ส่วนการสำรองและการจัดเก็บข้อมูล(Back up and Data Store)

ในส่วนการสำรองและการเก็บข้อมูลเมื่อเครื่องมือทำการเปรียบเทียบเพื่อหาความแตกต่างของซอร์สโค้ดเสร็จสิ้นแล้ว จะทำการส่งผลลัพธ์ที่ได้จากการเปรียบเทียบมายังส่วนนี้โดยในส่วนนี้เครื่องมือจะทำการจัดรูปแบบของข้อมูลที่ส่งมาให้อยู่ในรูปแบบที่ได้กำหนดขึ้นจากนั้นทำการบันทึกข้อมูลที่ได้ทำการจัดรูปแบบแล้วลงใน text file

3.1.5 ส่วนการกู้คืน(Recovery)

ส่วนของการกู้คืนเป็นการเรียกข้อมูลจาก text file ที่ได้ทำการบันทึกไว้โดยส่วนของการสำรอง แล้วนำมาทำการกระจายนิพจน์เพื่อนำมาทำการผสานกับซอร์สโค้ดต้นฉบับเพื่อที่จะได้ ซอร์สโค้ดในเวอร์ชั่นที่ผู้ใช้ต้องการ

3.2 การออกแบบเครื่องมือ

3.2.1 ส่วนติดต่อผู้ใช้งาน(User Interface)

ในการออกแบบ Interface ซึ่งเป็นส่วนติดต่อกับผู้ใช้งานเครื่องมือ ทางผู้วิจัยได้ทำการออกแบบ Interface ให้อยู่ในรูปของ GUIs(Graphic User Interfaces) เพื่อให้เกิดความสะดวกต่อผู้ใช้งานเครื่องมือ โดยรูปแบบของ GUIs จะแบ่งการทำงานออกเป็นสองส่วน คือ ส่วนสำหรับการนำเข้าซอร์สโค้ดเพื่อทำการเปรียบเทียบ และส่วนของการกู้คืนข้อมูล โดยในการออกแบบ GUIs จะใช้ tab control ในการแบ่งแยกการทำงานของทั้งสองส่วน โดยคลาสที่เกี่ยวข้องกับส่วนของ GUIs จะถูกรวบรวมไว้ในแพ็คเกจ mainGUI เพื่อให้ง่ายต่อการเรียกใช้งานและการจัดการ

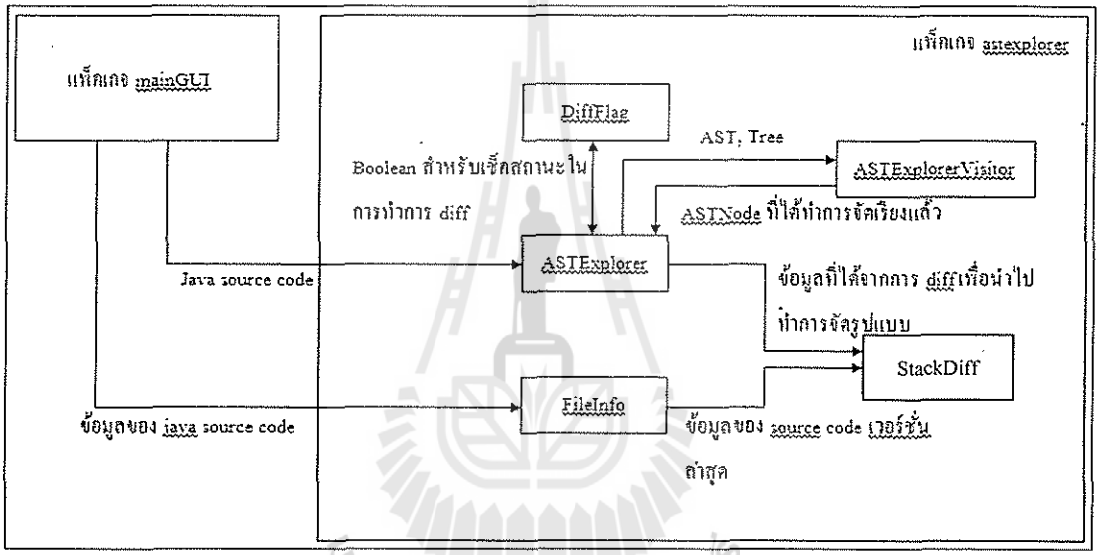
3.2.2 ส่วนการกระจายนิพจน์(Parser, filter and serialization)

ในส่วนของการกระจายนิพจน์ทางผู้วิจัยเลือกใช้ ASTParser คลาส ซึ่งเป็นคลาสสำเร็จรูปที่ใช้สำหรับทำการกระจายนิพจน์ของภาษาจาวา โดยการทำงานจะทำงานเมื่อผู้ใช้งานทำการ

นำเข้าซอร์สโค้ดที่จะทำการเปรียบเทียบ ในการจัดรูปแบบจะใช้ visitor pattern ในการทำการจัดรูปแบบ โดยนำ Tree มาใช้เพื่อการจัดเรียงข้อมูลและทำการนำส่งไปยังส่วนของ Interface เพื่อทำการแสดงผล โดยในการทำงานของการกระจายนิพจน์ การกรองข้อมูลและการจัดเรียง จะถูกรวบรวมไว้ในแพ็คเกจ astexplorer

3.2.2.1 แพ็คเกจ astexplorer

ในแพ็คเกจนี้จะประกอบด้วยคลาสต่างๆที่เกี่ยวข้องการกับทำการกระจายนิพจน์ ซึ่งรวมไปถึงการนำเข้าซอร์สโค้ดสำหรับการกระจายนิพจน์ด้วย ภายในแพ็คเกจจะประกอบด้วยคลาสทั้งหมด 5 คลาส



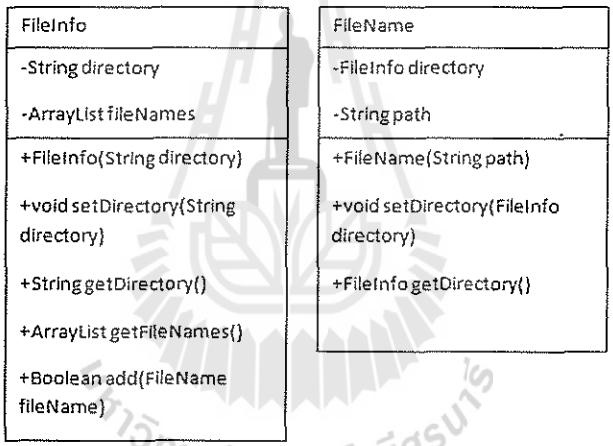
รูปที่ 3.2 แผนภาพการทำงานของคลาสภายในแพ็คเกจ astexplorer

รูปที่ 3.2 แสดงการทำงานของคลาสที่อยู่ภายในแพ็คเกจซึ่งในแพ็คเกจจะประกอบด้วยคลาสทั้งหมด 5 คลาส โดยในการทำงาน แพ็คเกจ mainGUI จะทำการส่งซอร์สโค้ดของภาษาจาวาที่จะทำการกระจายเข้ามาภายในคลาส ASTExplorer โดยภายในคลาสนี้จะทำการกระจายนิพจน์ของซอร์สโค้ดที่ส่งเข้ามาจากนั้นทำการส่งข้อมูลที่ได้จากการกระจายนิพจน์ต่อไปยังคลาส ASTExplorerVisitor เพื่อทำการจัดเรียงโหนดต่างๆของ AST ให้อยู่ในรูปแบบ Tree และส่งกลับคืนไปยังคลาส ASTExplorer จากนั้นเมื่อได้ Tree ซึ่งได้ทำการจัดเรียงโหนดในรูปแบบของ Tree แล้ว

ASTExplorer จะทำการส่ง Tree ที่ได้ทำการจัดเรียงโหนดไปยังแพ็คเกจ mainGUI เพื่อสำหรับแสดงผล นอกจากนี้ยังทำการส่ง Tree ที่ได้จัดเรียงข้อมูลแล้วไปยังเมธอดเพื่อทำการเปรียบเทียบหาความแตกต่างของ Tree ซึ่งข้อมูลที่ได้จากการเปรียบเทียบนั้น จะทำการส่งต่อไปยังคลาส StackDiff เพื่อทำการจัดรูปแบบและเขียนลงใน Text file สำหรับคลาส DiffFlag นั้น เป็นคลาสสำหรับทำการเช็คการเปรียบเทียบ โดยจะมีการส่งข้อมูลเป็น Boolean จากคลาส ASTExplorer มายังคลาส DiffFlag เพื่อทำการเช็คสถานะของการเปรียบเทียบ เมื่อทำการเช็คสถานะเสร็จแล้วก็จะทำการส่งข้อมูลของสถานะกลับไปยังคลาส ASTExplorer เพื่อให้ทำงานต่อไปตามเงื่อนไขที่ได้กำหนดไว้

3.2.2.2 แพ็คเกจ fileModel

ในแพ็คเกจนี้ประกอบด้วยคลาสทั้งหมด 3 คลาส ซึ่งออกแบบมาสำหรับทำการเก็บข้อมูลของซอร์สโค้ดที่ทำการรับเข้ามา โดยข้อมูลของซอร์สโค้ดประกอบด้วย ชื่อแฟ้มเอกสาร ชื่อเอกสาร ที่อยู่ของเอกสาร ซึ่งข้อมูลเหล่านี้จะถูกเรียกใช้งานจากแพ็คเกจอื่นๆ



รูปที่ 3.3 องค์ประกอบของคลาสภายในแพ็คเกจ fileModel

จากรูปที่ 3.3 คลาส FileInfo และคลาส FileName เป็นคลาส getter – setter ที่ใช้สำหรับการเก็บข้อมูลของซอร์สโค้ด นอกจากคลาสในรูปที่ 3.3 แล้ว ในแพ็คเกจ fileModel ยังมีคลาส ListOfFileModel ซึ่งการทำงานภายในคลาสนี้ จะเป็นการรวบรวมข้อมูลที่ได้จากการเรียกใช้คลาสในรูป 3.3 ให้อยู่ในรูปของ List สำหรับนำไปใช้งานในแพ็คเกจอื่นๆ

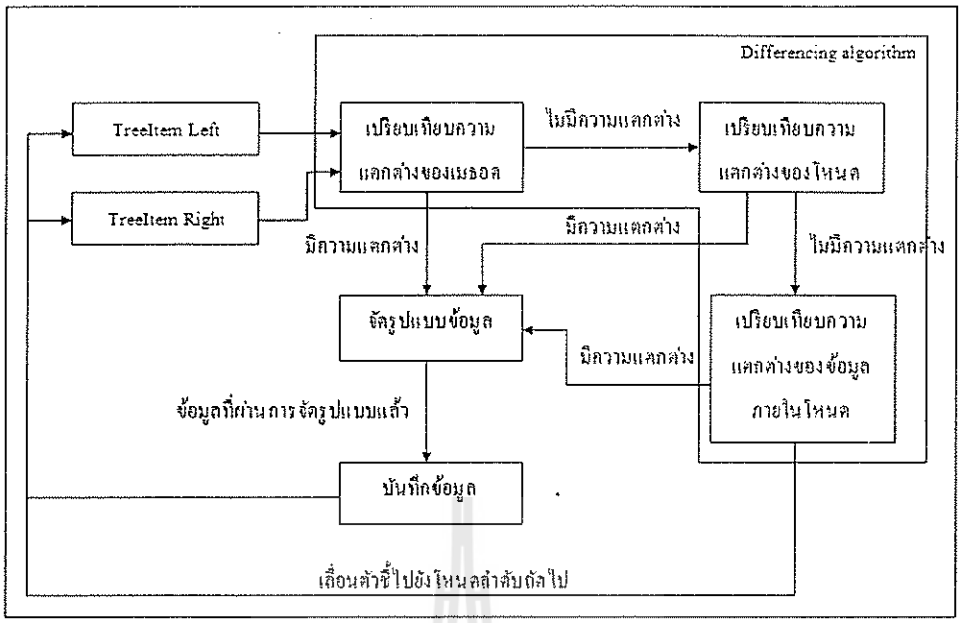
3.2.3 ส่วนการเปรียบเทียบหาความแตกต่างของซอร์สโค้ด(Differencing algorithm)

สำหรับการเปรียบเทียบหาความแตกต่างของซอร์สโค้ดนั้น การเปรียบเทียบจะเป็นการเปรียบเทียบโดยการวิเคราะห์โครงสร้างต้นไม้ของไวยากรณ์ภาษาที่ถูกจัดเรียงแล้ว โดยข้อมูลสำหรับการเปรียบเทียบจะถูกส่งเข้าเมธอด treeDiffTravel() ภายในเมธอดจะมีการรับพารามิเตอร์ต่างๆดังนี้

- TreeItem left, TreeItem right เป็น TreeItem ที่ผ่านการจัดเรียงข้อมูลจากการกระจายนิพจน์ ใช้เพื่อสำหรับทำการเปรียบเทียบหาความแตกต่างระหว่างซอร์สโค้ด
- ASTExplorer astExplorerL, ASTExplorer astExplorerR ใช้สำหรับการอ้างถึง ตัวแปรในระดับคลาสเพื่อนำไปใช้สำหรับในการแสดงผล เมื่อทำการเปรียบเทียบเสร็จสิ้น
- StyleRange styleRange ใช้สำหรับจัดรูปแบบของผลการเปรียบเทียบ
- StackDiff diff ใช้สำหรับในการอ้างถึงเมธอดภายในคลาส StackDiff เพื่อใช้ในการจัดรูปแบบของข้อมูลที่ได้จากการเปรียบเทียบก่อนทำการบันทึก
- DiffFlag df เป็นตัวแปรที่ใช้ในการเช็คสถานะในระหว่างการทำการเปรียบเทียบ

รูปแบบการเปรียบเทียบจะแบ่งออกเป็นสองระดับคือ การเปรียบเทียบหาความแตกต่างระดับเมธอดและระดับโหนด โดยในการเปรียบเทียบจะทำการเปรียบเทียบโหนดต่อโหนดในแบบ 1:1

เนื่องจากในการใช้ Tree ทำให้การมองเห็นข้อมูลสามารถมองเห็นได้ทีละชั้นของโครงสร้างต้นไม้ ดังนั้นในการเปรียบเทียบนั้นจำเป็นต้องทำการเรียกเมธอดแบบ Recursive เพื่อให้สามารถทำการมองเห็นข้อมูลและทำการเปรียบเทียบในระดับชั้นถัดไปได้



รูปที่ 3.4 การทำงานของการเปรียบเทียบหาความแตกต่าง

ในการเปรียบเทียบหาความแตกต่าง เริ่มต้นด้วยการส่ง Tree ที่ได้ทำการจัดเรียงข้อมูลที่ได้จากการกระจายนิพจน์ไปยังเมธอดที่สร้างไว้ จากนั้นจะทำการเริ่มเปรียบเทียบ โดยทำการเปรียบเทียบในส่วนของข้อมูลที่อยู่ในระดับเมธอดก่อน ถ้าเมธอดที่เปรียบเทียบใน Tree ของทั้งสองฝั่งไม่มีความแตกต่าง จะทำการเข้าไปเปรียบเทียบข้อมูลภายในโหนด ถ้าหากในการเปรียบเทียบในระดับเมธอดมีความแตกต่าง จะทำการส่งข้อมูลความแตกต่างที่ได้ไปทำการจัดรูปแบบและทำการบันทึก นอกจากนี้จะทำการเลื่อนตัวชี้ตำแหน่งไปยังเมธอดถัดไปโดยจะไม่สนใจโหนดลูกที่อยู่ภายในเมธอดที่มีความแตกต่าง สำหรับเมธอดที่ทำการเปรียบเทียบและไม่มีความแตกต่าง จะทำการเข้าไปเปรียบเทียบในระดับโหนดภายในเมธอด โดยในการเปรียบเทียบในระดับโหนดภายในเมธอดนั้น จะมีการเช็คประเภทของโหนดซึ่งถ้าประเภทของโหนดต่างกันจะทำการส่งข้อมูลไปจัดรูปแบบและทำการบันทึก นอกจากนี้จะทำการเลื่อนตัวชี้ตำแหน่งไปยังโหนดถัดไป แต่ถ้าประเภทของโหนดไม่มีความแตกต่างจะทำการเข้าไปเช็คยังค่าภายในโหนดเพื่อหาความแตกต่างอีกที

3.2.4 ส่วนการสำรองและการจัดเก็บข้อมูล(Back up and Data Store)

ในการจัดเก็บบันทึกส่วนต่างของข้อมูลที่ได้จากการเปรียบเทียบ เพื่อให้ง่ายต่อการติดตั้งเครื่องมือ การใช้งานและง่ายต่อการกู้คืนข้อมูล การเก็บบันทึกข้อมูลจึงบันทึกข้อมูลให้อยู่ในรูปแบบของ text file โดยข้อมูลที่จะทำการบันทึกลง text file จะต้องมีการจัดให้อยู่ในรูปแบบที่กำหนด โดยรูปแบบที่กำหนดคือ [Line][line][Label]data[End Label]

- [Line][line] คือการบ่งบอกตำแหน่งบรรทัดเริ่มต้นของข้อมูลที่ทำการเก็บเข้ามา ตัวอย่างเช่น [Line][2] เป็นการบ่งบอกว่าข้อมูลเริ่มต้นที่บรรทัดที่ 2

- [Label][End Label] คือการบอกประเภทของการเปลี่ยนแปลงที่เกิดขึ้นในการเปรียบเทียบหาความแตกต่างของซอร์สโค้ด โดยประเภทของความแตกต่างจะแบ่งออกได้เป็น

- [Method Modify][End Method Modify] คือประเภทของความแตกต่างที่เกิดขึ้นในระดับ Method

- [Node Modify][End Node Modify] คือประเภทของความแตกต่างที่เกิดขึ้นในระดับ node

- [Add][End Add] คือประเภทของความแตกต่างที่เกิดขึ้นจากการที่มีส่วนของซอร์สโค้ดได้ทำการเพิ่มเข้ามา โดยรวมถึงการเพิ่มเข้ามาของซอร์สโค้ดในระดับ Method และในระดับ node

- [Delete][End Delete] คือประเภทของความแตกต่างที่เกิดขึ้นจากการที่มีส่วนของซอร์สโค้ดถูกลบออกไป โดยรวมถึงการเพิ่มเข้ามาของซอร์สโค้ดในระดับ Method และในระดับ node

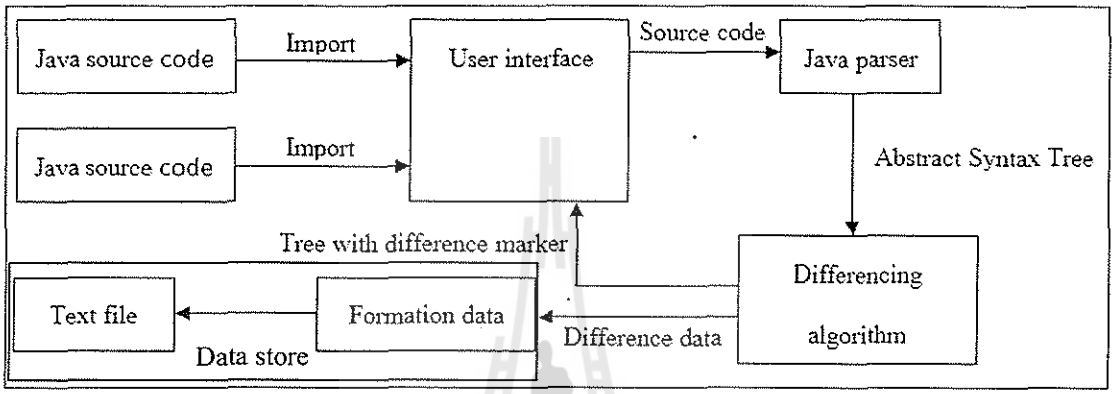
- data คือข้อมูลที่ได้จากการเปรียบเทียบหาความแตกต่าง

3.2.5 ส่วนการกู้คืน(Recovery)

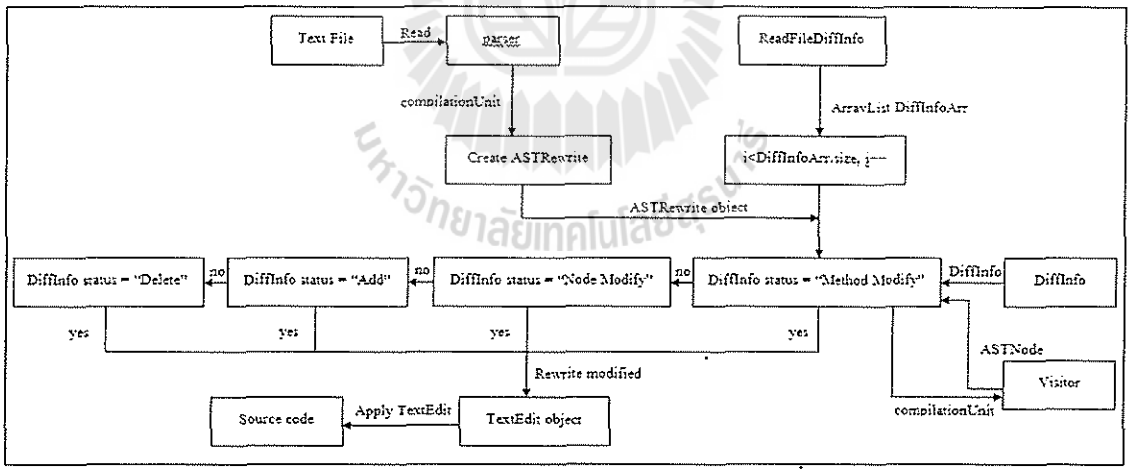
ในส่วนของการกู้คืนข้อมูล จะทำการรับเข้า text file ที่ใช้เก็บข้อมูลที่ได้จากการเปรียบเทียบแล้วนำมาทำการแยกข้อมูลในส่วนของ Label กับ data ออกจากกัน จากนั้นทำการนำข้อมูลในส่วนของ data ไปทำการกระจายนิพจน์เพื่อให้อยู่ในรูปแบบของ ASTNode และส่งต่อไปยังการทำงานหลักภายในส่วนของการกู้คืน โดยในการทำงานหลักจะตรวจสอบข้อมูล Labal ที่รับเข้ามา จากนั้นจะทำการนำเอา ASTNode ไปผสานกับซอร์สโค้ดเวอร์ชั่นล่าสุดที่ได้ทำการเก็บไว้เพื่อให้ออกมาเป็นซอร์สโค้ดในเวอร์ชั่นที่ผู้ใช้งานต้องการ ในการผสานซอร์สโค้ดนั้น สามารถทำได้โดยการใช้ ASTRewrite คลาสในการช่วยในการผสานข้อมูล หลังจากผสานข้อมูลเสร็จผลลัพธ์ที่ได้จะอยู่ในรูปแบบของวัตถุของคลาส ASTRewrite จำเป็นจะต้องมีการใช้ Document คลาสเพื่อทำการแปลงข้อมูลให้อยู่ในรูปของ String จากนั้นจึงทำการบันทึกเพื่อให้ได้ซอร์สโค้ด

3.3 รายละเอียดขั้นตอนการทำงานของเครื่องมือ

ในกระบวนการการทำงานของเครื่องมือนั้น จะแบ่งการทำงานออกเป็น 2 ส่วนหลักๆ ซึ่งก็คือ ส่วนนำเข้าซอร์สโค้ดและเปรียบเทียบ และส่วนของการกู้คืนโดยใน 2 ส่วนนี้จะแยกการทำงานออกจากกันโดยส่วนนำเข้าซอร์สโค้ดและเปรียบเทียบมีลำดับการทำงานดังรูปที่ 3.5 และส่วนของการกู้คืนจะมีลำดับการทำงานดังรูปที่ 3.6



รูปที่ 3.5 ขั้นตอนการทำงานในส่วนของการนำเข้าและเปรียบเทียบซอร์สโค้ด



รูปที่ 3.6 ขั้นตอนการทำงานของการกู้คืนซอร์สโค้ด

3.3.1 ส่วนติดต่อผู้ใช้งาน(User Interface)

ส่วนติดต่อผู้ใช้งานหรือ User Interface ของเครื่องมือจะถูกพัฒนาด้วยภาษาจาวาโดยการใช้ SWT/JFace ในการช่วยพัฒนา ในส่วนติดต่อผู้ใช้งานเป็นส่วนที่ใช้ในการเรียกใช้งานในส่วนต่างๆ ของเครื่องมือ โดยส่วนการติดต่อผู้ใช้งานจะถูกแบ่งหน้าที่หลักเป็นสองหน้าที่ด้วยแถบควบคุม โดยหน้าที่หลักของแถบควบคุมแถบที่หนึ่งของส่วนติดต่อผู้ใช้งานคือการนำเข้าซอร์สโค้ดสำหรับทำการเปรียบเทียบกันสองเวอร์ชั่น โดยซอร์สโค้ดที่จะนำเข้ามาทำการเปรียบเทียบนั้นจะต้องเป็นซอร์สโค้ดของภาษาจาวาที่เขียนอย่างถูกต้องตามหลักไวยากรณ์ของภาษาซึ่งจะต้องผ่านการตรวจสอบความถูกต้องด้วยตัวแปลภาษาของภาษาจาวาก่อนที่จะทำการนำเข้าและส่งต่อไปยังส่วนกระจายนิพจน์

สำหรับหน้าที่หลักของแถบควบคุมแถบที่สองของส่วนติดต่อผู้ใช้งานคือการทำการกู้คืนซอร์สโค้ดที่ได้ทำการเปรียบเทียบไว้ โดยในแถบควบคุมแถบที่สองจะทำการแสดงแฟ้มข้อมูลที่ผู้ใช้ได้ทำการสร้างไว้เพื่อเก็บความแตกต่างของซอร์สโค้ดที่ได้ทำการเปรียบเทียบในแถบควบคุมแถบที่หนึ่ง โดยในแฟ้มข้อมูลแต่ละแฟ้มจะแสดงชื่อคลาสของภาษาจาวาที่ผู้ใช้งานทำการเปรียบเทียบ จะมีการระบุวันที่เพื่อบ่งบอกถึงเวลาที่ผู้ใช้งานเครื่องมือทำการเปรียบเทียบซอร์สโค้ด เมื่อผู้ใช้งานทำเครื่องหมายเลือกที่หน้าชื่อของคลาสที่ได้มีการเปรียบเทียบไว้ ผู้ใช้งานจะสามารถทำการเรียกดูเพื่อเปรียบเทียบความแตกต่างของซอร์สโค้ดที่จะทำการกู้คืนก่อนที่ผู้ใช้งานจะกู้คืนซอร์สโค้ดเวอร์ชั่นนั้นๆ นอกจากนี้ผู้ใช้งานยังสามารถทำการลบแฟ้มข้อมูลของซอร์สโค้ดที่ผู้ใช้งานไม่ต้องการได้โดยการทำเครื่องหมายหน้าแฟ้มข้อมูลที่ต้องการลบ จากนั้นจึงทำการกดลบ

3.3.2 ส่วนการกระจายนิพจน์(Parser, filter and serialization)

เมื่อผู้ใช้ทำการนำเข้าซอร์สโค้ดมาจากส่วนติดต่อผู้ใช้งาน เครื่องมือจะทำการกระจายนิพจน์โดยการใช้ ASTParser ซึ่งเป็นคลาสของภาษาจาวาที่ช่วยในการกระจายนิพจน์และทำการกรองซอร์สโค้ดที่ได้ทำการกระจายแล้วโดยการใช้ ASTVisitor ซึ่งเป็น visitor pattern ในการแยกและกรองซอร์สโค้ด visitor pattern นั้นจะทำการวนเข้ายังโหนดต่างๆในซอร์สโค้ดจากนั้นจะทำการกรองเอาข้อมูลโดยการกรองซอร์สโค้ดที่ได้ผ่านการกระจายแล้วนั้น จะเอาเฉพาะส่วนที่จำเป็นในการเปรียบเทียบโดยการตัดส่วนที่ไม่จำเป็นออก

```
File file = new File(path);
BufferedReader in = new BufferedReader(new FileReader(file));
StringBuffer buffer = new StringBuffer();
String line = null;
while (null != (line = in.readLine())) {
    buffer.append("\t" + line);
    buffer.append("\n");
    if (monitor.isCanceled()) return;
}

ASTParser parser = ASTParser.newParser(AST.JLS3);
if (monitor.isCanceled()) return;
parser.setKind(ASTParser.K_COMPILATION_UNIT);
if (monitor.isCanceled()) return;
final String text = buffer.toString();
parser.setSource(text.toCharArray());
if (monitor.isCanceled()) return;
this.node = (CompilationUnit) parser.createAST(monitor);
```

รูปที่ 3.7 การอ่านข้อมูลจากซอร์สโค้ดที่นำเข้าและทำการกระจายนิพจน์

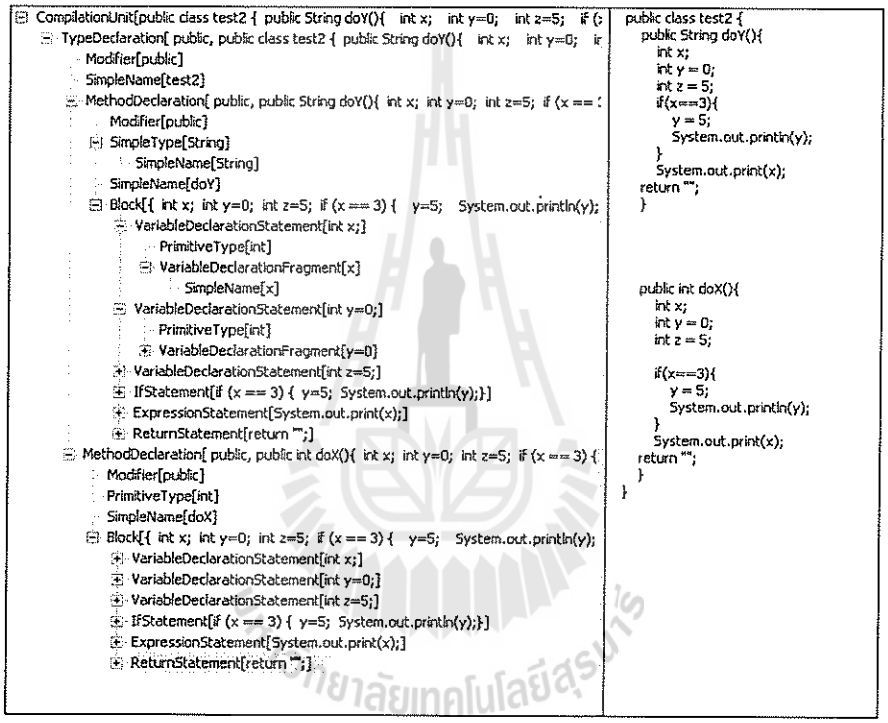
ในการกระจายนิพจน์ เครื่องมือจะทำการอ่านข้อมูลจากซอร์สโค้ดที่นำเข้าโดยข้อมูลที่นำเข้ามานั้นจะอยู่ในรูปแบบของ String จากนั้นเครื่องมือจะทำการแปลงค่าจาก string ให้กลายเป็น char[] เพื่อให้สามารถทำการกระจายนิพจน์โดยใช้ ASTParser คลาสได้

จากรูปที่ 3.7 ในการนำเข้าไฟล์จะต้องทำการสร้าง instance ของคลาส File ซึ่งเป็นคลาสที่อยู่ในแพ็คเกจ java.io จากนั้นทำการส่งพารามิเตอร์ที่ชื่อว่า path ซึ่งพารามิเตอร์นี้เป็นที่ระบุที่อยู่ของซอร์สโค้ดที่ต้องการนำเข้า จากนั้นทำการสร้าง instance ของคลาส BufferReader เพื่อใช้สำหรับการอ่านข้อมูลในซอร์สโค้ด และสร้าง instance เพื่อใช้ในการบันทึกค่าที่อ่านได้จากซอร์สโค้ดให้อยู่ในรูปของ String จากนั้นทำการสร้าง instance ของ ASTParser และทำการระบุประเภทของข้อมูลที่จะทำการกระจายนิพจน์ด้วยคำสั่ง setKind() จากนั้นจึงทำการส่งข้อมูลที่อ่านได้จากซอร์สโค้ดให้กับ instance ของ ASTParser คลาสเพื่อทำการสร้าง Abstract Syntax Tree(AST) โดยทำการแปลงข้อมูลที่จะส่งให้อยู่ในรูปแบบของ char[] จากนั้นจึงใช้คำสั่ง setSource() ทำการส่งข้อมูลไปทำการกระจายนิพจน์และใช้คำสั่ง createAST() ในการสร้าง Abstract Syntax Tree หลังจากทำการกระจายนิพจน์และสร้าง AST เสร็จเรียบร้อยแล้วเครื่องมือจะทำการเรียก accept() ดังรูปที่ 3.8 เพื่อทำการวนเข้าไปยังแต่ละโหนดใน AST โดยในการวนจะใช้ ASTExplorerVisitor คลาส สำหรับวนเข้าไปในแต่ละโหนดของ AST

```
ASTVisitor visitor = new ASTExplorerVisitor(treeControl,monitor);
node.accept(visitor);
```

รูปที่ 3.8 การเรียกใช้ accept() เมธอดเพื่อทำการวนเข้าไปยังแต่ละโหนดภายใน AST

สำหรับ ASTExplorerVisitor คลาสนั้น เป็นคลาสที่ extends มาจาก ASTVisitor คลาส โดยได้มีการส่งตัวแปรประเภท org.eclipse.swt.widgets.Tree เพื่อใช้สำหรับในการจัดเรียงโหนดของ AST ที่ได้ทำการ visit เข้าไปให้ออกมาอยู่ภายในรูปแบบของโครงสร้างต้นไม้สำหรับนำไปแสดงผลที่ส่วนติดต่อผู้ใช้งาน นอกจากนี้ภายใน ASTExplorerVisitor คลาสนั่น ได้มีการ implement visit เมธอดของ ASTVisitor คลาส เพื่อทำการดึงข้อมูลที่เกี่ยวข้องมาเตรียมไว้สำหรับทำการหาความแตกต่างในส่วนของการเปรียบเทียบและคัดกรองเอาส่วนที่ไม่จำเป็นออก ซึ่งผลลัพท์ที่ได้จากการกระจายนิพจน์และจัดเรียงโหนดจะได้ดังรูปที่ 3.9



รูปที่ 3.9 ตัวอย่างผลลัพท์ที่ได้จากการกระจายนิพจน์และการจัดเรียงใน org.eclipse.swt.widgets.Tree

การจัดเรียงโครงสร้างต้นไม้นั้น เป็นการนำเอาข้อมูลของซอร์สโค้ดที่ได้ผ่านการกระจายนิพจน์มาทำการสร้างโครงสร้างต้นไม้ วัตถุประสงค์ของการสร้างโครงสร้างต้นไม้ก็เพื่อการนำไปแสดงผลในส่วนติดต่อผู้ใช้งาน และการนำไปใช้เป็นข้อมูลนำเข้าเพื่อทำการเปรียบเทียบหาความแตกต่างของซอร์สโค้ด โดยโครงสร้างต้นไม้ที่ผ่านการจัดเรียงแล้วจะมีรูปแบบเหมือนในตัวอย่างที่ 3.9

การจัดเรียงข้อมูลและการสร้างโครงสร้างต้นไม้นั้น อาศัยการใช้ visitor pattern ช่วยในการสร้างและการจัดเรียง โดยในการจัดเรียงจะอยู่ในเมธอด preVisit() ที่อยู่ภายในคลาส ASTExplorerVisitor

```
ASTExplorerVisitor(Tree treeControl,IProgressMonitor monitor) {
    super(true);
    if (null == treeControl)
        throw new IllegalArgumentException();

    this.stack = new Stack();
    this.monitor = monitor;
    this.stack.push(treeControl);
}

public void preVisit(ASTNode node) {
    Object parent = this.stack.peek();
    TreeItem child = null;
    if (parent instanceof Tree)
        child = new TreeItem((Tree)parent,SWT.NONE);
    else
        child = new TreeItem((TreeItem)parent,SWT.NONE);

    child.setText(getNodeAsString(node));
    this.stack.push(child);
    child.setData(NODE,node);

}
```

รูปที่ 3.10 ASTExplorer constructor และเมธอด preVisitor()

จากรูปที่ 3.10 เป็น constructor ของคลาส ASTExplorerVisitor และเมธอด preVisit() ซึ่งเป็นส่วนที่ใช้ในการจัดเรียงข้อมูลลงใน Tree โดย ใน constructor เป็นการ assign ค่าให้กับตัวแปรต่างๆ โดยมีการทำการ push ตัวแปรประเภท Tree ลงใน stack สำหรับการนำไปจัดเรียง ภายในเมธอด preVisit() มีการทำการ peek ข้อมูลภายใน stack จากนั้นทำการตรวจสอบข้อมูลที่ทำการ peek โดยใช้เงื่อนไข if(parent instanceof Tree) ถ้าเป็นไปตามเงื่อนไข ตัวแปร TreeItem จะทำการสร้างโหนดโดยมี parent เป็น Tree แต่ถ้าไม่ใช่จะทำการสร้างโหนดโดยมี parent เป็น TreeItem จากนั้นทำการใส่ชื่อให้กับโหนดโดยใช้คำสั่ง child.setText(getNodeAsString(node)) โดยภายในคำสั่ง setText() มีการเรียกใช้เมธอด getNodeAsString() ค่าที่จะได้จากการเรียกใช้เมธอด getNodeAsString() จะมีค่าเป็น String ที่บอกชื่อประเภทของโหนดและค่าที่อยู่ภายในโหนด จากนั้นทำการ push ตัวแปร child กลับลงไปยัง stack และทำการใส่ข้อมูลลงในโหนดด้วยคำสั่ง child.setData() ดังตัวอย่างในรูปที่ 3.10

3.3.3 ส่วนการเปรียบเทียบหาความแตกต่างของซอร์สโค้ด(Differencing algorithm)

เพื่อเป็นการแยกแยะเวอร์ชั่นของซอร์สโค้ดและลดความซ้ำซ้อนของโค้ดที่จะทำการบันทึกลงในส่วนการสำรองและจัดเก็บข้อมูล จึงต้องทำการเปรียบเทียบเพื่อหาความแตกต่างของโค้ดที่มีการนำเข้ามา ในส่วนนี้จะใช้ข้อมูลที่ได้จากส่วนของการกระจายนิพจน์มาทำการเปรียบเทียบและหาความแตกต่าง

การทำงานในการเปรียบเทียบเริ่มต้นจากเมื่อเครื่องมือทำการกระจายนิพจน์และทำการจัดรูปแบบของซอร์สโค้ดให้อยู่ในรูปแบบของโครงสร้างต้นไม้เสร็จเรียบร้อยแล้ว เครื่องมือจะทำการเรียกใช้เมธอด treeDiffTravel() ซึ่งเป็นเมธอดสำหรับทำการเปรียบเทียบ

```
diff.addSelectionListener(new SelectionAdapter() {
    public void widgetSelected(SelectionEvent arg0){
        System.out.println("click");
        Tree treeL = ast1.treeControl;
        Tree treeR = ast2.treeControl;

        TreeItem leftItem = treeL.getItem(0);
        TreeItem rightItem = treeR.getItem(0);
        StyleRange styleRange = new StyleRange();
        StackDiff diff = new StackDiff();
        diff.setFileInfo(ASTExplorer.info);
        astexplorer.ASTExplorer.treeDiffTravel(leftItem, rightItem, ast1, ast2, styleRange, diff,new DiffFlag(false,false));
    }
});
```

รูปที่ 3.11 การเรียกใช้งานเมธอด treeDiffTravel()

โดยในการเรียกใช้งานเมธอด treeDiffTravel() เครื่องมือจะทำการเตรียมพารามิเตอร์ที่จำเป็นสำหรับการทำการเปรียบเทียบ โดยพารามิเตอร์ต่างๆที่ทำการส่งเข้าไปมีดังนี้

- TreeItem leftItem, TreeItem rightItem เป็น TreeItem ที่ได้จากการที่นำเข้าซอร์สโค้ดและทำการกระจายนิพจน์และจัดเรียงให้อยู่ในรูปแบบของโครงสร้างต้นไม้
- ast1, ast2 คือ วัตถุของคลาส ASTExplorer
- styleRange คือ วัตถุของคลาส StyleRange
- diff คือ วัตถุของคลาส StackDiff
- new DiffFlag() คือ วัตถุของคลาส DiffFlag

ภายในเมธอดจะทำการดึงข้อมูลของพารามิเตอร์ที่รับเข้าให้อยู่ในรูปแบบที่สามารถนำไปทำการเปรียบเทียบ

```
ASTNode leftNode = (ASTNode) left.getData(ASTExplorerVisitor.NODE);
ASTNode rightNode = (ASTNode) right.getData(ASTExplorerVisitor.NODE);
Tree treeL = astExplorerL.treeControl;
Tree treeR = astExplorerR.treeControl;
```

รูปที่ 3.12 การดึงข้อมูลพารามิเตอร์ที่รับเข้ามาในเมธอด treeDiffTravel()

โดยในการดึงข้อมูลพารามิเตอร์นั้นจะทำการนำเอาพารามิเตอร์ left และ right ซึ่งเป็นตัวแปรประเภท TreeItem มาทำการดึงข้อมูลโดยทำให้อยู่ในรูปแบบของ ASTNode เพื่อนำไปทำการเปรียบเทียบ และทำการดึงข้อมูลของพารามิเตอร์ astExplorerL.treeControl, astExplorerR.treeControl ไปไว้ในตัวแปร treeL และ treeR ซึ่งเป็นตัวแปรประเภท Tree ตามลำดับ จากนั้นจึงทำการเปรียบเทียบ โดยจะแบ่งการเปรียบเทียบเป็นกรณีที่ประเภทของโหนดไม่มีความแตกต่างและกรณีที่ประเภทของโหนดมีความแตกต่าง สำหรับกรณีที่ประเภทของโหนดไม่มีความแตกต่างจะทำการเช็คค่าภายในโหนดเพื่อหาความแตกต่างอีกที เมื่อทำการเปรียบเทียบถ้าเกิดมีความแตถต่างเกิดขึ้นจะทำมีการเรียกเมธอด createRange() เพื่อทำการเปลี่ยนสีของโหนดในการแสดงผลออกทาง Interface จากนั้นจึงนำข้อมูลของโหนดที่มีความแตกต่างไปทำการจัดรูปแบบ

```
styleRange = createRange(leftNode.getStartPosition(),leftNode.getLength(),astExplorerL.blue);
```

รูปที่ 3.13 การเรียกใช้เมธอด createRange()

รูปที่ 3.13 เป็นตัวอย่างการเรียกใช้งานเมธอด createRange() เพื่อเปลี่ยนแปลงสีของโหนดของโครงสร้างต้นไม้ เพื่อแสดงความแตกต่างสำหรับการนำไปแสดงผล โดยเมธอด createRange() มีรูปแบบดังนี้

createRange(int startPosition, int length, Color color)

- int startPosition คือ ตำแหน่งเริ่มต้นของโหนดที่มีความแตกต่าง
- int length คือ ความยาวของโหนด
- Color color คือ สีที่จะแสดงผล

จากนั้นเครื่องมือจะส่งข้อมูลของโหนดที่มีความแตกต่างไปยังคลาส StackDiff โดยการใช้วัตถุทำการเรียกเมธอดที่อยู่ภายในคลาส StackDiff และทำการส่งข้อมูลของโหนดไปทำการจัดรูปแบบและทำการบันทึก หลังจากที่ทำการบันทึกข้อมูลที่มีความแตกต่างเครื่องมือจะทำการเลื่อนตัวชี้ตำแหน่งโหนดไปยังโหนดถัดไปและทำการเรียกเมธอด diffTreeTravel() ซ้ำเพื่อทำการเปรียบเทียบหาความแตกต่างในระดับถัดไปจนกว่าโหนดของโครงสร้างต้นไม้ในฝั่งใดฝั่งหนึ่งจะหมด

3.3.4 ส่วนการสำรองและการจัดเก็บข้อมูล(Back up and Data Store)

เมื่อเครื่องมือทำการเปรียบเทียบหาความแตกต่างของซอร์สโค้ดแล้ว จะส่งผลลัพธ์ที่ได้มายังส่วนการสำรองและการจัดเก็บข้อมูล โดยในส่วนนี้ของเครื่องมือทำหน้าที่บันทึกผลลัพธ์ที่จะได้จากการเปรียบเทียบ โดยผลลัพธ์ที่ส่งมาจากส่วนการเปรียบเทียบโค้ดจะเป็นโหนดของคลาส ASTNode จึงต้องทำการดึงข้อมูลที่จำเป็นสำหรับในการเก็บ จากนั้นจึงทำการจัดเรียงข้อมูลให้อยู่ในรูปแบบที่ต้องการ

```
writeSkip(ASTNode nodeL, CompilationUnit cuL, ASTNode nodeR, CompilationUnit cuR, String status)
```

รูปที่ 3.14 รูปแบบของเมธอดที่ใช้ในการจัดรูปแบบข้อมูล

ในการจัดเรียงข้อมูลจะมีการเรียกใช้เมธอด ดังรูปที่ 3.14 โดยจะมีการส่งพารามิเตอร์ต่างๆเข้าไปดังนี้

- ASTNode nodeL, ASTNode nodeR เป็นตัวแปรประเภท ASTNode ของโหนดที่มีความแตกต่างของทั้งฝั่งซ้ายและฝั่งขวา

- CompilationUnit cuL, CompilationUnit cuR เป็นตัวแปรประเภท CompilationUnit ของซอร์สโค้ดทั้งฝั่งซ้ายและฝั่งขวา

- String status เป็นตัวแปรประเภท String ที่ใช้สำหรับบอกสถานะความแตกต่างของซอร์สโค้ด โดยสถานะความแตกต่างของซอร์สโค้ดจะแบ่งได้เป็น Method Modify, Node Modify, Add, Delete

สำหรับการหาบรรทัดของซอร์สโค้ดเพื่อให้ข้อมูลที่ได้เป็นไปตามรูปแบบที่ได้ทำการออกแบบไว้ สามารถทำได้โดยการให้วัตถุของคลาส CompilatinUnit เรียกใช้เมธอด getLineNumber()

หลังจากที่ทำการจัดรูปแบบของข้อมูลแล้วจะทำการส่งข้อมูลที่ได้ไปทำการบันทึกลงใน text file โดยข้อมูลที่บันทึกลงใน text file เรียบร้อยแล้วจะมีลักษณะดังรูปที่ 3.15 นอกจากเครื่องมือจะทำการเก็บบันทึกข้อมูลที่ได้จากการเปรียบเทียบแล้ว ตัวเครื่องมือจะทำการเก็บซอร์สโค้ดเวอร์ชั่นล่าสุดไว้ด้วยเพื่อใช้ในการอ้างอิงสำหรับการกู้คืนเมื่อผู้ใช้งานต้องการ

```
[Line][2][Method Modify]public static void doY(){  if (x != 5) {    System.out.println("");  }  x=10;  test1.dox2();}[End Method Modify]
[Line][9][Delete]public static void dox(){  test1.dox2();}[End Delete]
```

รูปที่ 3.15 แสดงตัวอย่างรูปแบบของข้อมูลที่ทำการเก็บลงใน text file

3.3.5 ส่วนการกู้คืน(Recovery)

ในส่วนนี้จะทำการสร้างซอร์สโค้ดในเวอร์ชั่นที่ผู้ใช้งานต้องการกู้คืนโดยอาศัยข้อมูลที่ได้จากส่วนการสำรองและการเก็บข้อมูลนำมาทำการสร้างโดยการอ้างอิงจากข้อมูลที่ได้เก็บไว้ใน text file ซึ่งเป็นข้อมูลที่มีการบ่งบอกว่า เวอร์ชั่นนั้นๆ มีการแก้ไข การเพิ่ม หรือการลบ ข้อมูลของซอร์สโค้ดในส่วนใดบ้าง เมื่อจะทำการกู้คืน เครื่องมือจะทำการเช็คและทำการรวมข้อมูลเพื่อไปแปลงเป็นซอร์สโค้ดต้นฉบับตามเวอร์ชั่นที่ผู้ใช้งานต้องการ

```
byte[] bytes = getFile(javaFile);
String source = new String(bytes,"UTF-8");
IDocument doc = new Document(source);
CompilationUnit astRoot = parseCompilationUnit(bytes);
```

รูปที่ 3.16 การอ่านข้อมูลจากจาวาไฟล์และการกระจายนิพจน์

ในการกู้คืน การทำงานจะเริ่มจากการนำเข้าซอร์สโค้ดในเวอร์ชั่นล่าสุดที่ได้มีการทำการเก็บไว้ จากนั้นจะส่งไปยังเมธอดเพื่อทำการอ่านข้อมูลภายในซอร์สโค้ดให้ออกมาอยู่ในรูปแบบของ byte[] จากนั้นทำการสร้างวัตถุของคลาส Document เพื่อเตรียมไว้สำหรับการแปลงข้อมูลที่ได้จากการผสานซอร์สโค้ดในขั้นตอนสุดท้าย เมื่อซอร์สโค้ดที่นำเข้ามาถูกแปลงให้อยู่ในรูปของ byte[] แล้ว เครื่องมือจะส่งข้อมูลนี้ไปทำการกระจายนิพจน์เพื่อให้ได้ข้อมูลที่อยู่ในรูปของวัตถุของคลาส CompilationUnit

```
ArrayList diffInfoArr = ReadFileDiffInfo.readFileDiffInfo(textFile);
ASTRewrite rewrite = ASTRewrite.create(astRoot.getAST());
```

รูปที่ 3.17 อ่านข้อมูลจาก text file และสร้างวัตถุของคลาส ASTRewrite

จากนั้น เครื่องมือจะทำการอ่านข้อมูลของ text file และทำการแยกส่วนของข้อมูลที่อยู่ภายใน text file โดยข้อมูลที่ทำการแยกแล้วจะถูกเก็บลงในตัวแปรในคลาส diffInfo ทำการเก็บวัตถุของคลาส diffInfo ลงใน ArrayList แล้วส่งกลับมายังตัวแปร diffInfoArr ดังในรูปที่ 3.17 ขั้นตอนถัดไปคือการสร้างวัตถุของคลาส ASTRewrite ในการสร้างวัตถุจะสามารถทำได้โดยการใช้คำสั่ง ASTRewrite.create() โดยมีการส่ง AST ที่ได้จากการกระจายนิพจน์ของซอร์สโค้ดที่ได้ผ่านการกระจายแล้วเป็นพารามิเตอร์เพื่อใช้ในการสร้างวัตถุ

```
MethodVisitor mvs = new MethodVisitor(diffInfo.getLine(),astRoot,"Method");
astRoot.accept(mvs);
ASTNode targetNode = mvs.targetMethodBlock;
```

รูปที่ 3.18 การหาโหนดเป้าหมายในการแก้ไขข้อมูล

หลังจากได้ทำการสร้างวัตถุของคลาส ASTRewrite แล้วเครื่องมือจะแยกการทำงานออกตามเงื่อนไข โดยภายในแต่ละเงื่อนไขจะมีการทำงานที่คล้ายกันคือการหาโหนดเป้าหมายที่ต้องการแก้ไขโดยในการกำหนดโหนดเป้าหมายนั้นจะใช้ visitor pattern มาช่วยในการกำหนดเป้าหมาย โดยทำการสร้างวัตถุของคลาส MethodVisitor ซึ่งเป็นคลาสที่สืบทอดมาจากคลาส ASTVisitor

```
MethodVisitor(int line, CompilationUnit cu)
```

รูปที่ 3.19 รูปแบบของ constructor ของ MethodVisitor คลาส

โดยในการสร้างวัตถุของคลาสจะมีการส่งพารามิเตอร์ต่างๆ เพื่อใช้ในการระบุโหนดเป้าหมายที่ต้องการทำการแก้ไขข้อมูลโดยพารามิเตอร์ที่ส่งเข้าไปมีดังนี้

- int line คือตัวเลขบอกบรรทัดของซอร์สโค้ดที่ได้มาจากข้อมูลใน text file

CompilationUnit cu คือตัวแปรประเภท CompilationUnit ที่ได้จากการสร้างในส่วนต้นของขั้นตอนการกู้คืน

จากนั้นเมื่อทำการเรียกใช้คำสั่ง accept() จะทำให้ได้โหนดเป้าหมายในการแก้ไขข้อมูลออกมาโดยอยู่ในรูปตัวแปรของคลาส ASTNode และเพื่อให้ข้อมูลที่อ่านได้จาก text file อยู่ในรูปของ ASTNode จึงจะต้องทำการกระจายนิพจน์และสร้าง visitor pattern สำหรับข้อมูลที่ได้จาก text file ในส่วนของการกระจายนิพจน์จะมีความแตกต่างกับการกระจายนิพจน์ของซอร์สโค้ดตรงที่การเลือกประเภทของข้อมูลที่จะทำการกระจายนิพจน์ สำหรับข้อมูลใน text file จะต้องทำการตั้งค่าประเภทการกระจายนิพจน์เป็นประเภท K_CLASS_BODY_DECLARATIONS

เมื่อได้ ASTNode เป้าหมายและ ASTNode ของข้อมูลใน text file แล้ว เครื่องมือจะทำการเช็คเงื่อนไขจากนั้นจะทำการแก้ไขข้อมูลของ ASTNode โดยการแก้ไขจะใช้วัตถุของคลาส ASTRewrite ช่วยในการแก้ไข เมื่อทำการแก้ไขข้อมูลเสร็จสิ้นเครื่องมือจะทำการแปลงวัตถุของคลาส ASTRewrite ให้อยู่ในรูปแบบของซอร์สโค้ดเพื่อส่งให้ผู้ใช้งาน

```
TextEdit edit = rewrite.rewriteAST(doc,null);
edit.apply(doc);
String newCode = doc.get();
```

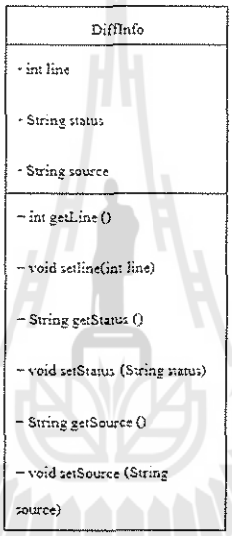
รูปที่ 3.20 การแปลงวัตถุของคลาส ASTRewrite ให้อยู่ในรูปแบบของซอร์สโค้ด

เมื่อได้วัตถุของ ASTRewrite ที่ได้จากการแก้ไข ASTNode แล้ว จะต้องทำการแปลงวัตถุให้กลับไปอยู่ในรูปของซอร์สโค้ดเพื่อให้ผู้ใช้สามารถนำไปใช้งานได้ทันที การแปลงวัตถุของ ASTRewrite จะใช้คำสั่ง rewriteAST() เพื่อทำการบันทึกการแก้ไขและส่งการบันทึกไปยังตัวแปรของคลาส TextEdit จากนั้นตัวแปรของคลาส TextEdit จะทำการเรียกคำสั่ง apply() เพื่อทำการบันทึกการเปลี่ยนแปลงไปยังวัตถุของคลาส Document ที่ได้ทำการสร้างไว้ตอนต้นของการทำงานและทำการเรียกคำสั่ง get() เพื่อดึงข้อมูลที่อยู่ในวัตถุของคลาส Document ออกมาให้อยู่ในรูปแบบของ String จากนั้นจึงทำการบันทึกซึ่งผลลัพธ์ที่ได้จะเป็นซอร์สโค้ดในเวอร์ชั่นก่อนหน้าเวอร์ชั่นล่าสุด เพื่อให้ได้ซอร์สโค้ดในเวอร์ชั่นที่ผู้ใช้งานต้องการกระบวนการการทำงานทั้งหมดจะถูกทำการวนซ้ำไปจนกว่าจะได้ซอร์สโค้ดในเวอร์ชั่นที่ผู้ใช้งานต้องการซึ่งเป็นผลลัพธ์สุดท้ายที่ได้จากเครื่องมือ

3.3.5.1 DiffInfo คลาส

เป็นคลาสที่ใช้สำหรับในการเก็บข้อมูลที่ได้จากการอ่านข้อมูลใน text file โดยเป็น getter – setter คลาส โดยภายในคลาสจะประกอบด้วยตัวแปร

- int line คือตัวแปรที่ใช้สำหรับเก็บค่าของเลขบรรทัด
- String status คือตัวแปรที่ใช้สำหรับทำการเก็บ status ที่อ่านค่าได้จาก text file
- String source คือตัวแปรที่ใช้สำหรับทำการเก็บซอร์สโค้ดที่อ่านได้จาก text file



รูปที่ 3.21 โครงสร้างของคลาส DiffInfo

3.4 การออกแบบชุดทดสอบ

ในการทดสอบจะแบ่งการทดสอบออกเป็นสองประเภทคือ ทดสอบหาความแตกต่างของซอร์สโค้ดและทดสอบความถูกต้องของการกู้คืน โดยในการทดสอบหาความแตกต่างของซอร์สโค้ดจะใช้ซอร์สโค้ดที่มีลักษณะต่างๆ เพื่อหาความแตกต่างในหลายๆรูปแบบ ซึ่งในการทดสอบจะสมมุติว่าซอร์สโค้ดที่นำมาทดสอบเป็นซอร์สโค้ดที่มีคลาสเป็นคลาสเดียวกัน

ชุดของซอร์สโค้ดที่จะนำมาทำการทดสอบหาความแตกต่างนั้น จะสามารถแยกได้เป็น

การเปรียบเทียบหาความแตกต่างในระดับ Method โดยในการเปรียบเทียบหาความแตกต่างในระดับ Method นี้ จะทำการเปรียบเทียบซอร์สโค้ดที่ Method การเพิ่ม การทดสอบเปรียบเทียบซอร์สโค้ดที่ Method มีการถูกลบออกและการทดสอบเปรียบเทียบซอร์สโค้ดที่มีการจัดเรียง Method ที่แตกต่างกัน โดยในการทดสอบในระดับ Method นี้ จะไม่สนใจความแตกต่างภายใน Method และคลาสที่ใช้ทดสอบจะเป็นคลาสที่มีชื่อเดียวกัน

```
public class test {
        public int doX(){

        }
}
```

```
public class test {
        public int doX(){

        }
        public int doY(){

        }
}
```

รูปที่ 3.22 ซอร์สโค้ดที่ใช้ทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการเพิ่ม

จากรูปที่ 3.22 เป็นซอร์สโค้ดที่ใช้สำหรับทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการเพิ่ม โดยจะมีการทำการเพิ่ม Method ที่ชื่อว่า doY() เข้าไปยังซอร์สโค้ดฝั่งขวาเพื่อใช้ในการทดสอบ

```
public class test {
        public int doX(){

        }
        public int doY(){

        }
}
```

```
public class test {
        public int doX(){

        }
}
```

รูปที่ 3.23 ซอร์สโค้ดที่ใช้ทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการลบ

จากรูปที่ 3.23 เป็นซอร์สโค้ดที่ใช้สำหรับทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการลบ โดยจะมีการทำการลบ Method ที่ชื่อว่า doY() ออกจากซอร์สโค้ดฝั่งขวาเพื่อใช้ในการทดสอบ

```
public class test {
        public int doY(){

        }
        public int doX(){

        }
}
```

```
public class test {
        public int doX(){

        }
        public int doY(){

        }
}
```

รูปที่ 3.24 ซอร์สโค้ดที่ใช้ทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการเรียงลำดับที่แตกต่างกัน

จากรูปที่ 3.24 เป็นซอร์สโค้ดที่ใช้สำหรับทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการเรียงลำดับที่แตกต่างกัน โดยจะทำการสลับลำดับของ Method ที่อยู่ภายในซอร์สโค้ดทั้งสองฝั่งให้มีความแตกต่างเพื่อใช้ในการทดสอบ

- การเปรียบเทียบหาความแตกต่างในระดับ node ซึ่งในการเปรียบเทียบหาความแตกต่างในระดับ node นี้ จะเป็นการเปรียบเทียบความแตกต่างของ node ที่อยู่ภายใน Method หรือซอร์สโค้ดที่อยู่ภายใน Method โดยซอร์สโค้ดที่นำมาใช้เปรียบเทียบจะแบ่งได้เป็น ซอร์สโค้ดที่มีการเพิ่ม statement และซอร์สโค้ดที่มีการลบออกของ statement ซอร์สโค้ดที่มีความแต่งต่างในเรื่องของตำแหน่งของ statement และความแตกต่างทางด้านค่าของตัวแปรต่างๆภายในซอร์สโค้ดและสุดท้ายจะทำการทดสอบโดยการหาความแตกต่างในระดับ Method และโหนดไปพร้อมๆกัน โดยการทำการรวมซอร์สโค้ดจากกรณีต่างๆไว้ในซอร์สโค้ดเดียวกันเพื่อทำการทดสอบ

```
public class test {
        public int doY(){
                int x;
                int y = 0;
                x = 3;
                if(x==3){
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
        return 2;
        }
}
```

```
public class test {
        public int doY(){
                int x;
                int y = 0;
                int z = 5;
                x = 3;
                if(x==3){
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
        return 2;
        }
}
```

รูปที่ 3.25 ซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่มีการเพิ่มของโหนด

จากรูปที่ 3.25 แสดงซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่มีการเพิ่มของโหนด โดยโหนดที่เพิ่มเข้ามานั้นคือโหนดของคำสั่ง int z = 5; ซึ่งจะมีการเพิ่มเข้ามาในซอร์สโค้ดฝั่งขวา

```
public class test {
        public int doY(){
                int x;
                int y = 0;
                int z = 5;
                x = 3;
                if (x == 3) {
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
                return 2;
        }
}
```

```
public class test {
        public int doY(){
                int x;
                int y = 0;
                x = 3;
                if(x==3){
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
                return 2;
        }
}
```

รูปที่ 3.26 ซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่มีการลบของโหนด

จากรูปที่ 3.26 แสดงซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่มีการลบของโหนด โดยโหนดที่ถูกลบออกไปนั้นคือโหนดของคำสั่ง int z = 5; ซึ่งจะมีการลบออกจากซอร์สโค้ดฝั่งขวา

```
public class test {
        public int doY(){
                int x;
                x = 3;
                int z = 5;
                int y = 0;
                if(x==3){
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
                return 2;
        }
}
```

```
public class test {
        public int doY(){
                int x;
                int y = 0;
                int z = 5;
                x = 3;
                if(x==3){
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
                return 2;
        }
}
```

รูปที่ 3.27 ซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่โหนดมีการเรียงลำดับที่แตกต่าง

จากรูปที่ 3.27 แสดงซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่โหนดมีการเรียงลำดับที่แตกต่างกัน โดยโหนดที่มีการเรียงลำดับคือโหนดของคำสั่ง x = 3; และ

โหนดของคำสั่ง int y = 0; ซึ่งเป็นโหนดที่อยู่ในซอร์สโค้ดฝั่งซ้ายโดยโหนดทั้ง 2 สลับตำแหน่งกับตำแหน่งของโหนดที่อยู่ในซอร์สโค้ดฝั่งขวา

```
public class test {
        public int doY(){
                int x;
                int y = 0;
                int z = 5;
                x = 3;
                if(x==3){
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
                return 2;
        }
}
```

```
public class test {
        public int doY(){
                int x;
                int y = 5;
                int z = 5;
                x = 10;
                if(x==3){
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
                return 2;
        }
}
```

รูปที่ 3.28 ซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่มีการแก้ไขข้อมูลภายในโหนด

รูปที่ 3.28 แสดงซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่มีการแก้ไขข้อมูลภายในโหนด โดยในการแก้ไขข้อมูลภายในโหนดนี้จะหมายถึงการแก้ไขค่าต่างๆ เช่น ค่อของตัวแปรภายในซอร์สโค้ด โดยซอร์สโค้ดที่ใช้ทดสอบจะมีการแก้ไขข้อมูลภายในโหนดของคำสั่ง int y = 0; ในซอร์สโค้ดฝั่งซ้ายเป็น int y = 5; ในซอร์สโค้ดฝั่งขวา และแก้ไขข้อมูลภายในโหนดของคำสั่ง x = 3; ในซอร์สโค้ดฝั่งซ้ายเป็น x = 10; ในซอร์สโค้ดฝั่งขวา

```
public class test {
        public int doA() {
                int x;
                int y = 0;
                x = 3;
                if(x==3){
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
                return 2;
        }

        public static void doB() {
                System.out.println("Method doB");
        }

        public String doZ(){
                return "Method doZ";
        }
}
```

```
public class test {
        public int doX(){
                int x;
                int y = 0;
                int z = 5;
                x = 3;
                if(x==3){
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
                return 2;
        }

        public int doY(){
                return 0;
        }

        public String doZ(){
                return "Method doZ";
        }
}
```

รูปที่ 3.29 ซอร์สโค้ดที่ใช้ทดสอบเปรียบเทียบหาความแตกต่างโดยรวมกรณีต่างๆไว้ในซอร์สโค้ดชุดเดียวกัน

รูปที่ 3.29 แสดงซอร์สโค้ดที่ใช้ทดสอบเปรียบเทียบหาความแตกต่างโดยรวมเอากรณีต่างๆที่ได้ทดสอบมารวมไว้ในซอร์สโค้ดชุดเดียวกันเพื่อจำลองซอร์สโค้ดให้เหมือนกับการใช้งานจริง โดยซอร์สโค้ดที่ใช้ทดสอบจะมีความแตกต่างทั้งในระดับ Method และระดับโหนด ในความแตกต่างในระดับโหนดจะเป็นความแตกต่างทางด้านของชื่อ Method และ Signature โดยซอร์สโค้ดฝั่งซ้ายจะมี Method ชื่อ doA() doB() และ doZ() ส่วนซอร์สโค้ดฝั่งขวาจะมี Method ชื่อ doX() doY() และ doZ() ตามลำดับ ภายใน Method มีความแตกต่างในระดับโหนด โดย Method doX() จะมีการลบโหนดของคำสั่ง int z = 5; ออก Method doB() และ doY() มีความแตกต่างกันทั้งในระดับ Method และในระดับโหนด ส่วน Method doZ() ทั้งในซอร์สโค้ดฝั่งซ้ายและซอร์สโค้ดฝั่งขวาจะไม่มีความแตกต่างกัน

สำหรับการทดสอบการกู้คืนนั้น จะใช้การทดสอบโดยการทดลองทำการกู้คืนข้อมูลโดยอาศัยข้อมูลจากชุดทดสอบหาความแตกต่างที่ได้ทำการเปรียบเทียบ แล้วจึงทำการทดลองกู้คืน ในการทดสอบการกู้คืนจะแบ่งการทดสอบออกเป็น การกู้คืนข้อมูลของเวอร์ชั่นที่ติดกัน การกู้คืนของเวอร์ชั่นที่ไม่ติดกัน

บทที่ 4

ผลการดำเนินการวิจัยและการอภิปรายผล

การทดสอบการทำงานของเครื่องมือเพื่อช่วยในการกู้คือโค้ดสำหรับภาษาจาวานั้น ในการทดสอบจะทำการแบ่งการทดสอบออกเป็น 2 ส่วนหลักๆ โดยทำการทดสอบในส่วนของการเปรียบเทียบหาความแตกต่างของซอร์สโค้ดเพื่อหาความถูกต้องของเครื่องมือในการเปรียบเทียบและหาความแตกต่าง และทำการทดสอบในส่วนของการกู้คืนเพื่อตรวจสอบความถูกต้องของเครื่องมือในการกู้คืนซอร์สโค้ดจากข้อมูลที่ได้ทำการเก็บบันทึกไว้

4.1 การทดสอบการเปรียบเทียบหาความแตกต่างของซอร์สโค้ด

ในการทดสอบการเปรียบเทียบหาความแตกต่างของซอร์สโค้ดนั้น จะเป็นการทดสอบเพื่อหาความถูกต้องในการเปรียบเทียบหาความแตกต่างของซอร์สโค้ด สำหรับซอร์สโค้ดที่จะนำมาทำการทดสอบนั้น ได้มีการออกแบบซอร์สโค้ดที่ใช้สำหรับทดสอบโดยแบ่งการทดสอบออกเป็นสองส่วนคือ การทดสอบเปรียบเทียบหาความแตกต่างในระดับ Method และการเปรียบเทียบหาความแตกต่างในระดับโหนด

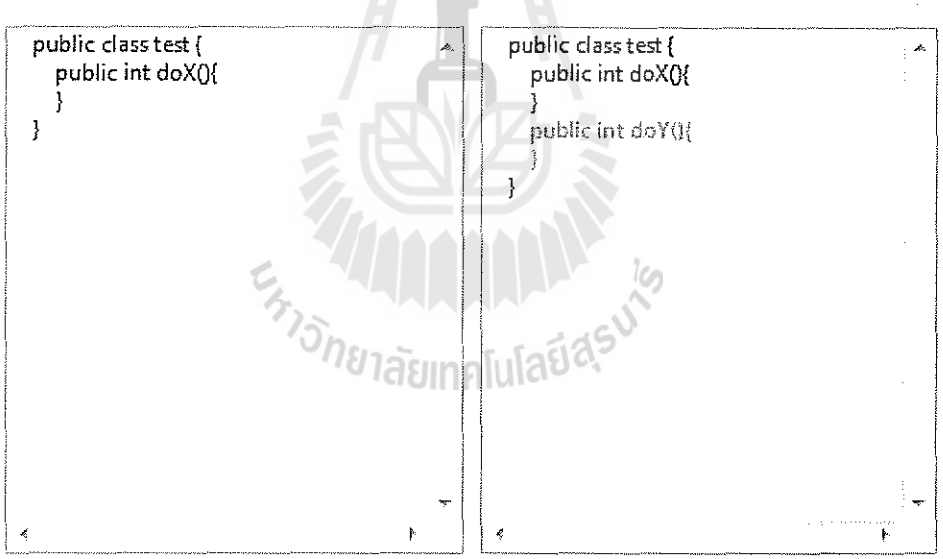
การทดสอบระดับ Method นั้น จะแบ่งการทดสอบออกเป็น 4 กรณีทดสอบคือ การทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการเพิ่มมาในคลาส การทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการถูกลบออกจากคลาส การทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการจัดเรียงที่แตกต่างกัน และการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มี Signature ที่แตกต่างกันและเหมือนกัน สำหรับในการทดสอบเปรียบเทียบหาความแตกต่างในระดับ Method นี้ ไม่สนใจซอร์สโค้ดที่อยู่ภายใน Method เพราะว่าเป็นการทดสอบในระดับ Method สำหรับ การเปรียบเทียบหาความแตกต่างของซอร์สโค้ดภายใน Method นั้น จะทำการทดสอบเพื่อเปรียบเทียบหาความแตกต่างในระดับโหนดต่อไป

```
public class test {
        public int doX(){
        }
}
```

```
public class test {
        public int doX(){
        }
        public int doY(){
        }
}
```

รูปที่ 4.1 ซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการเพิ่ม

จากรูปที่ 4.1 ซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการเพิ่ม จะมีความแตกต่างในด้านของจำนวน Method ที่อยู่ภายในคลาส ในการเปรียบเทียบ จะให้ซอร์สโค้ดทางฝั่งซ้ายเป็นซอร์สโค้ดตั้งต้นสำหรับใช้ทำการเปรียบเทียบ และซอร์สโค้ดฝั่งขวาจะเป็นซอร์สโค้ดที่นำมาเปรียบเทียบกับซอร์สโค้ดฝั่งซ้าย โดยจากรูปที่ 4.1 จะเห็นว่า ซอร์สโค้ดฝั่งขวามี Method ที่ชื่อว่า doY() เพิ่มเข้ามา



รูปที่ 4.2 ผลลัพธ์ที่ได้จากการเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการเพิ่ม

จากรูปที่ 4.2 เป็นการแสดงผลลัพธ์ที่จะได้จากการเปรียบเทียบหาความแตกต่างของซอร์สโค้ดในกรณีที่ Method มีการเพิ่มของเครื่องมือ โดยเครื่องมือสามารถบอกความแตกต่างของซอร์สในกรณีที่ Method มีการเพิ่มในรูปแบบของการทำไฮไลท์ในส่วนที่มีการเพิ่มเข้ามา

```
public class test {
        public int doX(){

        }
        public int doY(){

        }
}
```

```
public class test {
        public int doX(){

        }
}
```

รูปที่ 4.3 ซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการลบ

ในกรณีการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการลบนั้น จะใช้ซอร์สโค้ดดังรูปที่ 4.3 ในการทดสอบ จากซอร์สโค้ดจะเห็นได้ว่า Method ชื่อ doY() ในฝั่งซ้ายได้ถูกลบออกเมื่อมีการนำมาเทียบกับซอร์สโค้ดฝั่งขวา โดยผลลัพธ์ที่ได้จากการทดสอบเปรียบเทียบหาความแตกต่างกรณีที่ Method มีการลบนั้น จะแสดงผลลัพธ์ดังรูปที่ 4.4

```
public class test {
    public int doX(){
    }
    public int doY(){
    }
}
```

```
public class test {
    public int doX(){
    }
}
```

รูปที่ 4.4 ผลลัพธ์ที่ได้จากการเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการลบ

จากรูปที่ 4.4 จะเห็นได้ว่าเครื่องมือสามารถระบุถึงตำแหน่งที่มีการเปลี่ยนแปลงจากการเปรียบทียบหาความแตกต่างในกรณีที่ Method มีการลบได้ โดยเครื่องมือสามารถบ่งบอกตำแหน่งที่มีความแตกต่างโดยการทำไฮไลท์ในส่วนที่มีการลบออก

```
public class test {
        public int doY(){

        }
        public int doX(){

        }
}
```

```
public class test {
        public int doX(){

        }
        public int doY(){

        }
}
```

รูปที่ 4.5 ซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการเรียงลำดับที่แตกต่างกัน

สำหรับในการทดสอบเปรียบเทียบหาความแตกต่างของซอร์สโค้ดในกรณีที่ Method มีการเรียงลำดับที่แตกต่างกัน จะใช้ซอร์สโค้ดดังรูปที่ 4.5 ในการทดสอบ ซึ่งจากรูปจะเห็นว่าทั้งซอร์สโค้ดทั้งฝั่งซ้ายและขวา เป็นซอร์สโค้ดที่เหมือนกัน แตกต่างกันที่การเรียงลำดับของ Method ที่อยู่ภายในคลาส

```
public class test {
   public int doY(){
   }
   public int doX(){
   }
}
```

```
public class test {
   public int doX(){
   }
   public int doY(){
   }
}
```


รูปที่ 4.6 ผลลัพธ์ที่ได้จากการเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการเรียงลำดับที่แตกต่างกัน

ผลลัพธ์ที่ได้จากการเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการเรียงลำดับที่แตกต่างกัน จะแสดงดังรูปที่ 4.6 โดยในการเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการ

เรียงลำดับที่แตกต่างกันนั้น เครื่องมือจะทำการไฮไลท์ในส่วนที่มีความแตกต่าง จากรูปส่วนที่แตกต่างคือชื่อของ Method ทั้ง 2 Method

การทดสอบถัดจากการเปรียบเทียบหาความแตกต่างในระดับ Method ก็คือ การเปรียบเทียบหาความแตกต่างในระดับโหนด ซึ่งหมายถึงการเปรียบเทียบหาความแตกต่างที่เกิดขึ้นภายในซอร์สโค้ดที่อยู่ภายใน Method ในการทดสอบนี้จะแบ่งการทดสอบย่อยๆออกเป็น การเปรียบเทียบหาความแตกต่างในระดับโหนดในกรณีที่มีการเพิ่มเข้ามาของโหนด การเปรียบเทียบหาความแตกต่างในระดับโหนดในกรณีที่มีการลบออกของโหนด การเปรียบเทียบหาความแตกต่างในระดับโหนดในกรณีที่โหนดมีการเรียงลำดับที่แตกต่างและสุดท้ายการเปรียบเทียบหาความแตกต่างในระดับโหนดในกรณีที่โหนดมีการแก้ไขข้อมูลภายในโหนด

```
public class test {
        public int doY(){
                int x;
                int y = 0;
                x = 3;
                if(x==3){
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
        return 2;
        }
}
```

```
public class test {
        public int doY(){
                int x;
                int y = 0;
                int z = 5;
                x = 3;
                if(x==3){
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
        return 2;
        }
}
```

รูปที่ 4.7 ซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่มีการเพิ่มของโหนด

การทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่มีการเพิ่มของโหนดจะใช้ซอร์สโค้ดดังรูปที่ 4.7 ในการทดสอบเปรียบเทียบหาความแตกต่าง จากรูปจะเห็นว่าซอร์สโค้ดที่ใช้ทำการทดสอบนั้นมีการเพิ่มเข้ามาของโหนด โดยส่วนที่เพิ่มเข้ามาจะเป็นโหนดของคำสั่ง int z = 5; ซึ่งเป็นโหนดที่อยู่ในซอร์สโค้ดฝั่งขวามือ

```
public class test {
   public int doY(){
      int x;
      int y = 0;
      x = 3;
      if(x==3){
      y = 5;
      System.out.println(y);
      }
      System.out.print(x);
   return 2;
   }
}
```

```
public class test {
   public int doY(){
      int x;
      int y = 0;
      int z = 5;
      x = 3;
      if(x==3){
      y = 5;
      System.out.println(y);
      }
      System.out.print(x);
   return 2;
   }
}
```

รูปที่ 4.8 ผลลัพธ์ของการเปรียบเทียบหาความแตกต่างในกรณีที่มีการเพิ่มของโหนด

ผลลัพธ์ของการทดสอบเปรียบเทียบหาความแตกต่าในกรณีที่มีการเพิ่มของโหนดจะแสดงในรูปที่ 4.8 ซึ่งเครื่องมือจะทำการทำไฮไลท์ในส่วนที่มีความแตกต่างและส่วนที่มีการเพิ่มเข้ามาดังรูป

```
public class test {
        public int doY(){
                int x;
                int y = 0;
                int z = 5;
                x = 3;
                if (x == 3) {
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
                return 2;
        }
}
```

```
public class test {
        public int doY(){
                int x;
                int y = 0;
                x = 3;
                if(x==3){
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
                return 2;
        }
}
```

รูปที่ 4.9 ซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่มีการลบของโหนด

ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่มีการลบของโหนดจะใช้ซอร์สโค้ดในรูปที่ 4.9 ทำการทดสอบ ซึ่งจากรูปจะเห็นว่าโหนดของคำสั่ง int z = 5; ที่อยู่ในซอร์สโค้ดฝั่งซ้ายได้ถูกลบออกไปในซอร์สโค้ดฝั่งขวา ผลลัพธ์ที่ได้จากการทดสอบเปรียบเทียบหาความแตกต่างจะได้ผลลัพธ์ดังรูปที่ 4.10

```
public class test {
   public int doY(){
      int x;
      int y = 0;
      int z = 5;
      x = 3;
      if (x == 3) {
         y = 5;
         System.out.println(y);
      }
      System.out.print(x);
      return 2;
   }
}
```

```
public class test {
   public int doY(){
      int x;
      int y = 0;
      x = 3;
      if(x==3){
      y = 5;
      System.out.println(y);
      }
      System.out.print(x);
   return 2;
   }
}
```

รูปที่ 4.10 ผลลัพธ์ของการเปรียบเทียบหาความแตกต่างในกรณีที่มีการลบของโหนด

```
public class test {
        public int doY(){
                int x;
                x = 3;
                int z = 5;
                int y = 0;
                if(x==3){
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
                return 2;
        }
}
```

```
public class test {
        public int doY(){
                int x;
                int y = 0;
                int z = 5;
                x = 3;
                if(x==3){
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
                return 2;
        }
}
```

รูปที่ 4.11 ซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่โหนดมีการเรียงลำดับที่แตกต่าง

จากรูปที่ 4.11 แสดงซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่โหนดมีการเรียงลำดับที่แตกต่างกัน ซึ่งการเรียงลำดับของโหนดที่แตกต่างกันนั้น เป็นการเรียงลำดับโหนดของคำสั่ง จากรูปจะเห็นว่าซอร์สโค้ดฝั่งซ้ายและฝั่งขวาจะมีการเรียงลำดับของซอร์สโค้ดภายในที่แตกต่างกัน

```
public class test {
   public int doY(){
      int x;
      x = 3;
      int z = 5;
      int y = 0;
      if(x==3){
         y = 5;
         System.out.println(y);
      }
      System.out.print(x);
      return 2;
   }
}
```

```
public class test {
   public int doY(){
      int x;
      int y = 0;
      int z = 5;
      x = 3;
      if(x==3){
      y = 5;
      System.out.println(y);
      }
      System.out.print(x);
   return 2;
   }
}
```

รูปที่ 4.12 ผลลัพธ์ของการเปรียบเทียบหาความแตกต่างในกรณีที่โหนดมีการเรียงลำดับที่แตกต่าง

ผลลัพธ์ของการเปรียบเทียบหาความแตกต่างในกรณีที่โหนดมีการเรียงลำดับที่แตกต่างกัน จะแสดงในรูปที่ 4.12 ซึ่งจากรูปจะเห็นว่าเครื่องมือจะแสดงส่วนที่แตกต่างโดยการทำไฮไลท์ในส่วนที่มีความแตกต่าง

```
public class test {
        public int doY(){
                int x;
                int y = 0;
                int z = 5;
                x = 3;
                if(x==3){
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
                return 2;
        }
}
```

```
public class test {
        public int doY(){
                int x;
                int y = 5;
                int z = 5;
                x = 10;
                if(x==3){
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
                return 2;
        }
}
```

รูปที่ 4.13 ซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่มีการแก้ไขข้อมูลภายในโหนด

จากรูปที่ 4.13 แสดงซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่มีการแก้ไขข้อมูลภายในโหนด ซึ่งในกรณีนี้จะเป็นการแก้ไขค่าของตัวแปรภายในซอร์สโค้ด โดยจากรูปจะเห็นว่าค่าของตัวแปร int y และ int x จะค่าของตัวแปรที่แตกต่างกัน

```
public class test {
   public int doY(){
      int x;
      int y = 0;
      int z = 5;
      x = 3;
      if(x==3){
      y = 5;
      System.out.println(y);
      }
      System.out.print(x);
   return 2;
   }
}
```

```
public class test {
   public int doY(){
      int x;
      int y = 5;
      int z = 5;
      x = 10;
      if(x==3){
      y = 5;
      System.out.println(y);
      }
      System.out.print(x);
   return 2;
   }
}
```

รูปที่ 4.14 ผลลัพธ์ของการเปรียบเทียบหาความแตกต่างในกรณีที่มีการแก้ไขข้อมูลภายในโหนด

รูปที่ 4.14 แสดงผลลัพธ์ของการเปรียบเทียบหาความแตกต่างในกรณีที่มีการแก้ไขข้อมูลภายในโหนดโดยเครื่องมือจะทำการไฮไลท์ในส่วนของค่าของตัวแปรที่มีความแตกต่างกันดังรูป

นอกจากการทดสอบเปรียบเทียบความแตกต่างทั้งในระดับ Method และในระดับโหนดในกรณีต่างๆที่กล่าวมาแล้วจะมีการทดสอบเปรียบเทียบความแตกต่างโดยการรวมเอากรณีต่างๆไว้ในซอร์สโค้ดชุดเดียวกัน

```
public class test {
        public int doA() {
                int x;
                int y = 0;
                x = 3;
                if(x==3){
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
                return 2;
        }

        public static void doB() {
                System.out.println("Method doB");
        }

        public string doZ(){
                return "Method doZ";
        }
}
```

```
public class test {
        public int doX(){
                int x;
                int y = 0;
                int z = 5;
                x = 3;
                if(x==3){
                        y = 5;
                        System.out.println(y);
                }
                System.out.print(x);
                return 2;
        }

        public int doY(){
                return 0;
        }

        public string doZ(){
                return "Method doZ";
        }
}
```

รูปที่ 4.15 ซอร์สโค้ดที่ใช้ทดสอบเปรียบเทียบหาความแตกต่างโดยรวมกรณีต่างๆไว้ในซอร์สโค้ดชุดเดียวกัน

จากรูปที่ 4.15 แสดงซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างโดยรวมกรณีต่างๆไว้ในซอร์สโค้ดชุดเดียวกัน สำหรับในซอร์สโค้ดที่จะใช้ทดสอบนี้มีจุดแตกต่างกันโดยเริ่มจากชื่อของ Method และภายในซอร์สโค้ดยังมีการเพิ่มเข้ามาของโหนด นอกจากนั้นยังมี Method ที่มี signature ที่แตกต่างกัน

```
public class test {
   public int doA() {
   int x;
      int y = 0;
      x = 3;
      if(x==3){
      y = 5;
      System.out.println(y);
      }
      System.out.print(x);
   return 2;
   }
   public static void doB() {
      System.out.println("Method doB");
   }
   public String doZ(){
      return "Method doZ";
   }
}
```

```
public class test {
   public int doX(){
   int x;
      int y = 0;
      int z = 5;
      x = 3;
      if(x==3){
      y = 5;
      System.out.println(y);
      }
      System.out.print(x);
   return 2;
   }
   public int doY(){
      return 0;
   }
   public String doZ(){
      return "Method doZ";
   }
}
```

รูปที่ 4.16 ผลลัพธ์ของการเปรียบเทียบหาความแตกต่างโดยรวมกรณีต่างๆไว้ในซอร์สโค้ดชุดเดียวกัน

จากรูปที่ 4.16 แสดงผลลัพธ์ของการเปรียบเทียบหาความแตกต่างโดยรวมกรณีต่างๆไว้ในซอร์สโค้ดชุดเดียวกัน ผลลัพธ์ที่ได้แสดงให้เห็นถึงความแตกต่างของซอร์สโค้ดทั้ง 2 ชุด โดยเครื่องมือได้ทำไฮไลท์ในส่วนที่แตกต่างเพื่อแสดงผลลัพธ์ที่ได้จากการเปรียบเทียบ

จากการทดสอบเปรียบเทียบความแตกต่างโดยการทดสอบกับซอร์สโค้ดในกรณีที่แตกต่างกัน เครื่องมือสามารถทำการหาความแตกต่างได้ตามกรณีต่างๆของซอร์สโค้ดที่ได้ออกแบบไว้ โดยสามารถแยกการอภิปรายได้เป็นกรณีต่างๆตามที่ได้ทดสอบได้ดังนี้

การทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการเพิ่ม จากการทดสอบจะเห็นได้ว่า เครื่องมือสามารถหาความแตกต่างในส่วนที่มีการเพิ่มเข้ามาในระดับ Method ได้ ความ

แตกต่างในกรณีที่ Method มีการเพิ่ม จะเกิดในกรณีที่ภายในซอร์สโค้ดทั้ง 2 ฝั่งมีจำนวน Method ไม่เท่ากันและซอร์สโค้ดฝั่งขวาจะต้องมีจำนวน Method มากกว่า Method ฝั่งซ้าย

การทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการลบ จากการทดสอบจะเห็นได้ว่าเครื่องมือสามารถหาความแตกต่างในส่วนที่มีการลบออกในระดับ Method ได้ สำหรับความแตกต่างในกรณีที่ Method มีการลบอก จะเกิดในกรณีที่ภายในซอร์สโค้ดทั้ง 2 ฝั่งมีจำนวน Method ไม่เท่ากันและซอร์สโค้ดฝั่งซ้ายมีจำนวน Method ฝั่งซ้ายมากขวาฝั่งขวา

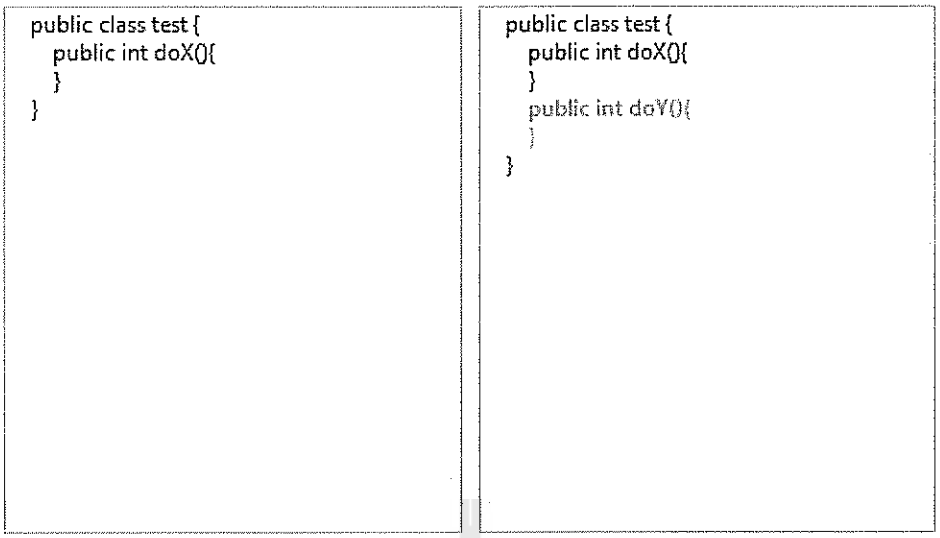
การทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการเรียงลำดับที่แตกต่าง จากการทดสอบจะเห็นได้ว่า เครื่องมือสามารถหาความแตกต่างได้ สำหรับความแตกต่างในกรณีนี้จะเกิดขึ้นก็ต่อเมื่อ Method ในซอร์สโค้ดมีการเรียงลำดับที่แตกต่างกัน

สำหรับในการส่วนของการเปรียบเทียบหาความแตกต่างนั้น ได้มีการใช้อัลกอริทึมอย่างง่ายในการเปรียบเทียบหาความแตกต่าง ซึ่งการเปรียบเทียบจะเป็นการเปรียบเทียบข้อมูลของโหนดแบบ 1:1 ด้วยข้อจำกัดดังกล่าวของอัลกอริทึม ทำให้เครื่องมือมีข้อจำกัดบางประการ

```
public class test {
        public int doX(){

        }
}
```

```
public class test {
        public int doY(){

        }
        public int doX(){

        }
}
```


รูปที่ 4.17 ซอร์สโค้ดที่ใช้ในการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการเพิ่ม



รูปที่ 4.18 ผลลัพธ์ที่ได้จากการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการเพิ่ม

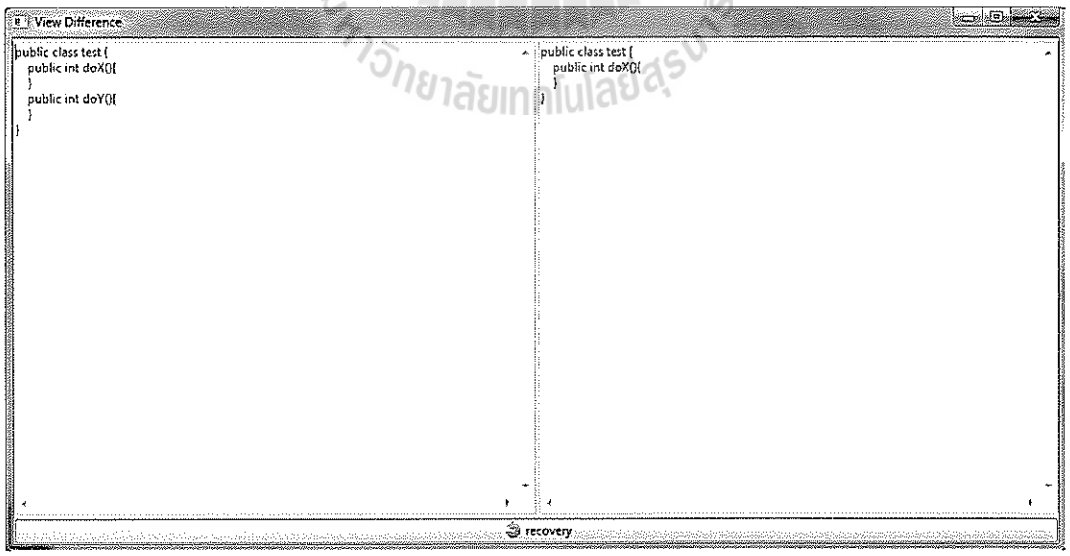
จากรูปที่ 4.17 และรูปที่ 4.18 จะเห็นได้ว่า ซอร์สโค้ดที่ใช้ทดสอบมีการเพิ่มเข้ามาของ Method ที่ชื่อว่า doY() แต่เมื่อทำการทดสอบแล้ว เครื่องมือแสดงผลลัพธ์ที่บอกว่า Method ที่มีการเพิ่มเข้ามาคือ Method ที่ชื่อ doX() และนอกจากเครื่องมือจะแสดงผลลัพธ์ที่มีการเพิ่มเข้ามาของ Method แล้ว ยังแสดงผลลัพธ์ที่เกิดจากความแตกต่างของการเปรียบเทียบ Method แบบการเรียงลำดับที่แตกต่าง โดยจากรูปจะเห็นได้ว่าเครื่องมือทำการไฮไลท์ในส่วนที่เป็นชื่อของ Method ที่ชื่อ doX() ด้วย สาเหตุเกิดจากข้อจำกัดของอัลกอริทึม ซึ่งเป็นการเปรียบเทียบแบบ 1:1 ทำให้เกิดข้อจำกัดในการเปรียบเทียบ สำหรับข้อจำกัดนี้จะเกิดขึ้นกับการทดสอบการเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการลบด้วย โดยผลลัพธ์ที่ได้จากการเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการลบจะได้ผลลัพธ์ออกมาในลักษณะเดียวกับการทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่ Method มีการเพิ่ม

สำหรับข้อจำกัดของอัลกอริทึมที่ได้กล่าวมาแล้ว จะเห็นได้ชัดเจนมากขึ้นเมื่อมีการทำการทดสอบเปรียบเทียบในระดับโหนด จากรูปที่ 4.7 4.8 ซึ่งเป็นซอร์สโค้ดที่ใช้ทดสอบเปรียบเทียบหาความแตกต่างในกรณีที่โหนดมีการเพิ่มและผลลัพธ์ของการทดสอบจะเห็นได้ว่า โหนดที่เครื่องมือแสดงไฮไลท์ว่ามีการเพิ่มเข้ามาของโหนดในซอร์สโค้ดฝั่งขวาคือโหนดของคำสั่ง return 2; ซึ่งเมื่อดูจากซอร์สโค้ดที่ใช้ในการเปรียบเทียบแล้วจะเห็นได้ว่า โหนดส่วนที่เพิ่มเข้ามาจริงๆคือโหนดของ

คำสั่ง int z = 5; ซึ่งเป็นโหนดที่มีการเพิ่มเข้ามาในซอร์สโค้ดทางฝั่งขวา แต่เครื่องมือแสดงผลลัพธ์ของการเปรียบเทียบว่าเป็นความแตกต่างในกรณีที่โหนดมีการเรียงลำดับแตกต่างกัน ซึ่งสาเหตุมาจากข้อจำกัดของอัลกอริทึมที่ใช้ โดยผลลัพธ์ที่เกิดในลักษณะนี้ จะเกิดขึ้นกับการเปรียบเทียบหาความแตกต่างในกรณีที่มีการลบของโหนดด้วย โดยสามารถดูได้จากรูปที่ 4.9 และ 4.10

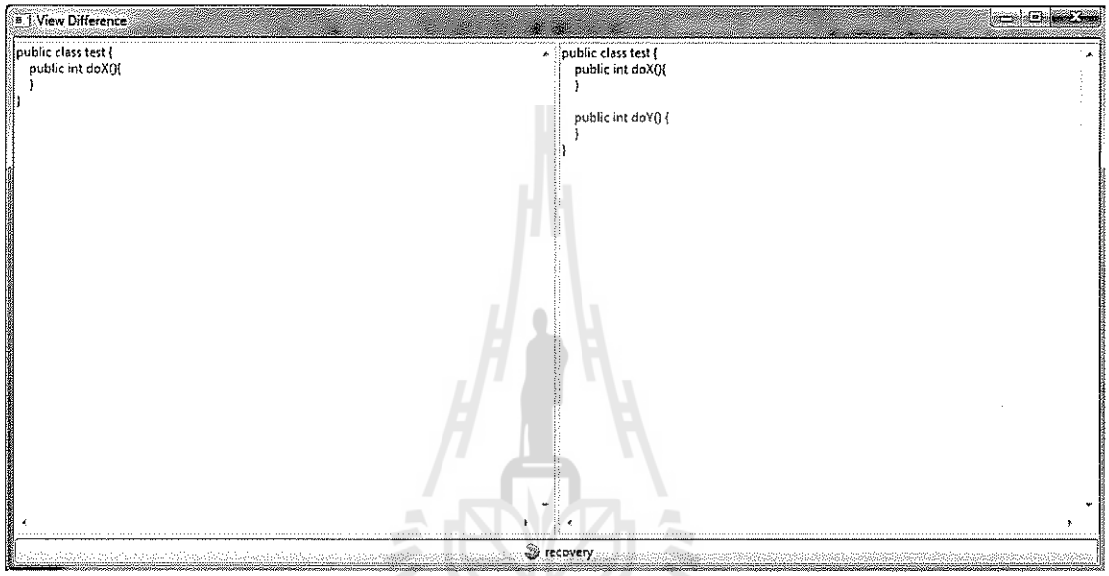
4.2 การทดสอบการกู้คืนข้อมูล

ในส่วนการทดสอบการกู้คืนข้อมูล จะเป็นการทดสอบเพื่อหาความถูกต้องของการกู้คืนซอร์สโค้ดโดยใช้เครื่องมือในการทำการกู้คืน หรือเรียกว่าเป็นการย้อนคืนของซอร์สโค้ด ในการทดสอบจะทดสอบโดยใช้ข้อมูลที่บันทึกไว้จากขั้นตอนการเปรียบเทียบมาทำการกู้คืน โดยจะแบ่งกรณีของการกู้คืนตามกรณีที่ได้ทดสอบในการเปรียบเทียบหาความแตกต่างของซอร์สโค้ด ผลลัพธ์ที่ได้จากการกู้คืนข้อมูลจะอยู่ในรูปแบบคลาสของภาษาจาวาซึ่งจะมีนามสกุล .java โดยก่อนการทำการกู้คืนจะมีหน้าจอของ User Interface แสดงผลลัพธ์ที่ได้หากทำการกู้คืนเพื่อให้ผู้ใช้ได้พิจารณาตัดสินใจก่อนจะทำการกู้คืน โดยในหน้าจอฝั่งซ้ายจะเป็นส่วนแสดงซอร์สโค้ดในเวอร์ชั่นล่าสุดที่ได้ทำการเปรียบเทียบไว้ ส่วนหน้าจอฝั่งขวาของเครื่องมือจะเป็นส่วนแสดงซอร์สโค้ดในเวอร์ชั่นที่ผู้ใช้ได้มีการทำการกู้คืน



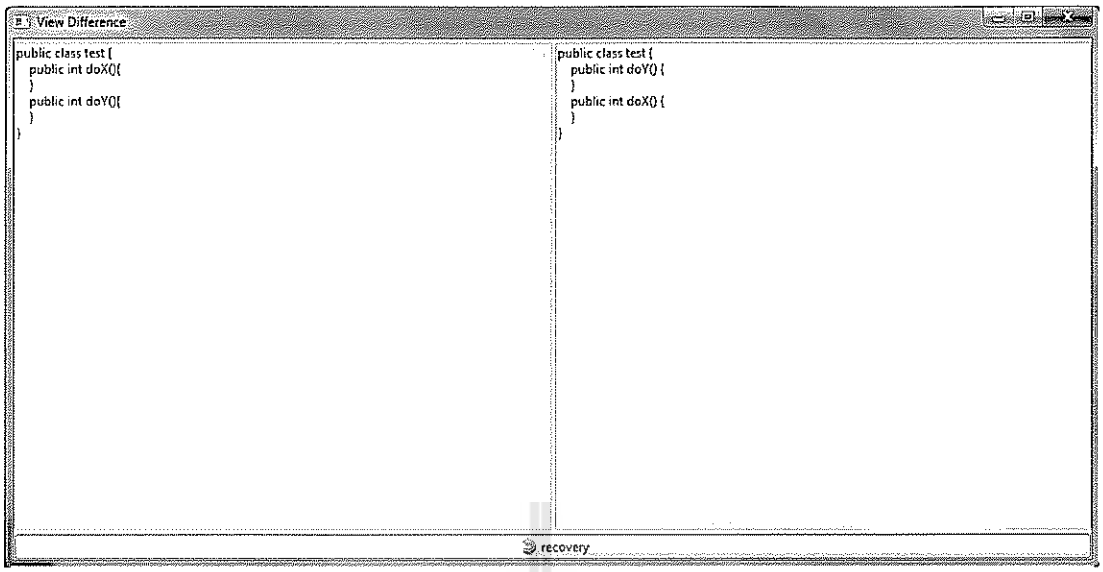
รูปที่ 4.19 ผลลัพธ์ที่ได้จากการกู้คืนในกรณีที่ Method มีการเพิ่ม

รูปที่ 4.19 แสดงผลลัพธ์ที่ได้จากการกู้คืนในกรณีที่ Method มีการเพิ่ม โดยจากซอร์สโค้ดฝั่งซ้ายในรูปซึ่งเป็นซอร์สโค้ดในเวอร์ชั่นล่าสุดที่ถูกเปรียบเทียบ ในการกู้คืนในกรณีนี้ เครื่องมือจะทำการลบ Method ที่มีการเพิ่มทำให้ได้ซอร์สโค้ดในเวอร์ชั่นฝั่งขวาซึ่งเป็นซอร์สโค้ดเวอร์ชั่นที่นำมาทำการเปรียบเทียบกับซอร์สโค้ดเวอร์ชั่นล่าสุด ซึ่งซอร์สโค้ดที่ได้ก็คือซอร์สโค้ดในฝั่งซ้ายในขั้นตอนของการเปรียบเทียบหาความแตกต่าง



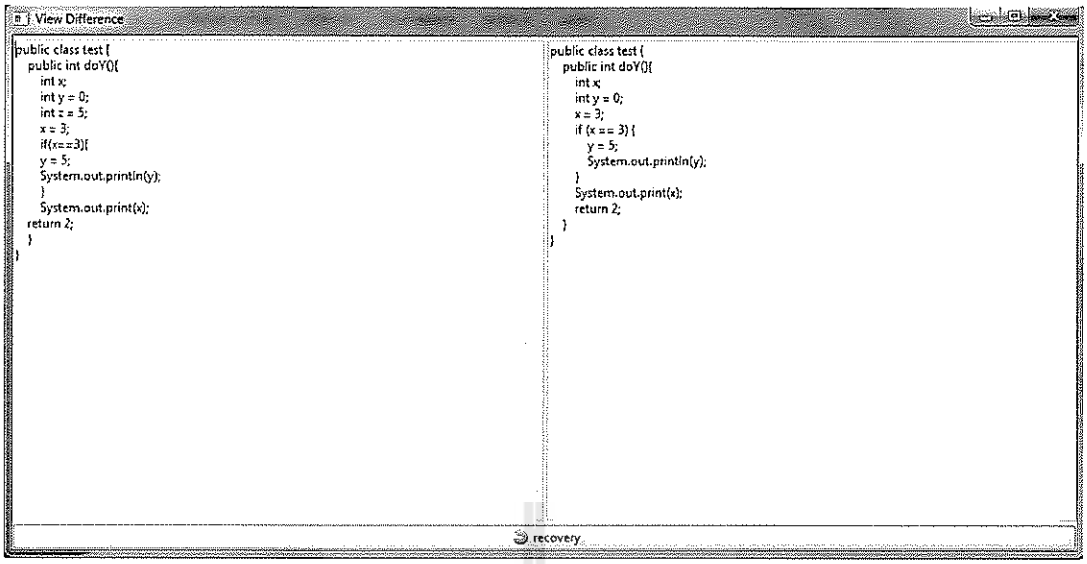
รูปที่ 4.20 ผลลัพธ์ที่ได้จากการกู้คืนในกรณีที่ Method มีการลบ

ในการกู้คืนซอร์สโค้ดที่เปรียบเทียบในกรณีที่ Method มีการลบ เครื่องมือจะทำการเพิ่ม Method ในส่วนที่ได้ถูกลบออกไปยังซอร์สโค้ดเวอร์ชั่นล่าสุดที่ได้เก็บไว้ ซึ่งเมื่อเพิ่มเข้ามาทำให้ได้ผลลัพธ์ของการกู้คืนดังรูปที่ 4.20 โดยซอร์สโค้ดฝั่งซ้ายในรูปเป็นซอร์สโค้ดเวอร์ชั่นล่าสุดที่ได้ทำการเก็บไว้ และซอร์สโค้ดฝั่งขวาคือผลลัพธ์ที่ได้จากการกู้คืนซึ่งก็คือซอร์สโค้ดฝั่งซ้ายในขั้นตอนของการเปรียบเทียบหาความแตกต่าง



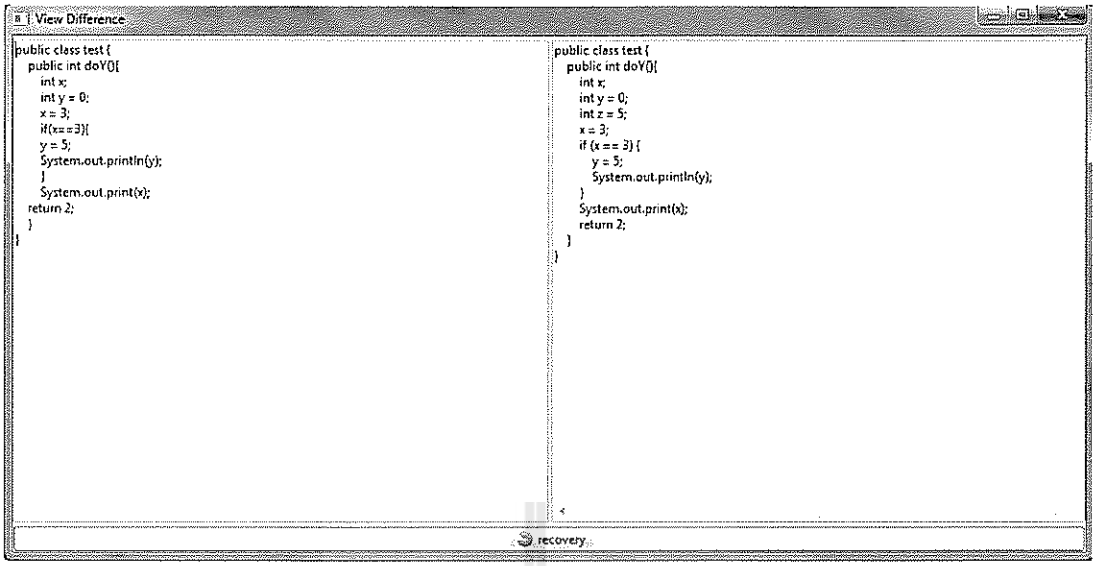
รูปที่ 4.21 ผลลัพธ์ที่ได้จากการกู้คืนในกรณีที่ Method มีการจัดเรียงลำดับที่แตกต่าง

สำหรับในการกู้คืนซอร์สโค้ดจากการเปรียบเทียบในกรณีที่ Method มีการเรียงลำดับที่แตกต่าง เครื่องมือจะทำการแทนที่ Method ที่มีความแตกต่างจากการเปรียบเทียบด้วย Method ที่ได้ทำการเก็บบันทึกไว้ จากรูปที่ 4.21 แสดงผลลัพธ์ที่ได้จากการกู้คืนในกรณีที่ Method มีการจัดเรียงลำดับที่แตกต่างกัน โดยจากรูปจะเห็นได้ว่าซอร์สโค้ดฝั่งซ้ายซึ่งเป็นซอร์สโค้ดเวอร์ชั่นล่าสุดใน Method ที่มีความแตกต่าง จะถูกแทนที่ด้วย Method ที่ได้ถูกบันทึกไว้ เมื่อทำการแทนที่แล้วผลลัพธ์ที่ได้จะเป็นซอร์สโค้ดในฝั่งขวาดังรูป



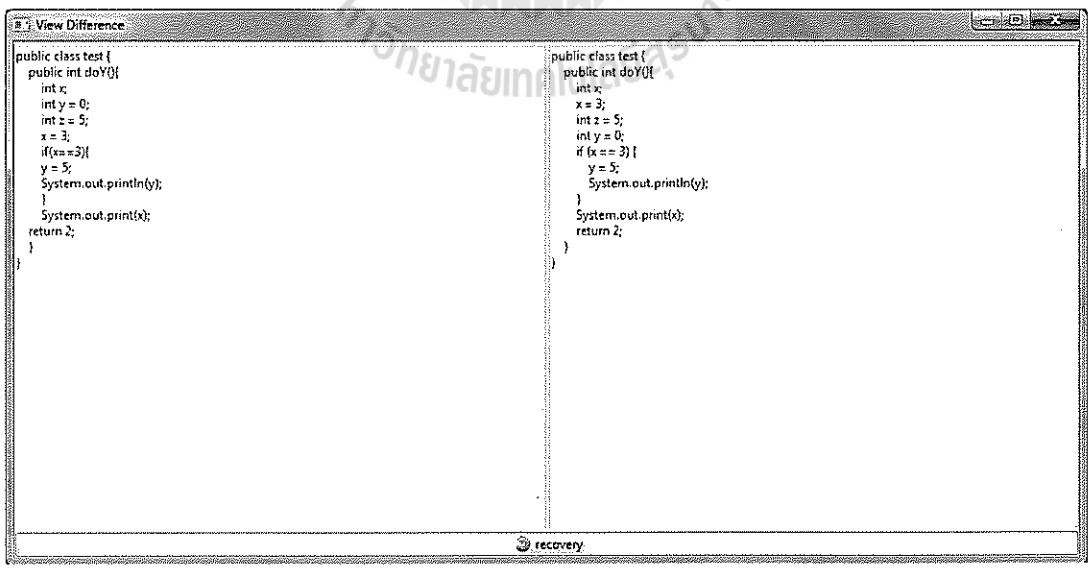
รูปที่ 4.22 ผลลัพธ์ที่ได้จากการกู้คืนในกรณีที่โหนดมีการเพิ่ม

ในการกู้คืนในระดับโหนดในกรณีที่โหนดมีการเพิ่มนั้น ผลลัพธ์ที่ได้จากการกู้คืนจะแสดงในรูปที่ 4.22 ซึ่งเครื่องมือจะทำการแทนที่ Method ที่มีการเปลี่ยนแปลงที่เกิดขึ้นในระดับโหนดด้วย Method ที่ได้ทำการเปรียบเทียบและบันทึกไว้ เนื่องจากว่า ในการบันทึกข้อมูลของการเปลี่ยนแปลงในระดับโหนดนั้น หากเกิดความแตกต่างไม่ว่ากรณีใดๆ เครื่องมือจะทำการตรวจหาว่าโหนดที่มีความแตกต่างนั้นอยู่ใน Method ใดแล้วทำการบันทึกข้อมูลของ Method เพื่อใช้ในการกู้คืน ซึ่งผลลัพธ์ที่ได้จะได้ซอร์สโค้ดในฝั่งขวาที่ได้มีการลบโหนดที่เพิ่มเข้ามาจากซอร์สโค้ดในฝั่งซ้าย



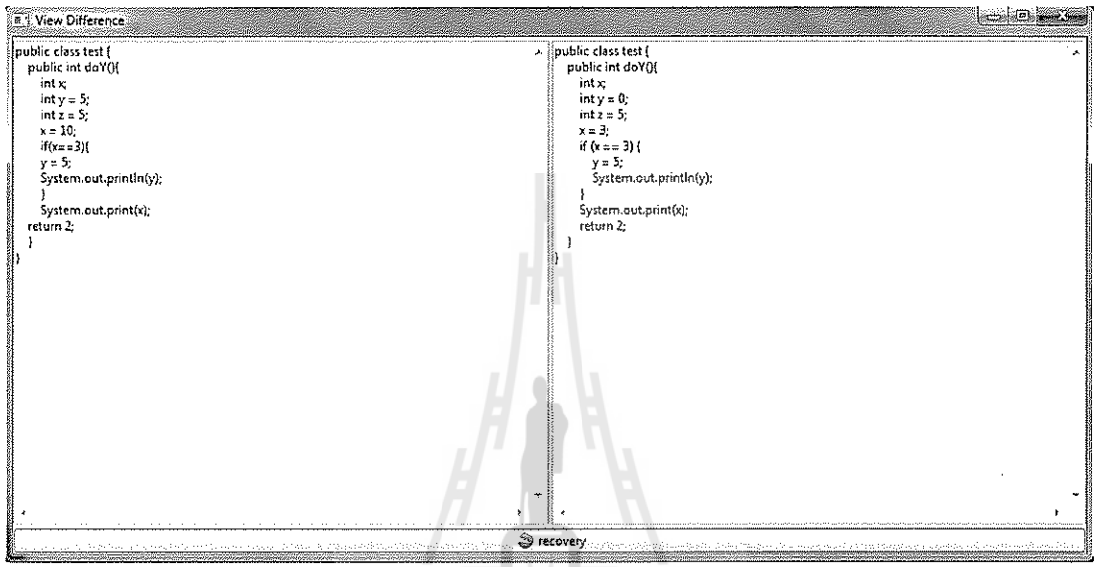
รูปที่ 4.23 ผลลัพธ์ที่ได้จากการกู้คืนในกรณีที่โหนดมีการลบ

จากรูปที่ 4.23 แสดงผลลัพธ์ที่ได้จากการกู้คืนในกรณีที่โหนดมีการลบ การทำงานของเครื่องมือในส่วนกู้คืนในกรณีนี้จะเหมือนกับการกู้คืนในกรณีที่โหนดมีการเพิ่ม ซึ่งก็คือการแทนที่ Method ที่มีความแตกต่างด้วย Method ที่ได้ทำการเปรียบเทียบและบันทึกข้อมูลไว้ ซึ่งผลลัพธ์ที่ได้ออกมาจะได้ซอร์สโค้ดในฝั่งขวาที่ได้มีการเพิ่มโหนดที่ถูกลบออกไปจากซอร์สโค้ดในฝั่งซ้าย

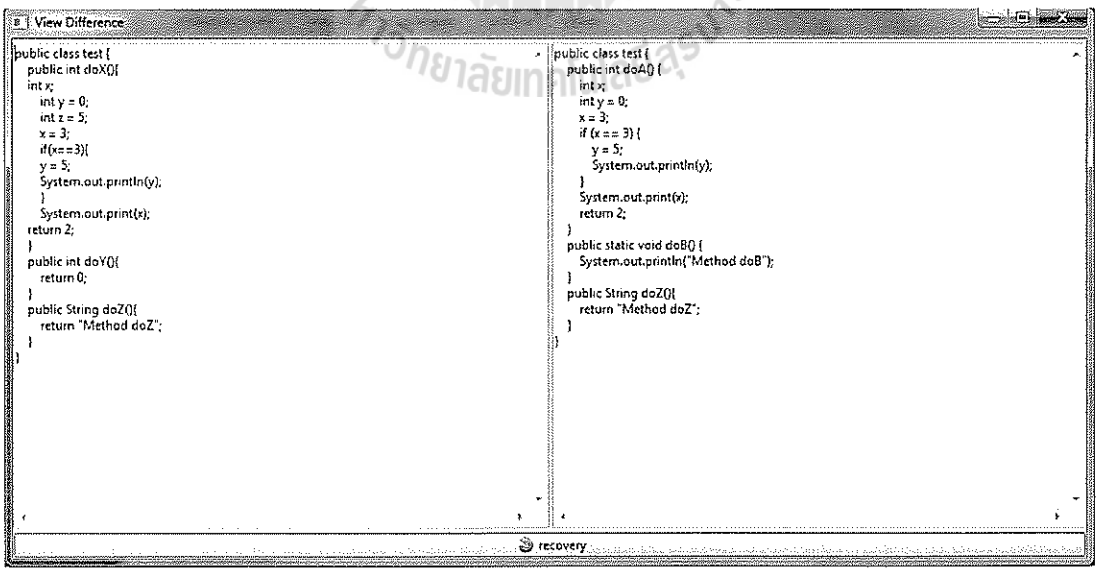


รูปที่ 4.24 ผลลัพธ์ที่ได้การกู้คืนในกรณีที่มีการจัดเรียงโหนดที่แตกต่าง

การกู้คืนซอร์สโค้ดในกรณีที่มีการจัดเรียงโหนดที่แตกต่าง ผลลัพธ์ที่ได้จะแสดงในรูปที่ 4.24 โดยเครื่องมือจะทำการแทนที่ Method ที่มีการเปลี่ยนแปลงในระดับโหนดด้วย Method ที่ได้ทำการเปรียบเทียบและบันทึกไว้ โดยการทำงานของเครื่องมือในลักษณะนี้จะรวมไปถึงการกู้คืนในกรณีที่มีการแก้ไขข้อมูลภายในโหนดดังรูปที่ 4.25 ด้วย



รูปที่ 4.25 ผลลัพธ์ที่ได้จากการกู้คืนในกรณีที่มีการแก้ไขข้อมูลภายในโหนด



รูปที่ 4.26 ผลลัพธ์ของการกู้คืนที่รวบรวมกรณีต่างๆไว้ในซอร์สโค้ดชุดเดียวกัน

ในรูปที่ 4.26 แสดงผลลัพของการกู้คืนที่รวบรวมกรณีต่างๆไว้ในซอร์สโค้ดชุดเดียวกัน โดยจากรูปเครื่องมือจะทำการแทนที่ Method ที่มีความแตกต่างด้วย Method ที่ได้ทำการเปรียบเทียบและบันทึกไว้เนื่องด้วย Method doX() และ Method doA() เป็นความแตกต่างในระดับ Method แต่เครื่องมือยังจะแสดงผลลัพธ์ในระดับโหนดในการเปรียบเทียบ ในการกู้คืนเครื่องมือจะทำการแทนที่ Method doX() ในซอร์สโค้ดฝั่งซ้ายด้วย Method doA() สำหรับ Method doY() และ Method doB() ยังคงเป็นความแตกต่างในระดับ Method และภายในระดับโหนดก็มีความแตกต่าง ดังนั้น เครื่องมือจะทำการแทนที่ Method doY() ด้วย Method doB() และสำหรับ Method doZ() จะเห็นว่าไม่มีความแตกต่าง เครื่องมือจึงไม่ดำเนินการใดๆ กับ Method doZ()

ในการทดสอบการกู้คืนที่ได้ทดสอบมาแล้วนั้น เป็นการทดสอบกู้คืนในลักษณะที่เวอร์ชั่นของ ซอร์สโค้ดที่ติดกัน สำหรับการทดสอบการกู้คืนในเวอร์ชั่นที่ไม่ติดกันนั้น จะใช้ซอร์สโค้ดดังรูปที่ 4.27 ในการทดสอบและทำการทดสอบกู้คืนเพื่อดูผลลัพธ์ที่ได้จากการทดสอบ

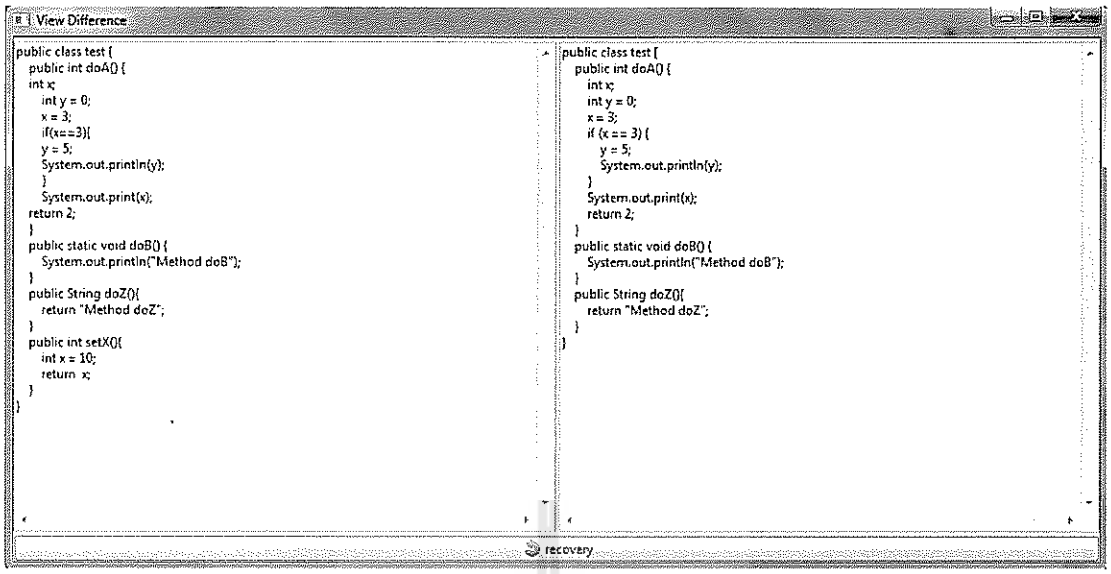
```
public class test {
        public int doA() {
                int x;
                int y = 0;
                x = 3;
                if(x==3){
                y = 5;
                System.out.println(y);
                }
                System.out.print(x);
                return 2;
        }

        public static void doB() {
                System.out.println("Method doB");
        }

        public String doZ(){
                return "Method doZ";
        }
}
```

```
public class test {
        public int doX(){
                int x;
                int y = 0;
                int z = 5;
                x = 3;
                if(x==3){
                y = 5;
                System.out.println(y);
                }
                System.out.print(x);
                return 2;
        }

        public int doY(){
                return 0;
        }

        public String doZ(){
                return "Method doZ";
        }
}
```

```
public class test {
        public int doA() {
                int x;
                int y = 0;
                x = 3;
                if(x==3){
                y = 5;
                System.out.println(y);
                }
                System.out.print(x);
                return 2;
        }

        public static void doB() {
                System.out.println("Method doB");
        }

        public String doZ(){
                return "Method doZ";
        }

        public int setX(){
                int x = 10;
                return x;
        }
}
```

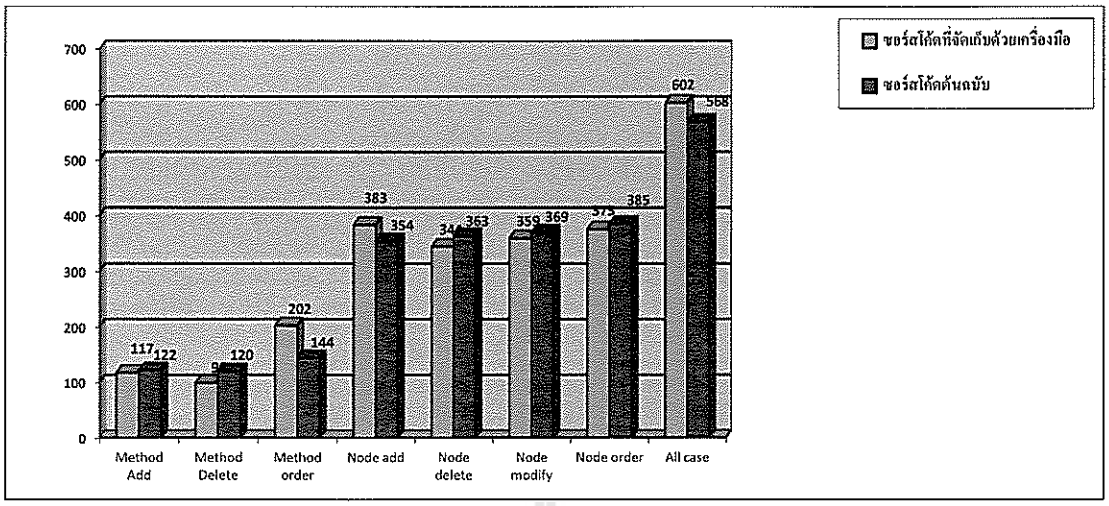
รูปที่ 4.27 ชุดของซอร์สโค้ดที่ใช้ทดสอบการกู้คืนในกรณีเวอร์ชั่นของซอร์สโค้ดไม่ติดกัน

จากรูปที่ 4.27 แสดงชุดของซอร์สโค้ดสำหรับใช้ทดสอบการกู้คืนในกรณีที่เวอร์ชั่นของซอร์สโค้ดไม่ติดกัน โดยขั้นตอนในการทดสอบจะทำการเปรียบเทียบซอร์สโค้ดทั้ง 3 โดยทำการเปรียบเทียบจาก ซอร์สโค้ดฝั่งซ้ายและซอร์สโค้ดตรงกลาง และทำการเปรียบเทียบซอร์สโค้ดเวอร์ชั่นตรงกลางกับซอร์สโค้ดทางฝั่งขวา ซึ่งเมื่อทำการเปรียบเทียบแล้วจะทำการทดสอบการกู้คืน โดยทำการย้อนกลับเวอร์ชั่นของซอร์สโค้ดเวอร์ชั่นฝั่งขวาย้อนกลับไปเป็นซอร์สโค้ดเวอร์ชั่นฝั่งซ้าย



รูปที่ 4.28 ผลลัพธ์ที่ได้จากการกู้คืนในกรณีเวอร์ชั่นของซอร์สโค้ดไม่ติดกัน

ผลลัพธ์ที่ได้จากการกู้คืนแสดงผลลัพธ์ดังรูปที่ 4.28 เครื่องมือสามารถทำการกู้คืนซอร์สโค้ดใน เวอร์ชั่นที่ไม่ติดกันได้ โดยในการกู้คืนในกรณีที่เวอร์ชั่นของซอร์สโค้ดไม่ติดกันนั้น เครื่องมือจะทำการกู้คืนซอร์สโค้ดในเวอร์ชั่นที่ติดกันไปเรื่อยๆจนถึงเวอร์ชั่นที่ผู้ใช้งานต้องการจึงจะแสดงผล โดยการทำงานของการกู้คืนนี้ เครื่องมือจะทำการกู้คืนซอร์สโค้ดจากฝั่งขวาไปยังซอร์สโค้ดในเวอร์ชั่นตรงกลางหลังจากนั้น เครื่องมือจะทำการนำซอร์สโค้ดเวอร์ชั่นตรงกลางมาเป็นซอร์สโค้ดเวอร์ชั่นตั้งต้นแทนซอร์สโค้ดฝั่งขวาและทำการอ่านข้อมูลที่ได้จากการเปรียบเทียบในระหว่างซอร์สโค้ดเวอร์ชั่นตรงกลางและซอร์สโค้ดเวอร์ชั่นฝั่งซ้ายจากนั้นจะทำการกู้คืนซอร์สโค้ดจากเวอร์ชั่นตรงกลางเป็นเวอร์ชั่นฝั่งซ้ายเมื่อได้ผลลัพธ์ตามที่ผู้ใช้ต้องการ ซึ่งในการทดสอบนี้คือซอร์สโค้ดเวอร์ชั่นฝั่งซ้าย เครื่องมือจะทำการแสดงผลลัพธ์ที่ได้และทำการลบซอร์สโค้ดเวอร์ชั่นตรงกลางที่ได้จากการกู้คืน



รูปที่ 4.29 แผนภูมิแท่งแสดงผลการเปรียบเทียบขนาดของซอร์สโค้ดที่จัดเก็บด้วยเครื่องมือและซอร์สโค้ดต้นฉบับ

จากรูปที่ 4.29 แสดงแผนภูมิแท่งแสดงผลการเปรียบเทียบขนาดของซอร์สโค้ดที่จัดเก็บด้วยเครื่องมือและซอร์สโค้ดต้นฉบับ โดยแนวแกน X คือกรณีต่างๆของซอร์สโค้ดที่ทำการเปรียบเทียบและแกน Y คือขนาดของเนื้อที่ที่ใช้ในการจัดเก็บซอร์สโค้ดมีหน่วยเป็น Bytes โดยในการเปรียบเทียบเนื้อที่ที่ใช้ในการจัดเก็บจะเปรียบเทียบเนื้อที่ของซอร์สโค้ดต้นฉบับกับเนื้อที่ของซอร์สโค้ดที่จัดเก็บโดยใช้เครื่องมือ สำหรับซอร์สโค้ดต้นฉบับนั้นจะเป็นซอร์สโค้ดในเวอร์ชั่นฝั่งซ้ายและฝั่งขวาของในแต่ละกรณีส่วนการจัดเก็บโดยใช้เครื่องมือนั้น เครื่องมือจะจัดเก็บในส่วนของซอร์สโค้ดเวอร์ชั่นล่าสุดและส่วนที่แตกต่างของซอร์สโค้ดเวอร์ชั่นที่นำมาเปรียบเทียบกับเวอร์ชั่นล่าสุด

จากแผนภูมิจะเห็นว่าในบางกรณีเครื่องมือใช้เนื้อที่ในการจัดเก็บซอร์สโค้ดน้อยกว่าซอร์สโค้ดต้นฉบับแต่ในบางกรณีซอร์สโค้ดต้นฉบับมีใช้เนื้อที่ในการจัดเก็บที่น้อยกว่า

ซอร์สโค้ดเวอร์ชั่นที่ 1 มีขนาด X Bytes

ซอร์สโค้ดเวอร์ชั่นที่ 2 มีขนาด Y Bytes

ดังนั้น เนื้อที่ของการจัดเก็บโดยไม่ใช้เครื่องมือ = X+Y Bytes

เนื้อที่ขอการจัดเก็บโดยใช้เครื่องมือ = Y+Diff Bytes

โดยที่ Diff = ส่วนที่แตกต่างของซอร์สโค้ด + รูปแบบการจัดเก็บ

ซึ่งจะเห็นได้ว่าการใช้เนื้อที่ในการจัดเก็บข้อมูลของเครื่องมือจะขึ้นอยู่กับ Diff ที่ได้จากการเปรียบเทียบหาความแตกต่าง ถ้า Diff > X เครื่องมือจะใช้เนื้อที่ในการจัดเก็บมากกว่าการจัดเก็บต้นฉบับหรือถ้า Diff < X เครื่องมือจะใช้เนื้อที่ในการจัดเก็บน้อยกว่าการจัดเก็บต้นฉบับ

จากการทดสอบการเปรียบเทียบหาความแตกต่างและทดสอบการกู้คืน ผลลัพธ์ที่ได้จากการทดสอบพบว่า เครื่องมือสามารถทำการเปรียบเทียบหาความแตกต่างในกรณีต่างๆ ได้และยังสามารถทำการกู้คืน ซอร์สโค้ดในเวอร์ชั่นที่ผู้ใช้งานต้องการ โดยอาศัยข้อมูลที่ได้จากการเปรียบเทียบเพื่อทำการสร้างซอร์สโค้ดในเวอร์ชั่นที่ผู้ใช้งานต้องการได้ ซึ่งเป็นการช่วยอำนวยความสะดวกให้กับผู้ใช้งานสามารถทำให้จัดการ ซอร์สโค้ดที่มีการเปลี่ยนแปลงได้อย่างเป็นระบบ แต่ทั้งนี้เครื่องมือนี้ยังมีข้อจำกัดในส่วนของการเปรียบเทียบหาความแตกต่างดังที่ได้กล่าวมาแล้วในส่วนการทดสอบเปรียบเทียบหาความแตกต่าง

บทที่ 5
สรุปและข้อเสนอแนะ

งานวิจัยนี้เป็นการพัฒนาเครื่องมือเพื่อช่วยในการกู้คืนซอร์สโค้ดสำหรับภาษาจาวา โดยเป็นเครื่องมือเพื่อช่วยให้ผู้พัฒนาซอฟต์แวร์เกิดความสะดวกในการจัดการกับซอร์สโค้ดในเวอร์ชั่นที่แตกต่างกัน ในการทดสอบจะแบ่งการทดสอบออกเป็น 2 การทดสอบคือ ทดสอบการเปรียบเทียบหาความแตกต่างของซอร์สโค้ดและทดสอบการกู้คืนซอร์สโค้ดที่ได้ทำการจัดเก็บโดยใช้เครื่องมือ ซึ่งสามารถสรุปผลการทดสอบได้ดังนี้

5.1 สรุปผลการทดสอบ

5.1.1 การทดสอบเปรียบเทียบหาความแตกต่างของซอร์สโค้ด

ในการทดสอบเปรียบเทียบหาความแตกต่างของซอร์สโค้ดสามารถแบ่งการเปรียบเทียบระดับของซอร์สโค้ดได้เป็นการเปรียบเทียบหาความแตกต่างในระดับ Method และการเปรียบเทียบหาความแตกต่างในระดับโหนด ซึ่งในการทดสอบเปรียบเทียบหาความแตกต่างของซอร์สโค้ดนั้น เครื่องมือสามารถทำการเปรียบเทียบและบ่งบอกจุดที่แตกต่างกันของซอร์สโค้ดได้โดยใช้อัลกอริทึมในการเปรียบเทียบแบบ 1:1 ในการเปรียบเทียบ แต่เนื่องด้วยข้อจำกัดของอัลกอริทึม ทำให้ในบางกรณีของการเปรียบเทียบเกิดการแสดงผลที่ผิดพลาดดังเช่นผลของการเปรียบเทียบในกรณีที่มีการเพิ่มโหนดและกรณีที่มีการลบออกของโหนดซึ่งเป็นการเปรียบเทียบในระดับโหนด นอกจากนี้ในการเปรียบเทียบอาจจะมีความไม่สะดวกในการเปรียบเทียบซอร์สโค้ดเนื่องจากกระบวนการของการเปรียบเทียบจำเป็นต้องใช้ซอร์สโค้ดในเวอร์ชั่นก่อนที่จะทำการแก้ไขเพื่อเป็นซอร์สโค้ดตั้งต้นสำหรับการเปรียบเทียบ อาจจะทำให้เกิดความไม่สะดวกต่อผู้ใช้งานได้

5.1.2 การทดสอบการกู้คืนซอร์สโค้ด

การทดสอบการกู้คืนซอร์สโค้ดจะใช้ข้อมูลที่ได้จากการทดสอบการเปรียบเทียบมาทำการกู้คืนโดยใช้เครื่องมือที่สร้างขึ้น ซึ่งผลการทดสอบเครื่องมือสามารถทำการกู้คืนซอร์สโค้ดจากข้อมูลที่ได้จากการทดสอบการเปรียบเทียบได้

นอกจากนี้ ในการทดสอบจะเห็นได้ว่า ในบางกรณีเครื่องมือมีการใช้พื้นที่ในการจัดเก็บที่มากกว่าการจัดเก็บซอร์สโค้ดต้นฉบับเนื่องด้วยรูปแบบของการจัดเก็บซึ่งอาจจะไปทำให้ใช้พื้นที่ในการจัดเก็บเพิ่มขึ้น แต่ทั้งนี้ทั้งนั้น เครื่องมือยังสามารถช่วยอำนวยความสะดวกในการจัดการซอร์สโค้ดให้เป็นระบบระเบียบ และยังสามารถกู้คืนซอร์สโค้ดที่ได้ทำการลบไปแล้วได้

5.2 ข้อเสนอแนะในงานวิจัย

ในงานวิจัยนี้เครื่องมือที่ได้ทำการพัฒนายังมีข้อจำกัดหลายๆอย่าง ฉะนั้นเพื่อให้เครื่องมือสามารถทำงานได้อย่างมีประสิทธิภาพเพิ่มขึ้นจึงน่าจะมีการวิจัยเพื่อพัฒนาเครื่องมือเพิ่มเติม อาทิ เช่น

- การปรับปรุงอัลกอริทึมที่ใช้ในการเปรียบเทียบเพื่อให้มีประสิทธิภาพเพิ่มขึ้นและเพื่อลดปัญหาทางด้านข้อจำกัดของการเปรียบเทียบนอกจากนี้ยังจะช่วยให้การแสดงผลลัพธ์ที่ได้จากการเปรียบเทียบเป็นไปได้อย่างถูกต้อง

- ปรับปรุงรูปแบบของการเก็บข้อมูลที่ได้จากการเปรียบเทียบ เพื่อลดขนาดของพื้นที่ที่ใช้สำหรับจัดเก็บข้อมูล

- การพัฒนาในส่วนของการเปรียบเทียบเพื่อให้การเปรียบเทียบสามารถทำได้ง่ายขึ้นโดยการทำให้เครื่องมือสามารถทำการจัดเก็บซอร์สโค้ดก่อนการแก้ไขได้เองและยังเป็นการเพิ่มความสะดวกต่อผู้ใช้เครื่องมือในการเปรียบเทียบจะจัดการกับซอร์สโค้ด

- ปรับปรุงเครื่องมือให้รองรับซอร์สโค้ดได้หลากหลายภาษาเพื่อนำไปประยุกต์ใช้ในการพัฒนาซอฟต์แวร์ในภาษาอื่นๆที่หลากหลายขึ้นเพื่อให้กระบวนการจัดเก็บและการจัดการของซอร์สโค้ดภาษาอื่นเป็นไปอย่างมีประสิทธิภาพและเป็นระบบ

รายการอ้างอิง

โอภาส เอี่ยมสิริวงศ์. **การวิเคราะห์และออกแบบระบบ System Analysis and Design.** กรุงเทพฯ:ซีเอ็ดยูเคชั่น, 2548.

Brian de Alwis, Jonathan Sillito. (2009). **Why Are Software Projects Moving From Centralized to Decentralized Version Control Systems?.** CHASE'09

Dirk Baumer, Phillipe Mulet. **Manipulating Java Programs.** [ออนไลน์]. ได้จาก http://www.eclipsecon.org/2004/EclipseCon_2004_TechnicalTrackPresentations/25_Baumer-Mulet.pdf

Gerado Canfora, Luigi Cerulo, Massimiliano Di Penta. (2007). **Identiflying Changed Source Code Lines from Version repositories.** Fourth International Workshop on Mining Software Repositories(MSR'07).

IBM Corporation and other. (2000). **Eclipse documentation – Archived Release.** [ออนไลน์]. ได้จาก http://help.eclipse.org/helios/index.jsp?topic= org.eclipse.jdt.doc.isv reference api org eclipse jdt core dom ASTParser.html

Ira D. Baxter, Andrew Yahim, Leonado Moura, Marcelo Sant'Anna, Lorraine Bier. (1998). **Clone Detection Using Abstract Syntax Trees.** ICSM'98.

Jennifer Vesperman. **Essential CVS.** O'Reilly, 2003.

Lars Vogel. (2009). **Eclipse JDT – Abstract Syntax Tree (AST) and the Java Model – Tutorial.** [ออนไลน์]. ได้จาก http://www.vogella.com/articles/EclipseJDT/article.html

Manoel Marques. (2005). **Exploring Eclipse's ASTParser How to use the parser to generate code.** [ออนไลน์]. ได้จาก http://www.ibm.com/developerworks/opensource/library/os-ast/

Martin Aesvhlinmann, Dirk Baumer, Jerome Lanneluc. (2005). **Java Tool Smithing Extending the Eclipse Java Development Tools.** Eclipse Conference 2005.

Miryung Kim, David Notkin. (2006). **Program Element Matching for Multi-Version Program Analyses.** MSR'06.

Programcreek.com. (2012). **Add Comments by using Eclipse JDT ASTRewrite.** [ออนไลน์]. ได้จาก http://www.programcreek.com/2012/06/add-comments-by-using-eclipse-jdt-astrewrite/

Rainer Koschke, Raimar Falke, Pierre Frenzel. (2006). **Clone Detection Using Abstract Syntax Suffix Trees.** In Proceesings of the 13th Working Conference on Reverse Engineering(WCRE'06).

Roger S. Pressman. **Software Engineering A Practitioner's approach.** 4th ed. McGraw Hill, 1997.

Taweesup Apiwattanapong, Alessandro Orso, Mary Jean Harrold. **A Differencing Algorithm for Object-Oriented Programs.** Georgia Institute of Technology Atlata, Geogia.

Thomas Kuhn, Olivier Thoman. (2006). **Abstract Syntax Tree.** [ออนไลน์]. ได้จาก http://www.eclipse.org/articles/Article-JavaCodeManipulation_AST/index.html

Timothy C., Lethbridege, Robert Laganiere. **Object-Oriented Software Engineering: Practical Software Development using UML and Java.** 2 nd ed. McGraw Hill, 2001.

Tobias Sager, Abraham Bernstein, Martin Pinzger, Chistoph Kiefer. (2006). **Detecting Similar Java Classes Using Tree Algorithm.** MSR'06.

WIKIPEDIA. (2012). **RevisionControl.** [ออนไลน์], ได้จาก http://en.wikipedia.org/wiki/Revision_control

WIKIPEDIA. (2012). **Software Configuration Management(SCM).** [ออนไลน์], ได้จาก http://th.wikibooks.org/wiki/Software_Configuration_Management

WIKIPEDIA. (2012). **Visitor pattern.** [ออนไลน์], ได้จาก http://en.wikipedia.org/wiki/Visitor_pattern

Wuu Yang. (1991). **Identifying syntactic differences between two programs.** Software Practice and Experience, 21(7) : 739-755.

ภาคผนวก ก

บทความทางวิชาการที่ได้รับการตีพิมพ์เผยแพร่ในระหว่างศึกษา

รายชื่อบทความที่ได้รับการตีพิมพ์เผยแพร่ในระหว่างการศึกษา

สมคะเน บาลลา และผู้ชายศาสตราจารย์ ดร.พิชโยทัย มหัทธนาภิวัฒน์. (ธันวาคม 2554). การหาความแตกต่างของซอร์สโค้ดโดยใช้โครงสร้างไวยากรณ์ต้นไม้(DIFFERENCE DETECTION USING ABSTRACT SYNTAX TREES). ใน **การประชุมวิชาการเสนอผลงานวิจัยระดับบัณฑิตศึกษาแห่งชาติครั้งที่ 23: บัณฑิตศึกษาไทยสู่ประชาคมอาเซียน.** นครราชสีมา: มหาวิทยาลัยเทคโนราชมงคลอีสาน., 4 หน้า

การหาความแตกต่างของซอร์สโค้ดโดยใช้โครงสร้างไวยากรณ์ต้นไม้
Difference Detection Using Abstract Syntax Trees

สมคะเน บาลลา* พิชโยทัย มหัทธนาภิวัฒน์

สาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี อ.เมือง จ.นครราชสีมา 30000
** E-mail: smkn.balla@gmail.com*

บทคัดย่อ

ในการพัฒนาซอฟต์แวร์ การเก็บสำรองซอร์สโค้ดเป็นเรื่องที่มีความสำคัญ ซอร์สโค้ดที่เก็บสำรองไว้นั้นอาจจะมีความแตกต่างกันในระหว่างเวอร์ชั่นต่างๆที่ได้ทำการเก็บสำรองจึงอาจจะนำมาซึ่งปัญหาในการแยกแยะความแตกต่างของซอร์สโค้ดทำให้เกิดความไม่สะดวกต่อนักพัฒนาและเป็นการเสียเวลาในการหาความแตกต่างระหว่างเวอร์ชั่นของซอร์สโค้ด ดังนั้นงานวิจัยนี้จึงได้นำเสนอเทคนิคในการประยุต์ใช้ Abstract Syntax Tree ในการเปรียบเทียบหาความแตกต่างของซอร์สโค้ดที่ต่างเวอร์ชั่นกันเพื่อเป็นทางเลือกและแนวทางในการนำไปใช้พัฒนาเครื่องมือสำหรับเปรียบเทียบหาความแตกต่างของซอร์สโค้ดสำหรับนักพัฒนาเพื่อใช้แก้ปัญหาข้างต้นดังที่กล่าวมาแล้ว

คำสำคัญ : Java Code, Differencing Algorithm, Abstract Syntax Tree

บทนำ

ในปัจจุบันการเก็บสำรองซอร์สโค้ดในระหว่างการพัฒนาซอฟต์แวร์เป็นเรื่องที่มีความสำคัญ ในการสำรองถ้าหากไม่มีกระบวนการในการแยกแยะเวอร์ชั่นที่มีประสิทธิภาพพอ อาจจะทำให้เสียเวลาและเกิดความไม่สะดวกในการหาความแตกต่างของเวอร์ชั่นของซอร์สโค้ดซึ่งปัญหาดังกล่าวอาจจะลุกลามทำให้การพัฒนาซอฟต์แวร์เกิดความล้าช้าและไม่มีประสิทธิภาพได้ ดังนั้น ทางผู้วิจัยจึงได้นำเสนอวิธีการเปรียบเทียบและหาความแตกต่างของซอร์สโค้ดโดยการนำโครงสร้างไวยากรณ์มาประยุกต์ใช้ในการหาเปรียบเทียบหาความแตกต่างของซอร์สโค้ดต่างเวอร์ชั่นกัน

ในเอกสารงานวิจัยนี้ จะอธิบายถึงเทคนิคที่ใช้ในการเปรียบเทียบหาความแตกต่างของซอร์สโค้ดภาษาจาวาโดยใช้ Syntax Tree Matching Technique ในการหาความแตกต่างเพื่อเป็นแนวทางสำหรับนำไปพัฒนาเครื่องมือที่ใช้ในการเปรียบเทียบหาความแตกต่างของซอร์สโค้ด

งานวิจัยที่เกี่ยวข้อง

จากงานวิจัยของ Miryung Kim, David Notkin[1] ได้รวบรวมและสำรวจเกี่ยวกับความสามารถและคุณสมบัติของ Matching Technique ซึ่งเป็นเทคนิคที่ใช้ในการวิเคราะห์เวอร์ชั่นของซอฟต์แวร์โดยเทคนิคที่ได้ทำการสำรวจมีดังนี้ Entity Name Matching, String Matching, Syntax Tree Matching, Control Flow Graph Matching, Program Dependence Graph Matching, Binary Code Matching, Clone Detection, Origin Analysis Tools โดยแต่ละเทคนิคนั้นมีข้อดีข้อเสียและผลกระทบที่แตกต่างกัน

Syntax Tree Matching เป็นเทคนิคที่ใช้ในการวิเคราะห์หาเวอร์ชั่นของซอฟต์แวร์โดยการใช้ Syntax Tree มาช่วยในการวิเคราะห์ความแตกต่างของแต่ละเวอร์ชั่น Syntax Tree หรือ Abstract Syntax Tree เป็นการแสดงโครงสร้างของซอร์สโค้ดในแต่ละภาษาในรูปแบบของโครงสร้างต้นไม้ โดยในแต่ละโหนดของโครงสร้างต้นไม้จะรวบรวมข้อมูลต่างๆไว้ เช่น ชื่อตัวแปร, ประเภทของตัวแปร เป็นต้น ในงานวิจัยของ Raimar Koschke, Raimar Falke, Pierre Frenzel[2] ได้นำเสนองานวิจัยที่ใช้ Suffix Tree มาใช้ในการหา Clone ใน Abstract Syntax Tree โดยใช้ Suffix Tree Detection อัลกอริทึมในการหา Clone ในงานวิจัยได้กล่าวไว้ว่า Suffix Tree เป็น String ที่แสดงจาก Root node ไปยัง Leaf node โดยปกติ Suffix Tree Detection เป็นอัลกอริทึมแบบ Based on token โดยงานวิจัยนี้ได้ประยุกต์โดยการจัดเรียง AST Node ที่ได้จากการกระจายนิพจน์ของซอร์สโค้ดให้อยู่ในรูปแบบที่ต้องการ จากนั้นทำการแปลง AST Node แต่ละโหนดให้อยู่ในรูปแบบของ String Based โดยใช้ Ukkonen อัลกอริทึม แล้วจึงใช้ Baker อัลกอริทึมในการหา Clone

Eclipse Abstract Syntax Tree คือกรอบงานพื้นฐาน (base framework) สำหรับเครื่องมือของ Eclipse IDE ซึ่งรวมเอาความสามารถในการ Refactoring, Quick fix และ Quick

Assist. โดย Eclipse Abstract Syntax Tree จะจัดรูปแบบของซอร์สโค้ด ให้อยู่ในรูปแบบของโครงสร้างต้นไม้ (Tree)

Abstract Syntax Tree เป็นแนวทางในการเข้าถึงซอร์สโค้ดภาษาจาวาของ Eclipse IDE โดยทุกๆ จาวาโค้ดจะอยู่ในรูปแบบของโหนดของ AST โดยแต่ละโหนดจะเป็นคลาสย่อยของ ASTNode คลาส โดยในทุกๆคลาสย่อยของ ASTNode จะระบุส่วนประกอบต่างๆของ Java Programming Language ตัวอย่างเช่น โหนดของการประกาศเมธอด (MethodDeclaretion) โดยคลาสของ AST ที่เกี่ยวข้องจะอยู่ในแพ็คเกจ org.eclipse.jdt.cor.dom

จากการศึกษางานวิจัยที่เกี่ยวข้องจะเห็นได้ว่า การวิเคราะห์หาความเหมือนหรือความแตกต่างสามารถนำเทคนิคหลายๆเทคนิคมาประยุกต์รวมกันเพื่อใช้วิเคราะห์หาความเหมือนหรือความแตกต่างของซอร์สโค้ดได้ ในงานวิจัยนี้จะทำการวิเคราะห์หาความแตกต่างของซอร์สโค้ดสองเวอร์ชันโดยใช้ Syntax Tree Matching Tecnique และทำการเปรียบเทียบแบบ String Based

ขั้นตอนการดำเนินงาน

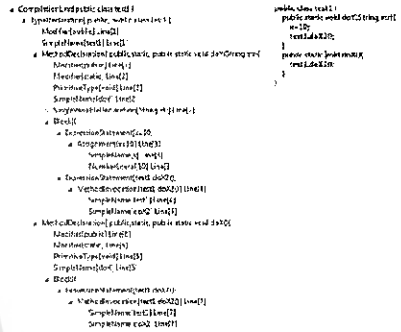
ในขั้นตอนการพัฒนาจะประกอบไปด้วยขั้นตอนต่างๆ ดังนี้

1. ทำการกระจายนิพจน์ของซอร์สโค้ดและทำการสร้าง Abstract Syntax Tree(AST)
2. ทำการจัดรูปแบบของ AST และกรองเอาส่วนที่จำเป็นสำหรับการเปรียบเทียบ
3. ทำการเปรียบเทียบ AST เพื่อวิเคราะห์หาความแตกต่างของซอร์สโค้ด

ในการกระจายนิพจน์นั้น เริ่มแรกทำการนำเข้าซอร์สโค้ดภาษาจาวา (.java) โดยซอร์สโค้ดที่นำเข้ามานั้นต้องเป็นซอร์สโค้ดที่มีการเขียนอย่างถูกต้องตามหลักไวยากรณ์ของภาษา จากนั้นทำการกระจายนิพจน์โดยใช้ ASTParser ซึ่งเป็นคลาสที่ช่วยทำการกระจายนิพจน์ แล้วทำการคัดกรองซอร์สโค้ดที่ได้จากการกระจายนิพจน์โดยการใช้ ASTVisitor โดยส่วนที่จำเป็นสำหรับการเปรียบเทียบจะประกอบไปด้วย

- ชื่อแพ็คเกจและชื่อคลาสหรืออินเตอร์เฟซ โดยในส่วนนี้จะเก็บในรูปแบบของ fully-qualified name โดย fully-qualified name นั้นจะประกอบไปด้วย ชื่อแพ็คเกจและตามด้วยชื่อคลาสหรือชื่ออินเตอร์เฟซ
- Mehtod Signature
- การประกาศตัวแปรพื้นฐาน เช่น int, char, String, float เป็นต้น
- คำสั่งควบคุมต่างๆ เช่น if else, for, while, do, try-catch, switch เป็นต้น
- คำสั่งต่างๆที่อยู่ภายใต้เงื่อนไขของคำสั่งควบคุม

จากนั้นทำการจัดรูปแบบของข้อมูลที่ได้จากการกระจายนิพจน์ให้อยู่ในรูปแบบดังภาพที่ 1 โดยในงานวิจัยนี้จะจัดรูปแบบของซอร์สโค้ดที่ผ่านการกระจายนิพจน์แล้วให้อยู่ในรูปแบบของโครงสร้างต้นไม้ โดยทำการจัดเรียงโหนดที่ได้จากการกระจายนิพจน์เรียงลำดับจากรูทโหนดไปยังโหนดล่างสุดของแต่ละ Block จากนั้นเก็บข้อมูลที่ทำการจัดเรียงไว้ใน stack เพื่อส่งไปทำการเปรียบเทียบ



รูปที่ 1 ซอร์สโค้ดที่ผ่านการกระจายนิพจน์และทำการจัดรูปแบบแล้ว

หลังจากทำการกระจายนิพจน์และทำการจัดรูปแบบของ AST แล้ว จากนั้นทำการเปรียบเทียบหาความแตกต่างโดยใช้อัลกอริทึมดังรูปที่ 2

```
input: source code L side stL
       source R side stR
procudure: ASTDiff(stL, stR)
while(stL is not empty) do
       getSignature(stL)
       getSignature(stR)
       if(signatureL equal signatureR) then
              if(nodeL is equal nodeR) then
             print nodeL
             print nodeR
                else
             skipNode(nodeL,nodeR)
              end if
       else
              skipMethod(nodeL)
       end if
if(nodeR is empty)  then
       break;
end while
if(nodeL is empty) then
       add caption to nodeR
else
       add cation to nodeL
end if
end procudure
```

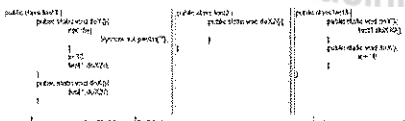
รูปที่ 2 อัลกอริทึมที่ขึ้นในการเปรียบเทียบซอร์สโค้ด

จากอัลกอริทึมในรูปที่สอง อัลกอริทึมจะทำการรับข้อมูลเข้าเป็น stL และ stR ซึ่งเป็น Stack ที่เก็บโหนดที่ได้จากการทำการกระจายนิพจน์ของซอร์สโค้ด โดย stL จะเก็บข้อมูลของซอร์สโค้ดทางฝั่งซ้ายและ stR จะเก็บซอร์สโค้ดทางฝั่งขวา จากนั้นทำการดึงข้อมูลใน Stack ทั้งสองออกมาเปรียบเทียบกัน โดยในการเทียบจะทำการเทียบโดยใช้วิธีการเปรียบเทียบแบบ String Based

การเปรียบเทียบข้อมูลใน Stack ทั้งสองจะเริ่มจากการเปรียบเทียบในระดับเมธอด โดยการเปรียบเทียบ signature ของเมธอดหาก signature ของเมธอดมีความแตกต่างกันก็จะทำการข้ามไปยังเมธอดถัดไป แต่ถ้าหาก signature ที่เปรียบเทียบเหมือนกันก็จะข้ามไปโหนดถัดไปซึ่งเป็นโหนดลูกและนำมาทำการเปรียบเทียบโดยทำการเปรียบเทียบแบบ String Based เหมือนกับการเปรียบเทียบ signature ของเมธอด โดยทำแบบนี้ไปเรื่อยๆ จนครบตามจำนวนข้อมูลที่เก็บอยู่ใน Stack หลังจากเปรียบเทียบแล้ว ถ้ามีโหนดที่เหลือจากการเปรียบเทียบก็จะทำการเพิ่มป้ายประกาศเข้าไปว่าโหนดที่เหลือเป็นโหนดที่ถูกเพิ่มหรือโหนดที่ถูกลบออกไปจากซอร์สโค้ด

ผลการศึกษาและอภิปราย

ในการทดสอบเพื่อหาความแตกต่างของซอร์สโค้ดโดยใช้ Syntax Tree Matching Technique โดยการทดสอบนั้นจะใช้ซอร์สโค้ดภาษาจาวาจำนวนสามซอร์สโค้ดโดยแบ่งเป็นสองชุดในการทดสอบ โดยซอร์สโค้ดทั้งสามจะมีความแตกต่างกันในระดับเมธอดไปจนถึงระดับโหนด เพื่อใช้ทดสอบในระดับเมธอดจะใช้ซอร์สโค้ดชุดที่หนึ่งซึ่งเป็นซอร์สโค้ดที่มีการประกาศเมธอดแต่ไม่มีคำสั่งภายในเมธอดในการทดสอบนี้จะเป็นการทดสอบหาความแตกต่างในระดับเมธอด สำหรับการทดสอบในระดับโหนด จะใช้ชุดทดสอบชุดที่สองซึ่งเป็นชุดของซอร์สโค้ดที่มีการประกาศเมธอดเหมือนกันแต่มีความแตกต่างกันของชุดคำสั่งภายในเมธอดซึ่งการทดสอบนี้เป็นการทดสอบหาความแตกต่างในระดับโหนด



รูปที่ 3 ซอร์สโค้ดที่ใช้ทำการทดสอบและจุดที่มีความแตกต่างกันในซอร์สโค้ด

จากการทดสอบโดยใช้ซอร์สโค้ดที่เตรียมมาพบว่า Syntax Tree Matching Technique สามารถใช้ทำการหาความแตกต่างของซอร์สโค้ดได้อย่างถูกต้องโดยสามารถระบุจุดที่มีความแตกต่างกันของซอร์สโค้ดได้ จากรูปที่ 4 เมื่อทำการทดสอบจะเห็นได้ว่าจุดที่มีความแตกต่างกันของซอร์สโค้ดจะถูกทำการข้าม (skip) โดยการข้ามจะทำการหารูทโหนดของจุดที่แตกต่างของซอร์สโค้ดแล้วทำการข้ามทั้งรูทโหนดและทำการเริ่มเปรียบเทียบที่รูทโหนดถัดไป

```
skip node
1210071443619
Directory: 1210071443629 created
[Parent node R] Block
- - - - - - - - - - - - - - - - MethodInvocation [skipR]
- - - - - - - - - - - - - - - - - - SimpleName [skipR]
- - - - - - - - - - - - - - - - - - SimpleName [skipR]
skip node
1210071443619
[Parent node L] Block
- - - - - - - - - - - - - - - - InfixExpression [skipL]
- - - - - - - - - - - - - - - - - - SimpleName [skipL]
- - - - - - - - - - - - - - - - - - NumberLiteral [skipL]
- - - - - - - - - - - - - - - - Block [skipL]
- - - - - - - - - - - - - - - - - - ExpressionStatement [skipL]
- - - - - - - - - - - - - - - - - - - - MethodInvocation [skipL]
- - - - - - - - - - - - - - - - - - - - - - QualifiedName [skipL]
- - - - - - - - - - - - - - - - - - - - - - - - SimpleName [skipL]
- - - - - - - - - - - - - - - - - - - - - - - - SimpleName [skipL]
- - - - - - - - - - - - - - - - - - - - SimpleName [skipL]
- - - - - - - - - - - - - - - - - - - - StringLiteral [skipL]
- - - - - - - - - - - - ExpressionStatement [skipL]
- - - - - - - - - - - - - - Assignment [skipL]
- - - - - - - - - - - - - - - - SimpleName [skipL]
- - - - - - - - - - - - - - - - NumberLiteral [skipL]
- - - - - - - - - - - - ExpressionStatement [skipL]
- - - - - - - - - - - - - - MethodInvocation [skipL]
- - - - - - - - - - - - - - - - SimpleName [skipL]
- - - - - - - - - - - - - - - - SimpleName [skipL]
```

รูปที่ 4 ผลลัพธ์ที่ได้จากการหาความแตกต่างโดยใช้ Syntax Tree Matching Technique

นอกจากนี้ยังสามารถระบุจุดที่มีการเพิ่มหรือการลบบรรทัดของซอร์สโค้ด โดยในการทดสอบได้มีการแก้ไขซอร์สโค้ดที่นำมาทำการทดสอบโดยการทำการเพิ่มจำนวนบรรทัดของซอร์สโค้ดในเวอร์ชั่นฝั่งขวา และทำการเปรียบเทียบซึ่งผลลัพธ์ที่ได้แสดงในรูปที่ 5

```
[delete]- - - - - - - - - Modifier
[delete]- - - - - - - - - Modifier
[delete]- - - - - - - - - PrimitiveType
[delete]- - - - - - - - - SimpleName
[delete]- - - - - - - - - Block
[delete]- - - - - - - - - - - - ExpressionStatement
[delete]- - - - - - - - - - - - - - MethodInvocation
[delete]- - - - - - - - - - - - - - - - SimpleName
[delete]- - - - - - - - - - - - - - - - SimpleName
```

รูปที่ 5 ผลลัพธ์ของการเปรียบเทียบแสดงให้เห็นการลบออกของบรรทัดในซอร์สโค้ด

สรุปผลการศึกษาวิจัย

ในการเปรียบเทียบซอร์สโค้ดโดยใช้ Syntax Tree Matching Technique สามารถใช้ทำการหาความแตกต่างของซอร์สโค้ดได้อย่างถูกต้องสามารถใช้งานได้และใช้เวลาไม่มากในการทำการหาความแตกต่างกันระหว่างซอร์สโค้ดสองเวอร์ชั่น ทั้งนี้เนื่องจากว่าซอร์สโค้ดที่นำมาทดสอบมีความซับซ้อนต่ำและมีจำนวนบรรทัดที่ไม่มากจนเกินไป อย่างไรก็ตามหากทำการทดสอบกับซอร์สโค้ดที่มีความซับซ้อนมากกว่านี้อาจทำให้ได้ผลลัพธ์ที่แตกต่างและอาจจะใช้เวลาเพิ่มขึ้นในการหาความแตกต่างของซอร์สโค้ดโดยใช้ Syntax Tree

Matching Technique โดยในอนาคตจะทำการทดสอบกับซอร์สโค้ดที่มีความซับซ้อนเพิ่มขึ้นและจำนวนบรรทัดที่เพิ่มขึ้น และทำการพัฒนาให้สามารถนำไปใช้ได้จริงในการเปรียบเทียบหาความแตกต่างของซอร์สโค้ดโดยการสร้างเป็นเครื่องมือในการหาความแตกต่างเพื่อช่วยให้การพัฒนาซอฟต์แวร์มีประสิทธิภาพเพิ่มขึ้นและลดระยะเวลาในการหาความแตกต่างของซอร์สโค้ดที่ต่างเวอร์ชั่นกันของโปรแกรมเมอร์

เอกสารอ้างอิง

[1] Miryung Kim, David Notkin. 2006. **Program Element Matching for Multi-Version Program Analyses.** MSR'06.

[2] Rainer Koschke, Raimar Falke, Pierre Frenzel. 2006. **Clone Detection Using Abstract Syntax Suffix Trees.** In Proceesings of the 13th Working Conference on Recerse Engineering(WCRE'06).

[3] Taweesup Apiwattanapong, Alessandro Orso, Mary Jean Harrold. **A Differencing Algorithm for Object-Oriented Programs.** Georgia Institute of Technology Atlata, Geogia.

[4] Martin Aesvhlinmann, Dirk Baumer, Jerome Lanneluc. 2005. **Java Tool Smithing Extending the Eclipse Java Development Tools.** Eclipse Conference 2005.

[5] Gerado Canfora, Luigi Cerulo, Massimiliano Di Penta. 2007. **Identiflying Changed Source Code Lines from Version repositories.** Fourth International Workshop on Mining Software Repositories(MSR'07).

ประวัติผู้เขียน

นายสมคะเน บาลลา เกิดเมื่อวันที่ 24 มกราคม พ.ศ. 2529 เริ่มศึกษาชั้นประถมศึกษาที่ 1-6 ที่โรงเรียนอนุบาลเลย ชั้นมัธยมศึกษาชั้นปีที่ 1-6 ที่โรงเรียนจุฬาภรณราชวิทยาลัยจังหวัดเลย จากนั้นเข้าศึกษาในระดับปริญญาตรีในสาขาวิชาวิศวกรรมคอมพิวเตอร์ มหาวิทยาลัยเทคโนโลยี-สุรนารี และสำเร็จการศึกษาเมื่อปี 2550 ภายหลังจากสำเร็จการศึกษาในระดับปริญญาตรี ได้เข้าศึกษาต่อในระดับปริญญาโท สาขาวิชาวิศวกรรมคอมพิวเตอร์ สำนักวิชาวิศวกรรมศาสตร์ มหาวิทยาลัยเทคโนโลยีสุรนารี ในปีการศึกษา 2551

ในระหว่างการศึกษา ได้รับความอนุเคราะห์อย่างดียิ่งจากคณาจารย์ในสาขาวิชาและได้รับความไว้วางใจให้เป็นผู้ช่วยสอนปฏิบัติการวิชา Object – Oriented Technology, Event – Driven Programming, System Analysis and Design และ Software Engineering

ผลงานวิจัย : ได้เสนอบทความเข้าร่วมในงานประชุมวิชาการเสนอผลงานวิจัยระดับบัณฑิตศึกษาแห่งชาติ ครั้งที่ 23 ประจำปี 2554 เรื่อง การหาความแตกต่างของซอร์สโค้ดโดยใช้โครงสร้างไวยากรณ์ต้นไม้