บทที่ 2

ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงแนวคิด ทฤษฎี และงานวิจัยต่าง ๆ ที่เกี่ยวข้อง เพื่อนำเสนอให้เห็นภาพรวมและกรอบของงานวิจัยมากขึ้น ซึ่งแนวคิดในการนำเสนอผลกระทบในการเปลี่ยนแปลงซอฟต์แวร์ที่ออกแบบด้วยโครงสร้างคลาสที่ต่างกัน มีทฤษฎีที่เกี่ยวข้อง คือ แนวคิดเชิงวัตถุ คลาสไดอะแกรม การเปลี่ยนแปลงในซอฟต์แวร์และประสิทธิภาพของซอฟต์แวร์ จากนั้นจะกล่าวถึงงานวิจัยที่เกี่ยวข้อง

2.1 แนวคิดเชิงวัตถุ (Object – Oriented Paradigm)

แนวคิดเชิงวัตถุเป็นแนวคิด ที่พยายามแก้ปัญหาของระบบซอฟต์แวร์ขนาดใหญ่และมีฟังก์ชันการทำงานที่ซับซ้อน โดยใช้วิธีการแก้ปัญหาโดยการแตกปัญหาออกเป็นส่วนย่อย ๆ และเรียกส่วนย่อย ๆ นี้ว่า "ออบเจคท์"

ออบเจคท์ (Object) ถูกนิยามด้วยแอททริบิวต์ (Attribute) และเมธธอด (Method) โดยที่แอททริบิวต์จะแสดงสถานะของออบเจคท์ ส่วนเมธธอดจะเป็นการแสดงการกระทำที่แสดงพฤติกรรมของออบเจคท์ในสถานะต่าง ๆ โดยเมื่อออบเจคท์แต่ละออบเจคท์ถูกสร้างขึ้นจะมีค่าตัวระบุออบเจคท์ (Object Identifier : OID) ซึ่งเป็นค่าประจำที่ทำให้ออบเจคท์แต่ละออบเจคท์มีความเป็นเอกภาพ (Unique) ในระบบ ซึ่งออบเจคท์ในระบบจะมีพฤติกรรมและแอททริบิวต์ของแต่ละออบเจคท์แตกต่างกัน แต่หากออบเจคท์ใด ๆ มีพฤติกรรมและแอททริบิวต์เหมือนกัน จะเรียกว่าเป็นออบเจคท์ชนิดเดียวกัน เนื่องจากโครงสร้างของออบเจคท์จะเหมือนกันทั้ง แอททริบิวต์และเมธธอด จึงมีการนิยามโครงสร้างของออบเจคท์ชนิดเดียวกันในรูปแบบของคลาส (Class) ซึ่งเปรียบเสมือนเป็นแม่แบบ (Template) ของออบเจคท์ และเรียกออบเจคท์ของคลาสว่าเป็นอินสแตนซ์ (Instance) ของคลาสนั้น และคลาสสามารถสืบทอดคุณสมบัติ (Inherit) ของตัวเองถ่ายทอดไปยังคลาสอื่น ๆ ได้ เพื่อทำให้เกิดคลาสใหม่ที่มีลักษณะเพิ่มเติมจากคลาสเดิม ซึ่งคลาสที่ถูกถ่ายทอดคุณสมบัติไปว่าซูเปอร์คลาส (Super Class) และเรียกคลาสที่รับการถ่ายทอดคุณสมบัติมาว่า ซับคลาส (Subclass) โดยการสืบทอดทั่วไปทำให้เกิดคุณสมบัติที่เจาะจงและมีความเป็นทั่วไปน้อยลงกว่าเดิม ดังนั้น หลักการสำคัญของแนวคิดเชิงวัตถุ ประกอบด้วยนิยามคำศัพท์ (ชาลี วรกุลพิพัฒน์ และ เทพฤทธิ์ บัณฑิตวัฒนาวงศ์, 2544) ดังนี้

1. **ออบเจคท์ (Object)** ในแนวคิดเชิงวัตถุ ออบเจคท์ หมายถึง สิ่งของทั้งสิ่งที่จับต้องได้และจับต้องไม่ได้ เช่น บัญชีลูกค้า เมนู หรือเหตุการณ์ต่าง ๆ เป็นต้น
2. **เอ็นแคปซูเลชัน (Encapsulation)** หมายถึง การจัดการกลุ่มของแนวความคิดที่มีความคล้ายคลึงกันเข้าเป็นหน่วยเดียวกันเพื่ออ้างถึงด้วยชื่อเดียวกัน ซึ่งในแนวความคิดเชิงวัตถุ การเอ็นแคปซูเลชัน หมายถึงการรวบรวมแอททริบิวต์และเมธธอดเข้าเป็นหน่วยเดียวกัน เพื่อที่ว่าแอททริบิวต์สามารถถูกเปลี่ยนแปลงได้อย่างเหมาะสมโดยผ่านเมธธอด และจะเรียกผลที่เกิดจากการใช้งานเอ็นแคปซูเลชันว่า การซ่อนข้อมูล (Information Hiding)
3. **คลาส (Class)** หมายถึง แม่แบบ ที่ประกอบไปด้วย ชื่อคลาส แอททริบิวต์ และเมธธอด สำหรับใช้ในการสร้างอินสแตนซ์ของคลาส โดยทุก ๆ คลาสที่ถูกสร้างขึ้นมาจากคลาสเดียวกันจะมีโครงสร้างและพฤติกรรมที่เหมือนกัน กล่าวคือ มีแอททริบิวต์และเมธธอดที่เหมือนกัน หากแต่ค่าในแอททริบิวต์อาจจะแตกต่างกัน
4. **อินสแตนซ์ (Instance)** หมายถึง ออบเจคท์ที่ถูกสร้างขึ้น (Instantiated) จากคลาส
5. **แอททริบิวต์ (Attribute)** หมายถึง คุณสมบัติ (Property) ของออบเจคท์ หรืออาจจะใช้แสดงถึงสถานะ (State) ของออบเจคท์ ณ เวลาใดเวลาหนึ่ง
6. **เมธธอดหรือตัวดำเนินการ (Method / Operation)** หมายถึง ฟังก์ชัน พฤติกรรม (Behavior) หรือบริการที่ออบเจคท์สามารถจะกระทำให้ได้ ทั้งนี้วัตถุประสงค์หลักของเมธธอด เพื่อใช้ในการจัดการแอททริบิวต์
7. **ซิกเนเจอร์ (Signature)** หมายถึง ส่วนที่ประกอบไปด้วย ชื่อของเมธธอด พารามิเตอร์ของเมธธอด และชนิดของข้อมูลที่ถูกส่งคืนจากเมธธอด เช่น boolean SampleMethod (int inputParameter) เป็นต้น จากตัวอย่างอธิบายได้ว่า
   - SampleMethod เป็นชื่อของเมธธอด
   - (int inputParameter) เป็นพารามิเตอร์ของเมธธอด
   - boolean เป็นชนิดของข้อมูลที่ถูกส่งคืนจากเมธธอด
8. **เมสเสจหรือข้อความ (Massage)** หมายถึง ส่วนที่ประกอบไปด้วยชื่อของเมธธอดและค่าของพารามิเตอร์ต่าง ๆ ของเมธธอดดังกล่าว โดยหลักการเชิงวัตถุจะ “ส่งเมสเสจไปยังออบเจคท์” คือ การเรียกใช้งานเมธธอดของออบเจคท์
9. **อินเทอร์เฟซ (Interface)** หมายถึง ชุดของของซิกเนเจอร์ทั้งหมดของคลาสใดคลาสหนึ่ง ซึ่งจะแสดงถึงสิ่งที่ออบเจคท์ของคลาสดังกล่าว สามารถกระทำหรือตอบสนอง

ได้ โดยก่อนที่จะมีการส่งเมสเสจไปยังออบเจคท์ใด ๆ จำเป็นจะต้องทราบถึงอินเทอร์เฟซของออบเจคท์นั้น ๆ ก่อน

10. **การสืบทอดคุณสมบัติ (Inheritance)** หมายถึง วิธีการในการสร้างคลาสใหม่จากคลาสเดิมที่มีอยู่ ทั้งนี้คลาสที่ถูกสร้างขึ้นใหม่จะมีวัตถุประสงค์ในการทำงานที่เฉพาะเจาะจงมากยิ่งขึ้น
11. **การเปลี่ยนรูป (Polymorphism)** หมายถึง การเปลี่ยนรูปของออบเจคท์หนึ่ง ๆ โดยในแนวคิดเชิงวัตถุ จะเป็นการที่แอททริบิวต์ของคลาสใดคลาสหนึ่งสามารถเปลี่ยนรูปแบบไปจากคลาสเดิมได้ นอกจากนี้ยังรวมถึงการที่เมธธอดเดียวกันมีพฤติกรรมการทำงานที่แตกต่างกัน เมื่อถูกใช้กับออบเจคท์ที่เกิดจากคนละคลาสกัน กล่าวคือ ออบเจคท์ที่เกิดจากต่างคลาสกันสามารถมีปฏิกริยาตอบสนองต่อเมธธอดชื่อเดียวกันได้อย่างแตกต่างกัน

2.2 คลาสไดอะแกรม (Class Diagram)

ในการพัฒนาซอฟต์แวร์เชิงวัตถุ ผู้พัฒนาซอฟต์แวร์จะมีการสร้างการใช้งานออบเจคท์คลาส และมีการสร้างความสัมพันธ์ระหว่างคลาสหรือออบเจคท์เหล่านั้น เช่น การสืบทอดคุณสมบัติของคลาส ดังนั้น ในการสร้างแบบจำลองเชิงวัตถุ มีความจำเป็นอย่างยิ่งที่จะต้องสร้างไดอะแกรมที่แสดงถึงองค์ประกอบดังกล่าวทั้งหมดอย่างชัดเจน เรียกไดอะแกรมนี้ว่า คลาสไดอะแกรม (Pressman, 2005)

โดยการพัฒนาซอฟต์แวร์เชิงวัตถุนั้น จะมองทุกอย่างที่สนใจเป็นออบเจคท์ (Object) ดังนั้น คลาส จึงเป็นชนิดของกลุ่มออบเจคท์ โดยจะหาคลาสของออบเจคท์ได้ จะต้องสามารถจัดหมวดหมู่ของออบเจคท์หลาย ๆ ออบเจคท์ได้ และที่สำคัญคือ ถ้าจะสร้างระบบใด ๆ ขึ้นมา การหาคลาสจากออบเจคท์ควรให้ตรงกับเรื่องที่เกี่ยวข้องที่กำลังพิจารณาอยู่ (Problem Domain) (Larman, 2002)

ดังนั้น วัตถุประสงค์ของการสร้างคลาสไดอะแกรม จึงเป็นการแสดงถึงโครงสร้างของระบบที่ประกอบไปด้วยคลาส และความสัมพันธ์ระหว่างคลาสเหล่านั้น ประกอบไปด้วย

คลาส (Class) คือ แบบจำลองของออบเจคท์ โดยในแต่ละคลาสประกอบด้วย 3 ส่วน ได้แก่

1. ชื่อคลาส
2. แอททริบิวต์ ประกอบไปด้วย ขอบเขตของการเข้าถึงแอททริบิวต์ 3 ประเภท ดังนี้
    (1) พับลิค (public) ถูกแสดงด้วยเครื่องหมายบวก (+) ซึ่งทุกคลาสสามารถมองเห็นและเรียกใช้งานแอททริบิวต์ได้

(2) ไพรเวท (private) ถูกแสดงด้วยเครื่องหมายลบ (-) ซึ่งแอททริบิวต์จะถูกมองเห็นภายในคลาสเท่านั้น และสามารถถูกเรียกใช้ได้เฉพาะภายในคลาสเดียวกัน โดยที่ซับคลาสก็ไม่สามารถเรียกใช้หรือมองเห็นแอททริบิวต์ของซูปเปอร์คลาสได้

(3) โพรเทคท์ (protected) ถูกแสดงด้วยเครื่องหมายชาร์ป (#) ซึ่งแอททริบิวต์สามารถถูกเรียกใช้ได้เฉพาะจากคลาสนั้นเองหรือซับคลาสของคลาสเท่านั้น

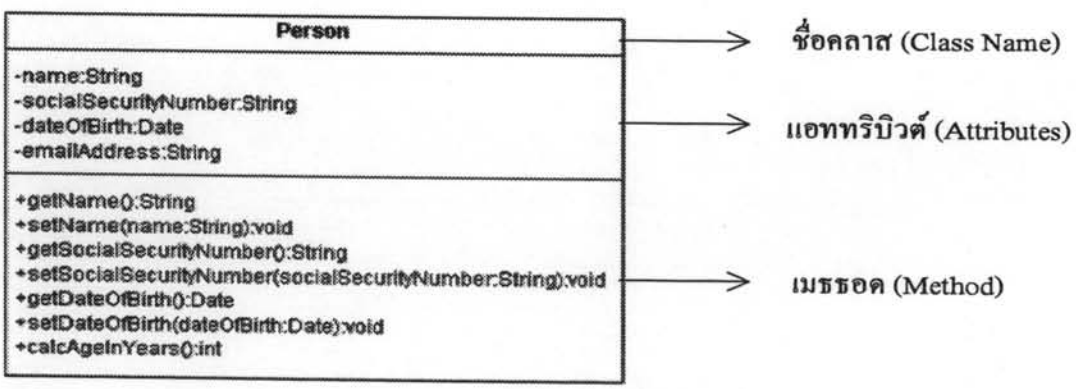
นอกจากนี้แอททริบิวต์ยังประกอบด้วยชื่อของแอททริบิวต์ และประเภทของ แอททริบิวต์ โดยแอททริบิวต์ในคลาสจะแบ่งออกเป็น 2 ประเภท คือ

(1) แอททริบิวต์เดี่ยว (Single – Valued Attributes) เป็นแอททริบิวต์ที่มีชนิดของข้อมูลภายในเป็นชนิดข้อมูลแบบพื้นฐาน เช่น อินทิเจอะ (interger) บูลีน (boolean) ชอร์ท (short) ไบท (byte) เป็นต้น

(2) แอททริบิวต์แบบมัลติแอททริบิวต์ (Multi – Valued Attributes) เป็นแอททริบิวต์ที่มีชนิดของข้อมูลภายในเป็นกลุ่มของชนิดข้อมูลแบบพื้นฐาน เช่น อะเรย์ (array) ลิสท์ (list) เซท (set) เป็นต้น

3. เมธธอด ประกอบไปด้วย ชนิดของการเข้าถึงเมธธอด ซึ่งจะมีชนิดการเข้าถึงเมธธอดเช่นเดียวกับแอททริบิวต์ที่กล่าวมาข้างต้น ชื่อเมธธอด พารามิเตอร์ และประเภทของค่าที่ส่งคืน

โดยสามารถแสดงรูปของคลาสและส่วนประกอบของคลาส ได้ดังรูปที่ 2-1



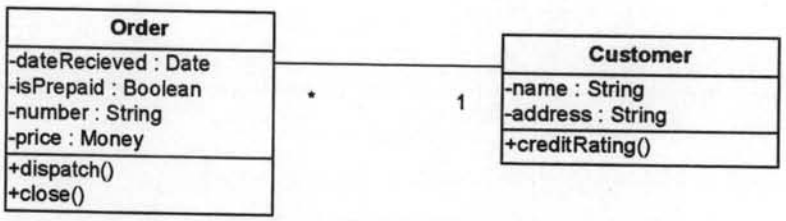
รูปที่ 2-1 แสดงการกำหนดแอททริบิวต์และเมธธอดในคลาส
(ชาลี วรกุลพิพัฒน์ และ เทพฤทธิ์ บัณฑิตวัฒนาวงศ์, 2544)

มัลติพลิซิตี (Multiplicity) ที่แต่ละด้านของความสัมพันธ์ จะมีการแสดงถึงจำนวนที่เป็นไปได้ของออบเจคท์ในแต่ละด้านนั้นต่อออบเจคท์แต่ละตัวในอีกด้านหนึ่ง ตัวอย่างเช่น "1" หมายถึง ความเป็นไปได้ของออบเจคท์ในด้านนั้นมีได้เพียง 1 ตัวเท่านั้น "0..1" หมายถึง ความเป็นไปได้ของออบเจคท์ในด้านนั้นมีได้ 0 หรือ 1 ตัว เป็นต้น

คลอลิไฟเออร์ (Qualifier) เป็นแอททริบิวต์หรือกลุ่มของแอททริบิวต์ที่ค่าของแอททริบิวต์จะใช้อ้างอิงการเชื่อมโยงระหว่างกันทั้ง 2 คลาสที่มีความสัมพันธ์ระหว่างกัน ซึ่งจะเป็นแอททริบิวต์ที่แสดงบนเส้นความสัมพันธ์ระหว่างคลาส

ความสัมพันธ์ระหว่างคลาส (Relationship) เป็นการอธิบายถึง ความสัมพันธ์ระหว่างสิ่งของต่าง ๆ โดยที่ในแต่ละรูปแบบของความสัมพันธ์จะมีการสร้างสัญลักษณ์ในการสื่อความหมายที่แตกต่างกันของแต่ละความสัมพันธ์ จะประกอบด้วย

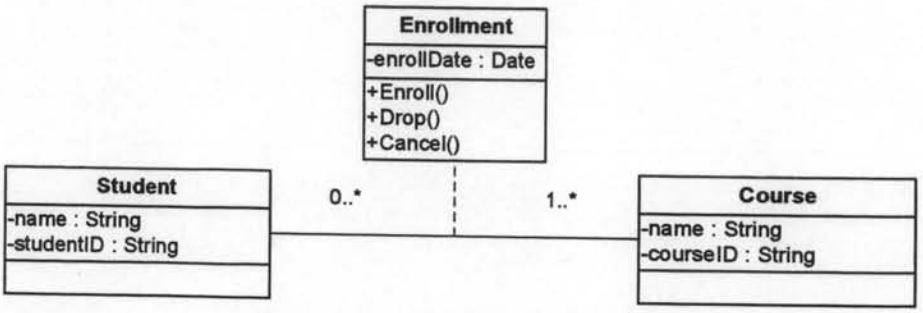
1. **ความสัมพันธ์แบบแอสโซซิเอชัน (Association Relationship)** เป็นความสัมพันธ์ที่อธิบายถึงการเชื่อมต่อระหว่างออบเจคท์ของคลาสหนึ่งกับออบเจคท์ของอีกคลาสหนึ่งที่สามารถโต้ตอบ (Interact) กันได้



รูปที่ 2-2 แสดงความสัมพันธ์แบบแอสโซซิเอชัน (Larman, 2002)

จากรูปที่ 2-2 จะเห็นได้ว่าเป็นการแสดงความสัมพันธ์ระหว่างคลาส Customer และคลาส Order ซึ่งเป็นความสัมพันธ์ของความเกี่ยวข้องกันระหว่างลูกค้าและสินค้า เนื่องจากลูกค้าหนึ่งคนสามารถสั่งซื้อสินค้าได้หลายชิ้น และสินค้าหลายชิ้นเป็นของลูกค้าหนึ่งคน

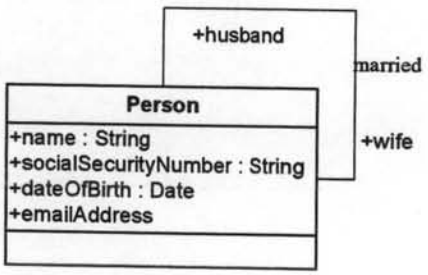
2. **ความสัมพันธ์แบบแอสโซซิเอชันคลาส (Association Class Relationship)** เป็นความสัมพันธ์แบบหนึ่ง ที่เส้นเชื่อมความสัมพันธ์ระหว่างคลาส 2 คลาส จะมีคลาสเป็นของตัวเอง เพิ่มขึ้นมาในเส้นความสัมพันธ์นั้น ๆ



รูปที่ 2-3 แสดงความสัมพันธ์แบบแอสโซซิเอชันคลาส (โอภาส เอี่ยมสิริวงศ์, 2545)

จากรูปที่ 2-3 จะเห็นได้ว่าเป็นการแสดงความสัมพันธ์ระหว่างนิสิต (Student) และรายวิชาเรียน (Course) ซึ่งนิสิตมีความสามารถลงทะเบียน (Enrollment) เรียนตามรายวิชา

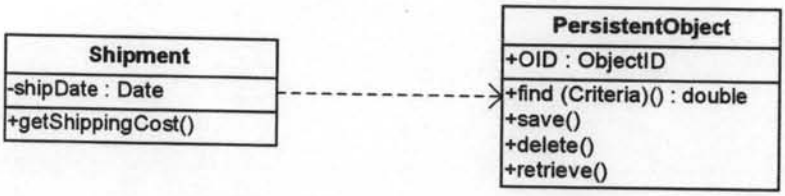
3. **ความสัมพันธ์แบบรีเคอร์ซีพแอสโซซิเอชัน (Recursive Association Relationship)** เป็นความสัมพันธ์ในอีกรูปแบบหนึ่งของความสัมพันธ์แบบแอสโซซิเอชัน ที่อยู่ในรูปแบบของคลาสหนึ่งคลาสมีความสัมพันธ์กับตัวของคลาสมันเอง



รูปที่ 2-4 แสดงความสัมพันธ์แบบรีเคอร์ซีพแอสโซซิเอชัน
(ชาลี วรกุลพิพัฒน์ และ เทพฤทธิ์ บัณฑิตวัฒนาวงศ์, 2544)

จากรูปที่ 2-4 จะเห็นได้ว่าเป็นการแสดงความสัมพันธ์ที่ภรรยาแต่งงานกับสามี ซึ่งทั้งสามีและภรรยาเป็นบทบาทของบุคคลที่แต่งงานกันแล้ว ภายใต้ความสัมพันธ์ชื่อ "married" เท่านั้น กล่าวโดยสรุปคือ บทบาทที่กำหนดไว้จะหมายถึงบทบาทภายใต้รูปแบบของความสัมพันธ์นั้น ๆ เท่านั้น

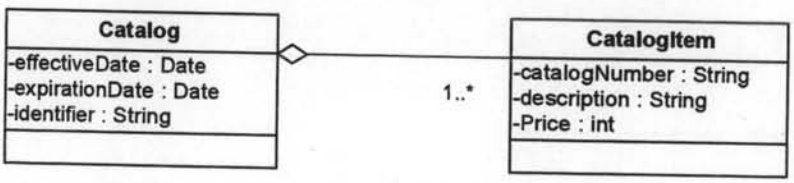
4. **ความสัมพันธ์แบบดีเพนเดนซี (Dependency Relationship)** เป็นความสัมพันธ์ที่คลาสหนึ่งมีคุณสมบัติและพฤติกรรมขึ้นอยู่กับอีกคลาสหนึ่ง เช่น คลาส A มีความสัมพันธ์แบบขึ้นแก่กันกับคลาส B นั่นหมายถึง เมื่อมีการเปลี่ยนแปลงค่าในคลาส A จะทำให้คลาส B มีการเปลี่ยนแปลงค่าตามไปด้วย



รูปที่ 2-5 แสดงความสัมพันธ์แบบดีเพนเดนซี
(ชาลี วรกุลพิพัฒน์ และ เทพฤทธิ์ บัณฑิตวัฒนาวงศ์, 2544)

จากรูปที่ 2-5 จะเห็นได้ว่าความสัมพันธ์แบบดีเพนเดนซี เป็นความสัมพันธ์แบบพึ่งพิง เมื่อมีการเปลี่ยนแปลงที่ตัวข้อมูล (แอททริบิวต์) หรือการเปลี่ยนแปลงการดำเนินการ (เมธธอด) เกิดขึ้นกับคลาสที่ถูกพึ่งพิง ได้แก่คลาส PersistentObject จะส่งผลกระทบต่อคลาสที่พึ่งพิง ได้แก่ คลาส Shipment เช่น การเลื่อนกำหนดส่งสินค้า ต้นทุนการส่งสินค้าเปลี่ยนแปลง เป็นต้น

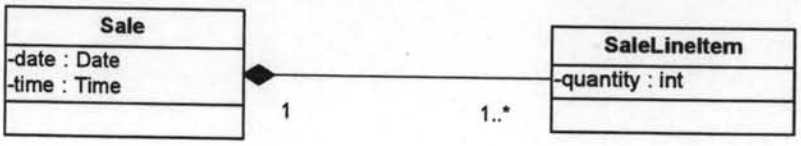
5. **ความสัมพันธ์แบบแอกกริเกชัน (Aggregation Relationship)** เป็นความสัมพันธ์ในลักษณะ "ประกอบด้วย (Has - a)" ซึ่งจะมีคลาสที่แสดงถึงสิ่งของที่ใหญ่กว่าที่ประกอบไปด้วยสิ่งของที่เล็กกว่า ความสัมพันธ์ชนิดนี้เป็นกรณีพิเศษของความสัมพันธ์แบบแอสโซซิเอชัน โดยสามารถแสดงได้ด้วยความสัมพันธ์แบบแอสโซซิเอชันที่มีปลายด้านที่เป็นสิ่งของที่ใหญ่กว่าเป็นรูปสี่เหลี่ยมขนมเปียกปูนโปร่ง



รูปที่ 2-6 แสดงความสัมพันธ์แบบแอกกริเกชัน (Larman, 2002)

จากรูปที่ 2-6 จะเห็นได้ว่าเป็นความสัมพันธ์ที่แคตตาล็อกสินค้า (Catalog) ประกอบไปด้วยรายการสินค้าย่อย ๆ (CatalogItem)

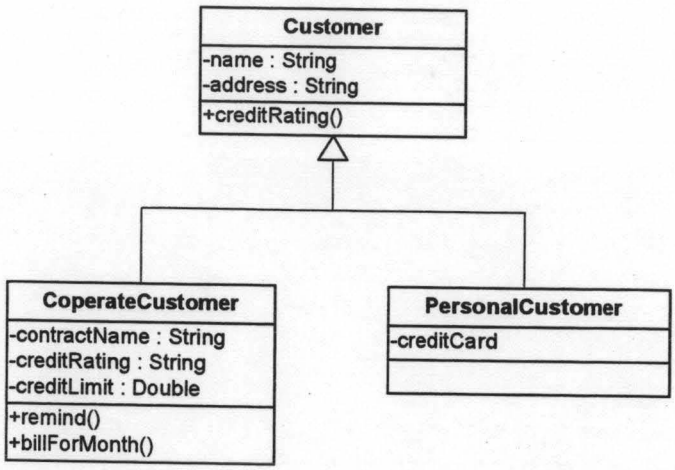
6. **ความสัมพันธ์แบบคอมโพซิชัน (Composition Relationship)** เป็นความสัมพันธ์รูปแบบหนึ่งของความสัมพันธ์แบบแอกกริเกชัน แต่จะแสดงถึงลักษณะของความเป็นเจ้าของที่ชัดเจนยิ่งขึ้น ซึ่งในความสัมพันธ์แบบคอมโพซิชัน ออบเจคท์ที่ใหญ่กว่าจะมีหน้าที่ในการจัดการเกี่ยวกับการสร้างและการทำลายออบเจคท์ที่เล็กกว่า โดยสามารถแสดงได้ด้วยความสัมพันธ์แบบแอสโซซิเอชันที่มีปลายด้านที่เป็นสิ่งของที่ใหญ่กว่าเป็นรูปสี่เหลี่ยมขนมเปียกปูนทึบ



รูปที่ 2-7 แสดงความสัมพันธ์แบบคอมโพซิชัน (Larman, 2002)

จากรูปที่ 2-7 จะเห็นได้ว่าเป็นความสัมพันธ์ออบเจคท์คลาส Sale ประกอบด้วยสมาชิกของออบเจคท์คลาส SaleLineItem (คลาส SaleLineItem จะถูกพิจารณาเป็นส่วนประกอบหนึ่งของคลาส Sale) โดยที่ คลาส Sale จะมีหน้าที่ในการจัดการการเพิ่ม การลบ การทำทรานแซคชันแต่ละรายการสินค้าในคลาส SaleLineItem ได้

7. **ความสัมพันธ์แบบคลอลิไฟล์แอสโซซิเอชัน (Qualified Association Relationship)** เป็นความสัมพันธ์รูปแบบหนึ่งที่มีแอททริบิวต์หรือกลุ่มของแอททริบิวต์ที่มีค่าของแอททริบิวต์ดังกล่าวที่ใช้อ้างอิงถึงการเชื่อมโยงระหว่างกันระหว่างคลาส 2 คลาสที่มีความสัมพันธ์กัน ซึ่งเป็นแอททริบิวต์ของเส้นความสัมพันธ์แบบแอสโซซิเอชัน คอมโพซิชัน และแอกกริเกชัน
8. **ความสัมพันธ์แบบเจเนอรัลไลเซชัน (Generalization Relationship)** เป็นความสัมพันธ์ที่อธิบายถึงความสัมพันธ์ระหว่างซูปเปอร์คลาสและซับคลาส ที่ทำให้เกิดกลไกการสืบทอดคุณสมบัติขึ้นมาได้ ซึ่งอาจจะมองว่าเป็นความสัมพันธ์แบบ "เป็นชนิดหนึ่งของ (Is – a – kind – of)"



รูปที่ 2-8 แสดงความสัมพันธ์แบบเจเนอรัลไลเซชัน (Larman, 2002)

จากรูปที่ 2-8 จะเห็นได้ว่าเป็นความสัมพันธ์ระหว่างคลาสในการสืบทอดคุณสมบัติ จากคลาส Customer ซึ่งเป็นซูปเปอร์คลาส ไปยังซับคลาส PersonalCustomer และคลาส CoperateCustomer ซึ่งจะหมายถึงคลาส PersonalCustomer และคลาส CoperateCustomer ยังมีคุณสมบัติบางส่วนที่สืบทอดคุณสมบัติมาจากคลาส Customer แต่จะมีวัตถุประสงค์ของการทำงานที่เฉพาะเจาะจงมากกว่า

9. **ความสัมพันธ์แบบเรียลไลเซชัน (Realization Relationship)** เป็นความสัมพันธ์ลักษณะนี้ในการระบุความสัมพันธ์ระหว่างอินเทอร์เฟซกับคลาส โดยอินเทอร์เฟซจะระบุถึง เมธธอดที่เสนอและคลาสจะใช้บริการตามที่ระบุไว้ในอินเทอร์เฟซเท่านั้น

2.3 การเปลี่ยนแปลงโครงสร้างคลาสให้เป็นโครงสร้างคลาสในความสัมพันธ์แบบแอสโซซิเอชันและโครงสร้างคลาสในความสัมพันธ์เจเนอรัลไลเซชัน (Association Relationship and Generalization Relationship)

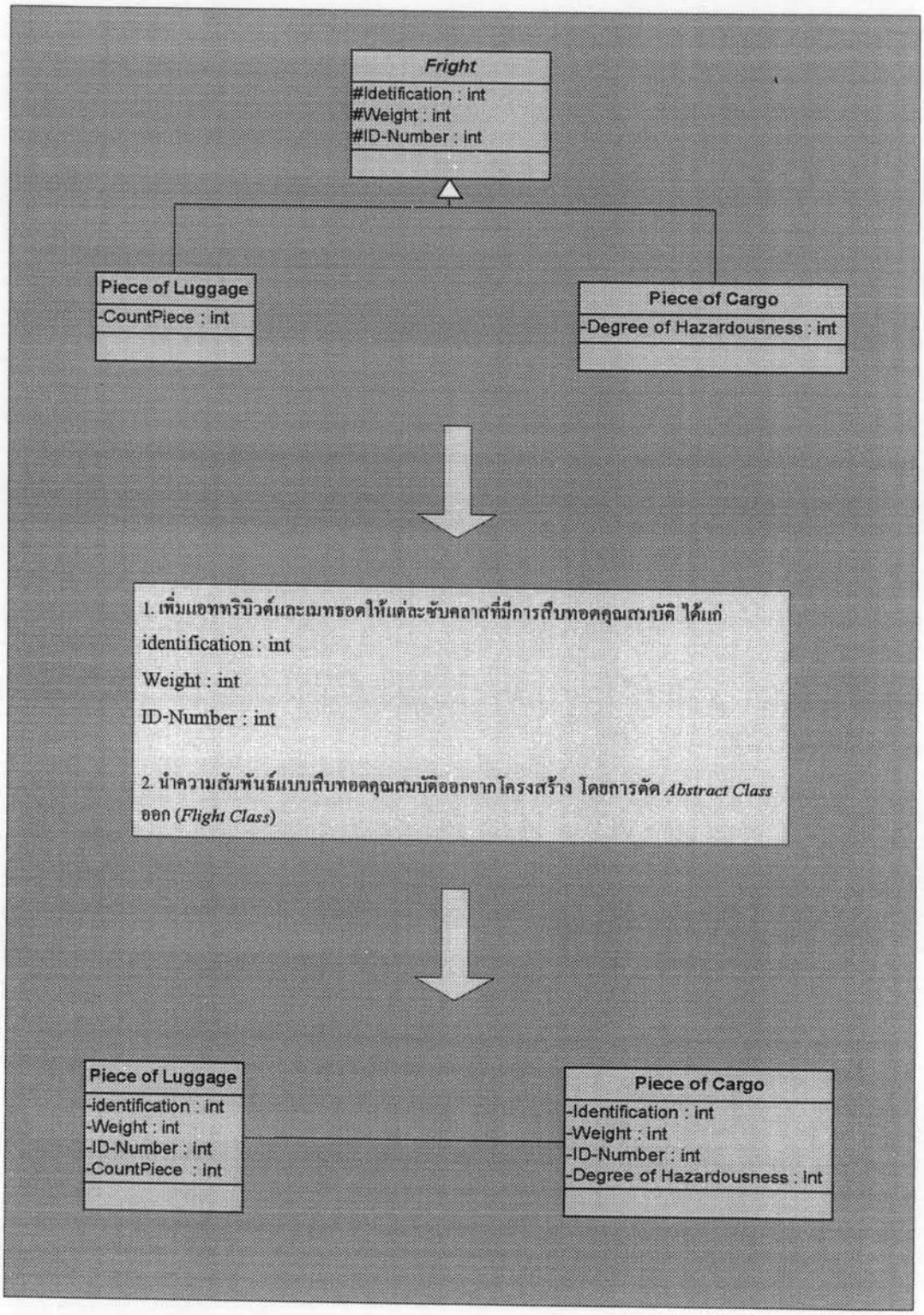
การเปลี่ยนแปลงโครงสร้างคลาสมีด้วยกัน 2 แบบ คือ การเปลี่ยนแปลงโครงสร้างคลาสในความสัมพันธ์แบบเจเนอรัลไลเซชันให้เป็นโครงสร้างคลาสในความสัมพันธ์แบบแอสโซซิเอชัน และการเปลี่ยนแปลงโครงสร้างคลาสในความสัมพันธ์แบบแอสโซซิเอชัน ให้เป็นโครงสร้างคลาสในความสัมพันธ์แบบเจเนอรัลไลเซชัน โดยมีหลักการเปลี่ยนแปลงโครงสร้างคลาส ดังนี้

2.3.1 หลักการเปลี่ยนโครงสร้างคลาสในความสัมพันธ์แบบเจเนอรัลไลเซชัน ให้เป็นโครงสร้างคลาสในความสัมพันธ์แบบแอสโซซิเอชัน

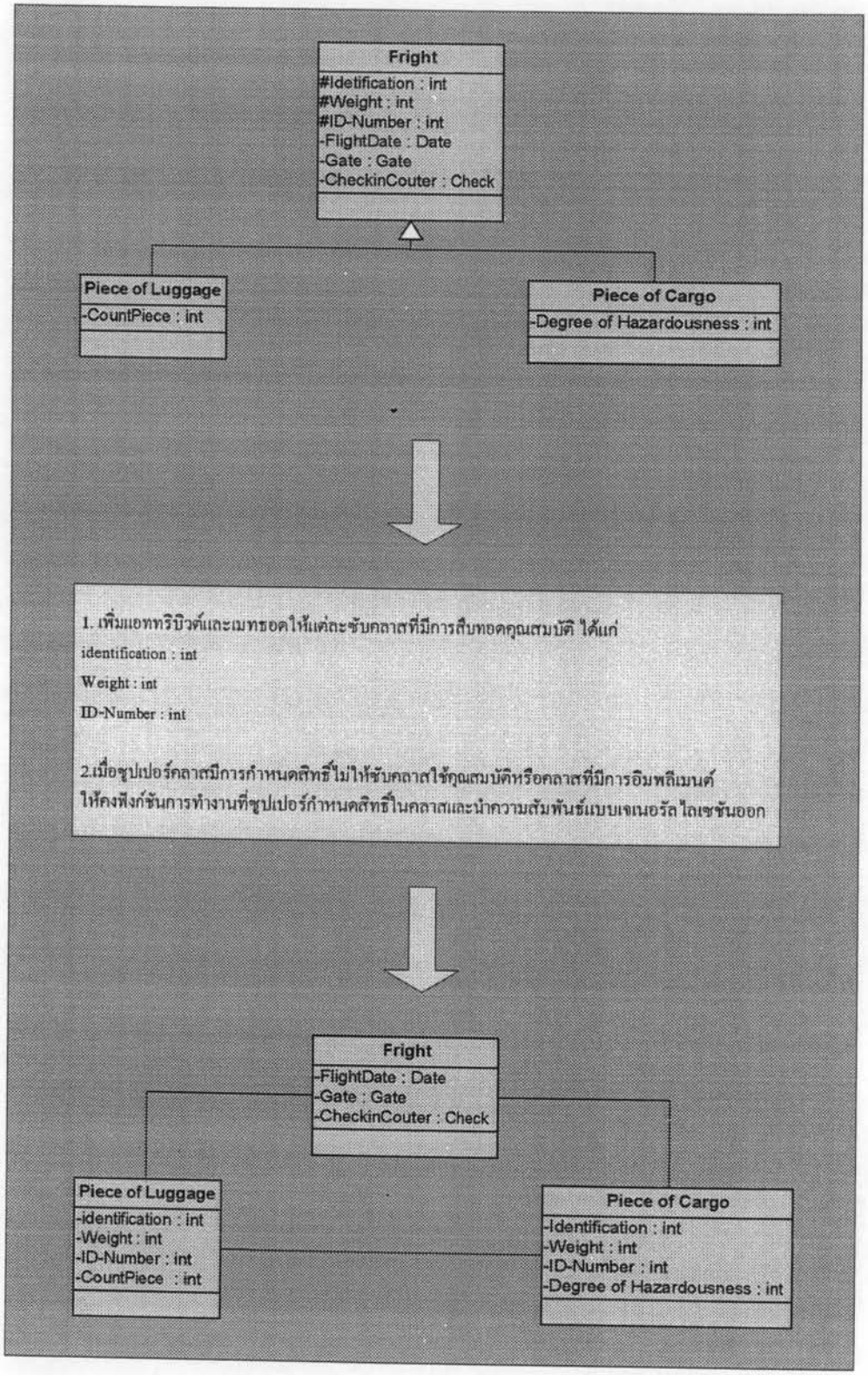
Daly และคณะ (1995) สร้างคลาสที่ไม่มีระดับชั้นในการสืบทอดคุณสมบัติโดยการนำเอาความสัมพันธ์ระหว่างคลาสที่มีการสืบทอดคุณสมบัติกันออก และเพิ่มจำนวนสมาชิกของแอททริบิวต์และเมธธอดที่สัมพันธ์กันลงในแต่ละคลาสก่อนที่จะมีการสืบทอดคุณสมบัติ โดยทุก ๆ แอบสแทรคท์คลาส (Abstract Classes) ในโครงสร้างคลาสที่มีระดับชั้นการสืบทอดคุณสมบัติ จะถูกนำออกทั้งหมดเมื่อเปลี่ยนมาเป็นโครงสร้างที่ไม่มีระดับชั้นในการสืบทอดคุณสมบัติ

Cartwright และ Shepperd (1998) ศึกษาผลกระทบของการสืบทอดคุณสมบัติภายในซอฟต์แวร์เชิงวัตถุ โดยศึกษาเปรียบเทียบระหว่างคลาสที่มีการสืบทอดคุณสมบัติและคลาสที่ไม่มีการสืบทอดคุณสมบัติ ซึ่งงานวิจัยสร้างคลาสที่ไม่มีการสืบทอดคุณสมบัติโดยการเติมแอททริบิวต์และเมธธอดที่ซูปเปอร์คลาสถ่ายทอดคุณสมบัติให้แต่ละซับคลาสที่สืบทอดคุณสมบัติโดยตรง และนำความสัมพันธ์แบบสืบทอดคุณสมบัติออกจากโครงสร้างคลาส

Unger และ Prechelt (1998) ศึกษาผลกระทบจากการสืบทอดคุณสมบัติว่ามีความสามารถพัฒนาผลิตภัณฑ์ซอฟต์แวร์ (Productivity) และลดระยะเวลาในการพัฒนาซอฟต์แวร์ (Development Time) พบว่าจุดประสงค์เฉพาะของระดับชั้นของความลึกของการสืบทอดคุณสมบัติทำให้ยากในการทำความเข้าใจซอฟต์แวร์ แต่ผลกระทบขึ้นอยู่กับปัจจัยอื่น ๆ เช่น ความซับซ้อนของซอฟต์แวร์และประเภทของการบำรุงรักษาซอฟต์แวร์มากกว่าระดับความลึกของการสืบทอดคุณสมบัติ โดยงานวิจัยประเมินผลจากโปรแกรมที่มีระดับความลึกในการสืบทอดคุณสมบัติและโปรแกรมที่ไม่มีระดับความลึกในการสืบทอดคุณสมบัติ โดยโปรแกรมที่ไม่มีระดับความลึกในการสืบทอดคุณสมบัติสร้างได้ ดังนี้ ถ้าหากซูปเปอร์คลาสเป็นแอบสแทรคท์คลาส (Abstract Classes) ทุก ๆ แอบสแทรคท์คลาสจะถูกนำออกจากโครงสร้างคลาสและฟังก์ชันการทำงาน (แอททริบิวต์และเมธธอด) ของแอบสแทรคท์คลาสจะถูกเพิ่มเข้าที่คลาสที่เป็นซับคลาส (Sub Classes) ฟังก์ชันการทำงานที่ไม่มีการใช้งานและถูกกำหนดไว้ในแอบสแทรคท์คลาสจะถูกนำออกทั้งหมด และถ้าหากซูปเปอร์คลาสเป็นคลาสที่มีการอิมพลีเมนต์ (Implement Classes) ให้คงฟังก์ชันการทำงานที่ซูปเปอร์คลาสกำหนดสิทธิ์ไว้ให้เป็นการทำงานของตัวเองไว้ในคลาส และนำความสัมพันธ์แบบสืบทอดคุณสมบัติออกจากโครงสร้างคลาส แสดงได้ดังรูปที่ 2-9 และรูปที่ 2-10



รูปที่ 2-9 แสดงการเปลี่ยนโครงสร้างคลาสในความสัมพันธ์แบบเจเนอรัลไลเซชันให้เป็นโครงสร้างคลาสในความสัมพันธ์แบบแอสโซซิเอชัน (ซูปเปอร์คลาสเป็นแอบสแทรคท์คลาส)



รูปที่ 2-10 แสดงการเปลี่ยนโครงสร้างคลาสในความสัมพันธ์แบบเจเนอรัลไลเซชันให้เป็นโครงสร้างคลาสในความสัมพันธ์แบบแอสโซซิเอชัน (ซูปเปอร์คลาสเป็นอิมพลีเมนต์คลาส)

Harison, Counsell และ Nithi (2000) ต้องการศึกษาความสามารถในการเปลี่ยนแปลงซอฟต์แวร์ (Modifiability) และความสามารถในการทำความเข้าใจซอฟต์แวร์ (Understandability) ของการออกแบบซอฟต์แวร์โดยใช้หลักการเชิงวัตถุ โดยเปรียบเทียบจากระดับความลึกของการสืบทอดคุณสมบัติในหลายระดับความลึกและไม่มีระดับความลึกในการสืบทอดคุณสมบัติ ซึ่งคลาสที่ไม่มีระดับความลึกในการสืบทอดคุณสมบัติสร้างโดยการนำข้อมูลหรือแอททริบิวต์และพฤติกรรมหรือเมธธอดจากซูปเปอร์คลาสเพิ่มให้แต่ละซับคลาสที่มีการสืบทอดคุณสมบัติโดยตรง และนำเอาความสัมพันธ์ที่มีการสืบทอดคุณสมบัติออก

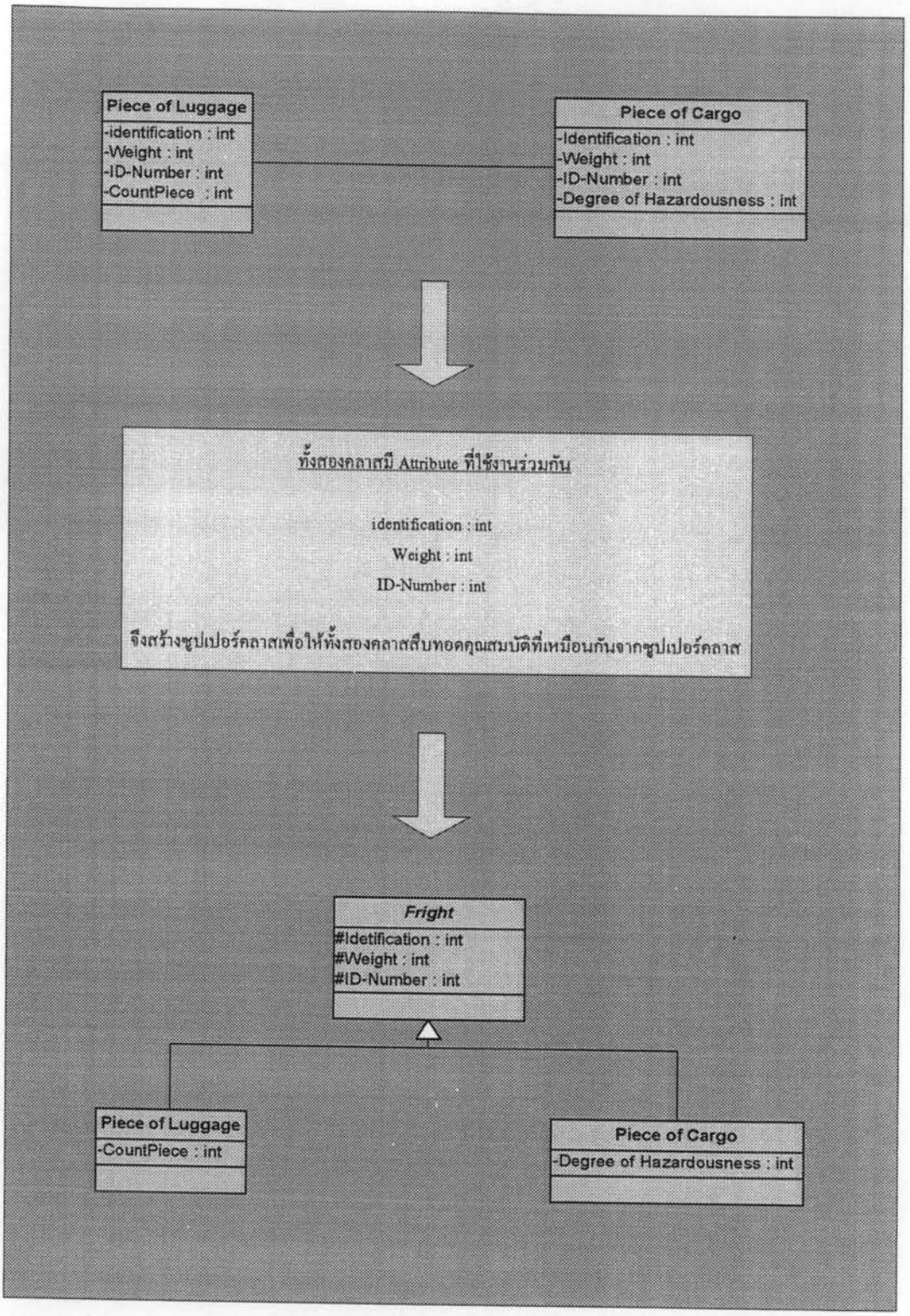
นอกจากนี้งานวิจัยของ Perchelt และคณะ (2003) ศึกษาระดับความลึกของการสืบทอดคุณสมบัติว่าเป็นปัจจัยที่มีผลกับต้นทุนในการบำรุงรักษาซอฟต์แวร์ งานวิจัยของ Lakos และ Lewis (2000) ศึกษาพฤติกรรมการสืบทอดคุณสมบัติ ว่ามีประโยชน์ในการใช้โค้ดร่วมกันหรือการนำกลับมาใช้ใหม่ (Reusability) มากกว่าการออกแบบคลาสในลักษณะอื่น ๆ โดยทำการทดลองเปรียบเทียบระหว่างคลาสที่มีการสืบทอดคุณสมบัติและคลาสที่ไม่มีการสืบทอดคุณสมบัติ โดยสร้างคลาสที่ไม่มีการสืบทอดคุณสมบัติเช่นเดียวกับงานวิจัยของ Cartwright และ Shepperd (1998)

2.3.2 หลักการเปลี่ยนโครงสร้างคลาสในความสัมพันธ์แบบแอสโซซิเอชันให้เป็นโครงสร้างคลาสในความสัมพันธ์แบบเจเนอรัลไลเซชัน

โครงสร้างคลาสในความสัมพันธ์แบบเจเนอรัลไลเซชัน คือ คลาสประเภทที่มีการสืบทอดคุณสมบัติเป็นระดับชั้น (Generalization Class Hierarchy) โดยที่ออบเจคท์คลาสที่มีฟังก์ชันการทำงานทั่วไป (General) ถูกจัดการให้อยู่ระดับบนสุดของระดับชั้นและคลาสที่มีการระบุคุณสมบัติพิเศษเพิ่มเติมจะมีการสืบทอดคุณสมบัติแอททริบิวต์และเมธธอดจากระดับบนสุดของระดับชั้นหรือที่เรียกว่าซูปเปอร์คลาส (Super Class) โดยระดับชั้นที่สืบทอดคุณสมบัติลึกลงมาหรือที่เรียกว่าซับคลาส (Subclass) สามารถระบุแอททริบิวต์และเมธธอดเพิ่มขึ้นได้ (Sommerville, 2001) ซึ่งได้แก่การเคลื่อนย้ายข้อมูลหรือ แอททริบิวต์และพฤติกรรมหรือเมธธอด ที่มีลักษณะการทำงานที่มีความเหมือนกันและมีการใช้งานร่วมกันมากำหนดเป็นความสัมพันธ์แบบซูปเปอร์คลาสและซับคลาส (Larman, 2002)

ดังนั้น คลาสประเภทที่มีการสืบทอดคุณสมบัติสร้างได้โดย การกำหนดคลาสใหม่ให้เป็น "คลาสที่มีคุณลักษณะเดียวกัน" กับคลาสเดิมที่มีอยู่ ด้วยวิธีการนี้สามารถใช้คุณสมบัติต่าง ๆ ของคลาสเดิม เช่น แอททริบิวต์ เมธธอด หรืออื่น ๆ ทั้งหมดของคลาสเดิมที่มีอยู่ได้ โดยไม่ต้องมีการแก้ไขใด ๆ แต่เจ้าของคลาสมีสิทธิ์ที่จะกำหนดไม่ให้ใช้คุณสมบัติใด ๆ ที่ไม่ต้องการให้คลาสอื่นใช้ได้เช่นกัน รวมทั้งคลาสใหม่สามารถกำหนดคุณสมบัติใหม่ ๆ เพิ่มเติมจากที่คลาสเดิมไม่มีได้อีก

ด้วย (กิตติ ภักดีวัฒนะกุล, 2545) หรือทำได้โดยการพิจารณาแยกแอททริบิวต์และเมธธอดที่มีร่วมกันจากหลาย ๆ คลาสในความสัมพันธ์แบบแอสโซซิเอชัน และนำคุณสมบัติที่มีร่วมกันมากำหนดรวมกันสร้างเป็นคลาสเจเนอรัลไลเซชันซูปเปอร์คลาส (Generalization Super Class) ขึ้น โดยลักษณะที่มีร่วมกันนั้นหมายรวมถึง แอททริบิวต์ เมธธอด และความสัมพันธ์ภายในคลาส จากนั้นนำคุณลักษณะที่ใช้ร่วมกันมาประกาศที่ซูปเปอร์คลาสที่สร้างขึ้น แล้วประยุกต์ซับคลาส (Subclass) ให้มีการเรียกใช้งานซูปเปอร์คลาส โดยการสืบทอดคุณสมบัติของแอททริบิวต์ และเมธธอดที่มีคุณสมบัติร่วมกัน อย่างไรก็ตามขอบเขตความสัมพันธ์ระหว่างซูปเปอร์คลาสและซับคลาส มีความสัมพันธ์กันดังนี้ ทุก ๆ สเททเมินท์ (Statement) ที่ถูกสร้างขึ้นในซูปเปอร์คลาสจะถูกนำไปใช้กับทุก ๆ ซับคลาส แล้วเรียกว่าซับคลาสมีการสืบทอดคุณสมบัติแอททริบิวต์ เมธธอด และความสัมพันธ์มาจากซูปเปอร์คลาส (Grassle and Baumann, 2005) ดังรูปที่ 2-11



รูปที่ 2-11 แสดงการเปลี่ยนโครงสร้างคลาสในความสัมพันธ์แบบแอสโซซิเอชันให้เป็นโครงสร้างคลาสในความสัมพันธ์แบบเจเนอรัลไลเซชัน

2.4 ประเภทของเมสเสจ (Message)

แนวคิดของซอฟต์แวร์เชิงวัตถุ คลาสหรือออบเจคท์เป็นหน่วยพื้นฐานในการพัฒนาซอฟต์แวร์ และการทำงานของซอฟต์แวร์จะเกิดขึ้นจากการทำงานร่วมกันโดยการส่งข้อความหรือเมสเสจ (Message Calling) ระหว่างคลาสทั้งหมดที่เป็นองค์ประกอบของซอฟต์แวร์ (ชาลี วรกุลพิพัฒน์ และ เทพฤทธิ์ บัณฑิตวัฒนาวงศ์, 2544) เมสเสจที่มีการรับส่ง และเรียกใช้งานระหว่างคลาสสามารถแบ่งออกได้เป็น 3 ประเภทคือ เมสเสจที่เป็นแอททริบิวต์ เมสเสจที่เป็นอาร์เรย์ และเมสเสจที่เป็นเมธธอด ซึ่งแสดงเป็นแผนภาพต้นไม้ในรูปที่ 2-12 และมีรายละเอียดการเรียกใช้งานเมสเสจดังนี้

2.4.1 เมสเสจที่เป็นแอททริบิวต์

เมสเสจที่เป็นแอททริบิวต์ สามารถเรียกต่อไปได้เรื่อยๆ โดยจะมีเครื่องหมาย . เป็นตัวแบ่งโทเคนในการเรียกใช้งานแอททริบิวต์แต่ละตัว สำหรับการเรียกใช้งานเมสเสจที่เป็นแอททริบิวต์สามารถแบ่งเป็น 2 ส่วน คือ

1 ส่วนโทเคนแรก ในโทเคนแรกสามารถแบ่งออกเป็นได้ 4 กรณี คือ

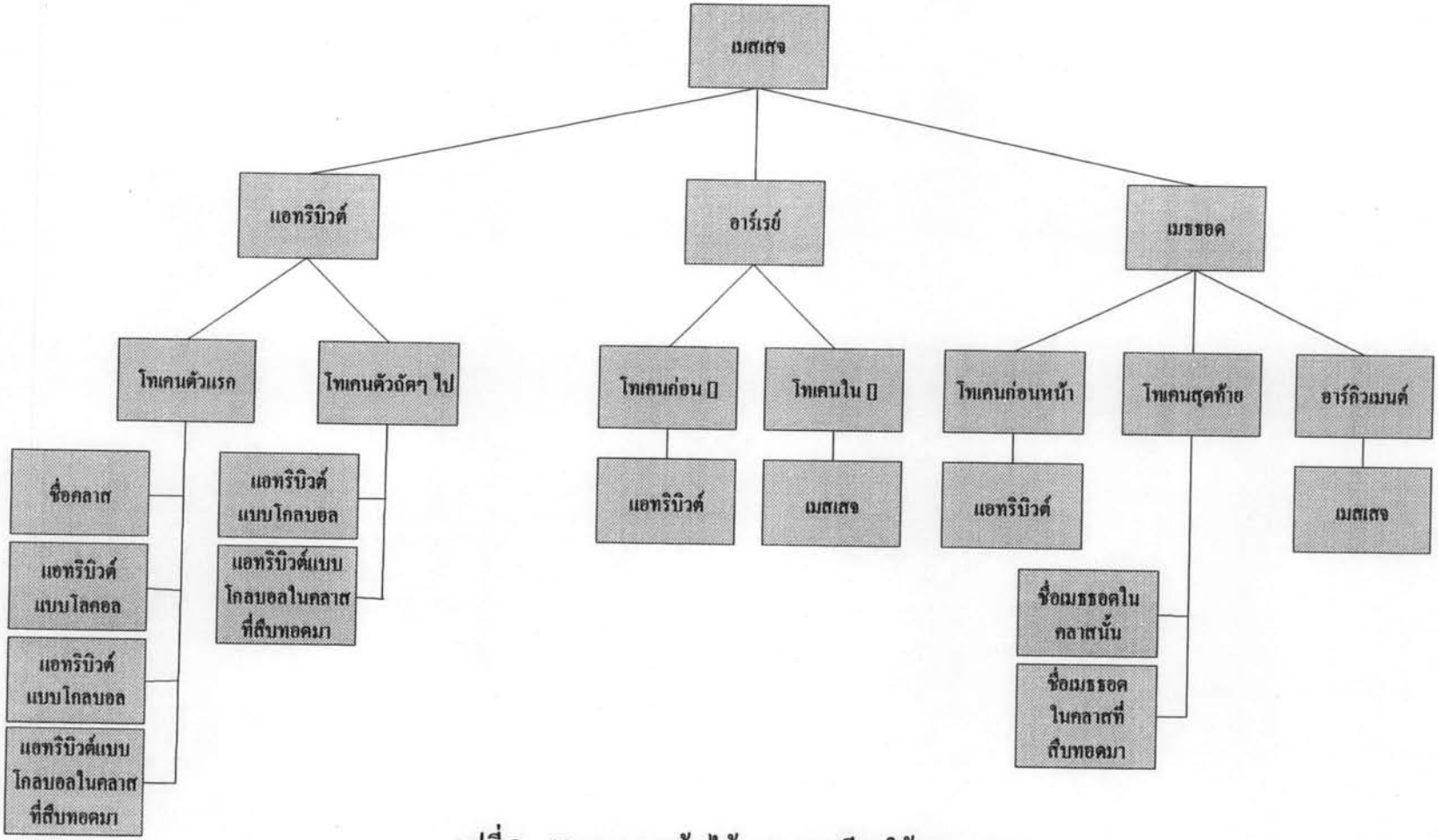
- ชื่อคลาส เช่น การเรียกใช้ B ในเมสเสจ B.b1 ในตารางที่ 2 - 1 จะเป็นการเรียกใช้คลาส B โดยตรง

ตารางที่ 2 - 1 ตัวอย่างการเรียกใช้งานแอททริบิวต์ที่เป็นชื่อคลาส

```
class A {
    public int a1;
    public A(){
        a1 = B.b1;
    }
}
```

```
class B {
    public static int b1;
}
```

- แอททริบิวต์ที่เป็นแบบโลคอล เป็นแอททริบิวต์ที่ประกาศอยู่ในเมธธอดที่เรียกใช้งาน เช่น การเรียกใช้ b ในเมสเสจ b.b1 ในตารางที่ 2 - 2 ซึ่งเป็นแอททริบิวต์ที่ประกาศใช้ภายในเมธธอด A ()



รูปที่ 2 – 12 แผนภาพต้นไม้แสดงการเรียกใช้งานเมสเสจ

ตารางที่ 2 - 2 ตัวอย่างการเรียกใช้งานแอททริบิวต์ที่เป็นแอททริบิวต์แบบโลคอล

```
class A {
    public int a1;
    public A(){
        B b = new B();
        a1 = b.b1;
    }
}
```

```
class B {
    public int b1;
}
```

- แอททริบิวต์ที่เป็นแบบโกลบอล เป็นแอททริบิวต์ที่ประกาศเป็นโกลบอลของคลาสที่เรียกใช้งาน เช่น การเรียกใช้งาน b ในเมสเสจ b.b1 ในตารางที่ 2 - 3 จะเรียกใช้งานแอททริบิวต์ b ที่ประกาศเป็นโกลบอลในคลาส A

ตารางที่ 2 - 3 ตัวอย่างการเรียกใช้งานแอททริบิวต์ที่เป็นแอททริบิวต์แบบโกลบอล

```
class A {
    public int a1;
    B b ;
    public A(){
        b = new B();
        a1 = b.b1;
    }
}
```

```
class B {
    public int b1;
}
```

- แอททริบิวต์ที่เป็นแบบโกลบอลในคลาสที่มีการสืบทอด จะเป็นการเรียกใช้แอททริบิวต์ที่ประกาศเป็นแบบโกลบอลในคลาส ซึ่งเป็นคลาสที่มีการสืบทอดคุณสมบัติมา เช่น การเรียกใช้แอททริบิวต์ b1 ในตารางที่ 2 - 4 ซึ่งจะเป็นการเรียกใช้งานแอททริบิวต์ b1 ในคลาส B ซึ่งคลาส A สืบทอดคุณสมบัติมา

ตารางที่ 2 - 4 ตัวอย่างการเรียกใช้แอททริบิวต์ที่เป็นแอททริบิวต์แบบโกลบอลที่มีการสืบทอด

```
class A extends B{
    public int a1;
    public A(){
        a1 = b1;
    }
}
```

```
class B {
    public int b1;
}
```

2 ส่วนโทเคนถัด ๆ ไป สำหรับการเรียกใช้งานแอททริบิวต์ในโทเคนถัด ๆ ไป สามารถแบ่งออกเป็น 2 กรณีคือ

- แอททริบิวต์ที่เป็นแบบโกลบอล จะเป็นการเรียกใช้งานแอททริบิวต์ที่ประกาศเป็นโกลบอลที่สามารถเข้าถึงได้จากโทเคนหรือแอททริบิวต์ตัวก่อน เช่น การเรียกใช้งาน b1 ในเมสเสจ b.b1 ในตารางที่ 2 - 5 จะเป็นการเรียกใช้งานแอททริบิวต์ b1 โดยเรียกผ่านแอททริบิวต์ b ในคลาส A

ตารางที่ 2 - 5 ตัวอย่างเรียกใช้แอททริบิวต์ที่เป็นแอททริบิวต์แบบโกลบอลที่เข้าถึงจากโทเคนก่อนหน้า

```
class A {
    public int a1;
    B b;
    public A(){
        b = new B();
        a1 = b.b1;
    }
}
```

```
class B {
    public int b1;
}
```

- แอททริบิวต์ที่เป็นแบบโกลบอลในคลาสที่สืบทอดคุณสมบัติมา จะเป็นการเรียกใช้งานแอททริบิวต์ที่ประกาศเป็นแบบโกลบอลในคลาสที่สืบทอดคุณสมบัติมา ซึ่งสามารถเข้าถึงได้จาก

โทเคนหรือแอททริบิวต์ตัวก่อน เช่น การเรียกใช้แอททริบิวต์ c1 ในเมสเสจ b.c1 ในตารางที่ 2 - 6 จะเป็นการเรียกใช้แอททริบิวต์ c1 ในคลาส C โดยการเรียกใช้งานผ่านแอททริบิวต์ b ในคลาส A เนื่องจากคลาส B ทำการสืบทอดคุณสมบัติมาจากคลาส C

ตารางที่ 2 - 6 ตัวอย่างการเรียกใช้แอททริบิวต์ที่เป็นแอททริบิวต์แบบโกลบอลในคลาสที่สืบทอดคุณสมบัติมา ที่เข้าถึงได้จากโทเคนก่อนหน้า

class A{
public int a1;
B b;
public A(){
b = new B();
a1 = b.**c1**;
}
}
	class B extends C{
	public int b1;
	}
		class C{
		public int c1;
		}

2.4.2 เมสเสจที่เป็นอาร์เรย์

เมสเสจที่เป็นอาร์เรย์ สามารถแบ่งเป็น 2 ส่วน คือ

1 ส่วนหน้าเครื่องหมาย [ ] ซึ่งก็คือ แอททริบิวต์ เช่น การเรียกใช้ b ในเมสเสจ b[getIndex ()] ในตารางที่ 2 - 7 ซึ่งจะเป็นการเรียกใช้งานแอททริบิวต์ b ในคลาส A

2 ส่วนในเครื่องหมาย [ ] ซึ่งก็คือ เมสเสจ เช่น การเรียกใช้ getIndex () ในเมสเสจ b[getIndex()] ในตารางที่ 2 - 8

ตารางที่ 2 - 7 ตัวอย่างการเรียกใช้งานเมสเสจที่เป็นอาร์เรย์ในส่วนก่อนเครื่องหมาย [ ]

```
class A {
    public int index;
    B[] b;
    B b1;
    public A(){
        b = new B[1];
        b1 = b[getIndex()];
    }
    public int getIndex(){
        return index;
    }
}
```

```
class B {
}
```

ตารางที่ 2 - 8 ตัวอย่างการเรียกใช้งานเมสเสจที่เป็นอาร์เรย์ส่วนในเครื่องหมาย [ ]

```
class A {
    public int index;
    B[] b;
    B b1;
    public A(){
        b = new B[1];
        b1 = b[getIndex()];
    }
    public int getIndex(){
        return index;
    }
}
```

```
class B {
}
```

2.4.3 เมสเสจที่เป็นเมธธอด

การเรียกใช้งานเมสเสจที่เป็นเมธธอด สามารถแบ่งได้เป็น 3 ส่วนคือ

1 ส่วนหน้าเครื่องหมาย ( ) ทั้งหมด ยกเว้นโทเคนสุดท้าย ซึ่งก็คือ แอททริบิวต์ ซึ่งอาจมีหรือไม่มีก็ได้ เช่น การเรียกใช้งาน b.c ในเมสเสจ b.c.getInt () ในตารางที่ 2 - 9

ตารางที่ 2 - 9 ตัวอย่างการเรียกใช้งานเมธธอดในส่วนที่เป็นการเรียกใช้แอททริบิวต์

```
class A{
    public int a1;
    B b;
    public A(){
        b = new B();
        b.c.setInt(a1);
    }
}
```

```
class B {
    public int b1;
    public C c;
    public B(){
        c = new C();
    }
}
```

```
class C{
    public int c1;
    public void setInt(int i){
        c1= i;
    }
}
```

2 ส่วนหน้าเครื่องหมาย ( ) โทเคนสุดท้าย ซึ่งจะเป็นชื่อของเมธธอด โดยเมธธอดที่เรียกใช้งานนั้นจะอยู่ที่คลาสใดขึ้นอยู่กับว่ามีโทเคนในหัวที่ข้อ 1 หรือไม่

- ถ้าไม่มี จะเป็นเมธธอดที่อยู่ในคลาสที่เรียกใช้งานหรือคลาสที่สืบทอดคุณสมบัติมา เช่น การเรียกใช้งานเมธธอด getInt () ในตารางที่ 2 - 10 จะเป็นเมธธอดภายในคลาสที่เรียกใช้งาน โดยเมธธอด getInt () จะเป็นเมธธอดที่อยู่ในคลาส A ส่วนตารางที่ 2 - 11 จะเป็นเมธธอดในคลาสที่มีการสืบทอดคุณสมบัติมา โดยเมธธอด getInt () จะเป็นเมธธอดที่อยู่ในคลาส B แต่ A สามารถเรียกใช้งานได้ เนื่องจากคลาส A ทำการสืบทอดคุณสมบัติมาจากคลาส B

ตารางที่ 2 - 10 ตัวอย่างการเรียกใช้งานเมธธอดในส่วนที่เป็นชื่อเมธธอด ที่เป็นเมธธอดภายในคลาสที่เรียกใช้งาน

```
class A {
    public int a1;
    public A(){
        a1 = getInt();
    }
    public int getInt(){
        return 0;
    }
}
```

```
class B {
    public int b1;
}
```

ตารางที่ 2 - 11 ตัวอย่างการเรียกใช้งานเมธธอดในส่วนที่เป็นชื่อเมธธอด ในคลาสที่มีการสืบทอดคุณสมบัติ

```
class A extends B{
    public int a1;
    public A(){
        a1 = getInt();
    }
}
```

```
class B {
    public int b1;
    public int getInt(){
        return 0;
    }
}
```

- ถ้ามี จะเป็นเมธธอดในคลาสที่เข้าถึงได้จากโทเคนก่อนหน้า ซึ่งอาจจะเป็นเมธธอดในคลาสนั้นหรือเมธธอดในที่คลาสดังกล่าวที่มีการสืบทอดคุณสมบัติมาก็ได้ เช่น การเรียกใช้เมธธอด getInt () ในเมสเสจ b.getInt () ในตารางที่ 2 - 12 จะเป็นเมธธอดในคลาสจากโทเคนก่อนหน้า ซึ่งจะเป็นเมท ธอดที่อยู่ในคลาส B โดยสามารถเรียกใช้งานผ่านแอททริบิวต์ b ส่วนในตารางที่ 2 - 13 จะเป็นเมธธอดที่มีการสืบทอดคุณสมบัติมา ซึ่งเมธธอด getInt () จะอยู่ในคลาส C แต่คลาส A สามารถเรียกใช้งานผ่านแอททริบิวต์ b ได้ เนื่องจากคลาส B ทำการสืบทอดคุณสมบัติมาจากคลาส C

ตารางที่ 2 - 12 ตัวอย่างการเรียกใช้งานเมธธอดในส่วนที่เป็นชื่อเมธธอด ที่เป็นเมธธอดภายในคลาสที่เข้าถึงได้จากโทเคนก่อนหน้า

```
class A {
    public int a1;
    B b ;
    public A(){
        b = new B();
        a1 = b.getInt();
    }
    public int getInt(){
        return 0;
    }
}
```

```
class B extends C{
    public int b1;
    public int getInt(){
        return 0;
    }
}
```

```
class C {
}
```

ตารางที่ 2 - 13 ตัวอย่างการเรียกใช้งานเมธธอดในส่วนที่เป็นชื่อเมธธอด ซึ่งคลาสที่เข้าถึงได้จากโทเคนก่อนหน้าสืบทอดคุณสมบัติมา

```
class A {
    public int a1;
    B b ;
    public A(){
        b = new B();
        a1 = b.getInt();
    }
    public int getInt(){
        return 0;
    }
}
```

```
class B extends C{
    public int b1;
}
```

```
class C {
    public int getInt(){
        return 0;
    }
}
```

3 ส่วนอาร์กิวเมนต์ คือ ส่วนในเครื่องหมาย ( ) ซึ่งจะเป็นเมสเสจ เช่น การเรียกใช้ a1 ในเมสเสจ b.c.setInt (a1) ในตารางที่ 2 - 14 ซึ่งจะเป็นเมสเสจแบบแอททริบิวต์ คือ แอททริบิวต์ a1 ในคลาส A หรือ การเรียกใช้ b.getInt () ในเมสเสจ b.c.setInt (b.getInt ()) ซึ่งจะเป็นเมสเสจแบบเมธธอด โดยเมธธอด getInt () เป็นเมธธอดในคลาส B สามารถเรียกใช้งานโดยผ่านแอททริบิวต์ b ที่ประกาศอยู่ในคลาส A ในตารางที่ 2 – 15

ตารางที่ 2 - 14 ตัวอย่างการเรียกใช้งานเมธธอดที่มีอาร์กิวเมนต์เป็นเมสเสจแบบแอททริบิวต์

```
class A{
    public int a1;
    B b;
    public A(){
        b = new B();
        b.c.setInt(a1);
    }
}
```

```
class B {
    public int b1;
    public C c;
    public B(){
        c = new C();
    }
}
```

```
class C{
    public int c1;
    public void setInt(int i){
        c1= i;
    }
}
```

ตารางที่ 2 - 15 ตัวอย่างการเรียกใช้งานเมธธอดที่มีอาร์กิวเมนต์เป็นเมสเสจแบบเมธธอด

```
class A{
    public int a1;
    B b;
    public A(){
        b = new B();
        b.c.setInt( b.getInt() );
    }
}
```

```
class B {
    public int b1;
    public C c;
    public B(){
        c = new C();
    }
    public int getInt(){
        return b1;
    }
}
```

```
class C{
    public int c1;
    public void setInt(int i){
        c1= i;
    }
}
```

2.5 การเปลี่ยนแปลงในซอฟต์แวร์ (Software Change)

การเปลี่ยนแปลงในซอฟต์แวร์โดยทั่ว ๆ ไป สามารถแบ่งเป็นประเภทต่าง ๆ (Li, 1998) ตามการพิจารณา ได้ดังนี้

1. **การพิจารณาจากเวลาที่เกิดการเปลี่ยนแปลง** สามารถแบ่งได้เป็น 2 ประเภท คือ
   - การเปลี่ยนแปลงแบบสถิตย์ (Static Change) เป็นการเปลี่ยนแปลงที่เกิดขึ้นในขณะที่ไม่ได้ประมวลผลโปรแกรม คือ การเปลี่ยนแปลงในซอร์สโค้ดของโปรแกรม
   - การเปลี่ยนแปลงแบบพลวัต (Dynamic Change) เป็นการเปลี่ยนแปลงที่เกิดขึ้น ณ ขณะที่ประมวลผล (Execute) โปรแกรม
2. **การพิจารณาจากผลกระทบที่เกิดจากการเปลี่ยนแปลง** สามารถแบ่งได้เป็น 2 ประเภท คือ
   - การเปลี่ยนแปลงที่มีผลกระทบกับวากยสัมพันธ์ (Syntactic Impact) ซึ่งเป็นการเปลี่ยนแปลงที่เมื่อเกิดขึ้นแล้วจะทำให้โปรแกรมทำงานผิดพลาด คอมไพล์โปรแกรมไม่ผ่าน
   - การเปลี่ยนแปลงที่มีผลกระทบกับความหมาย (Semantic Impact) เป็นการเปลี่ยนแปลงที่เกิดขึ้นแล้ว โปรแกรมยังสามารถทำงานได้ คอมไพล์ผ่าน แต่เกิดผลกระทบทำให้การทำงานในแง่ของความหมายหรือพฤติกรรมการทำงานของโปรแกรมเปลี่ยนแปลงไป

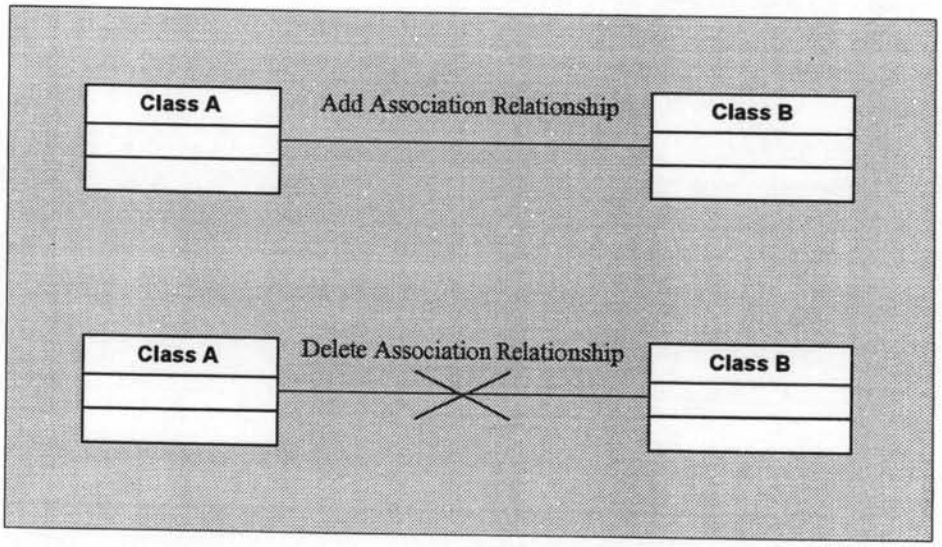
โดยการเปลี่ยนแปลงในซอฟต์แวร์เชิงวัตถุ สามารถแบ่งเป็นประเภทต่าง ๆ ตามจุดที่เกิดการเปลี่ยนแปลง (Elish, 2003 ; Kung, 1994 ; Li, 1996) ได้ดังนี้

2.5.1 การเปลี่ยนแปลงที่ความสัมพันธ์ระหว่างคลาส

- การเพิ่มการสืบทอดคุณสมบัติ เป็นการทำให้คลาสที่เพิ่มการสืบทอดคุณสมบัตินั้น กลายเป็นซับคลาสและสืบทอดคุณสมบัติจากซูปเปอร์คลาสมา การเพิ่มการสืบทอดคุณสมบัติอาจเกิดผลกระทบกับซับคลาสหรือไม่ก็ได้ ทั้งนี้ขึ้นอยู่กับคุณสมบัติที่ซับคลาสได้สืบทอดมานั้น มีเหตุขัดแย้งกันหรือไม่
- การลบการสืบทอดคุณสมบัติ เป็นการทำให้ซับคลาสกลายเป็นคลาสอิสระ ดังนั้น การเปลี่ยนแปลงในลักษณะนี้ อาจจะเกิดผลกระทบหรือไม่ก็ได้ เนื่องจากซับคลาสอาจจะไม่ได้เรียกใช้งานคุณสมบัติที่สืบทอดมา แต่ถ้าหาก

ซับคลาสมีการเรียกใช้งานคุณสมบัติสืบทอดมาก็จะได้รับผลกระทบ เนื่องจากไม่สามารถเรียกใช้งานคุณสมบัติที่สืบทอดมาได้

- การเพิ่มความสัมพันธ์แอสโซซิเอชันระหว่างคลาส เป็นการเพิ่มการเชื่อมต่อการทำงานระหว่างคลาสให้มีการทำงานร่วมกัน ซึ่งคลาสที่เพิ่มการเชื่อมต่อจะต้องมีอย่างน้อยหนึ่งเมธธอดที่ถูกเปลี่ยนแปลง ดังนั้นคลาสที่เพิ่มการเชื่อมต่อความสัมพันธ์จะได้รับผลกระทบจากการเปลี่ยนแปลง แสดงได้ดังรูปที่ 2-13
- การลบความสัมพันธ์แอสโซซิเอชันระหว่างคลาส เป็นการทำให้คลาสที่มีการเชื่อมต่อความสัมพันธ์ไม่มีการทำงานร่วมกัน ซึ่งคลาสที่ถูกลบการเชื่อมต่อจะต้องมีอย่างน้อยหนึ่งเมธธอดที่ถูกเปลี่ยนแปลง ดังนั้นคลาสที่เพิ่มการเชื่อมต่อความสัมพันธ์จะได้รับผลกระทบจากการเปลี่ยนแปลง แสดงได้ดังรูปที่ 2-13

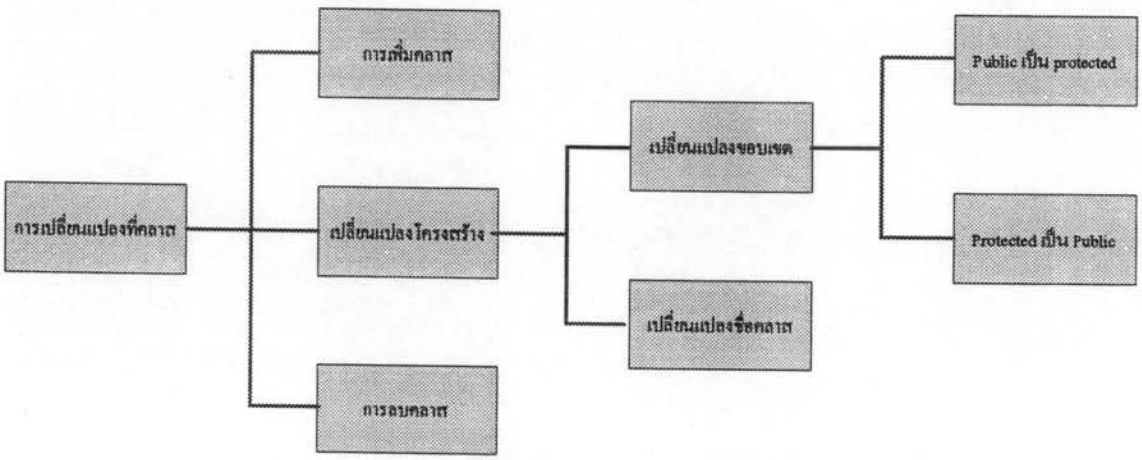


รูปที่ 2 – 13 แสดงการเพิ่มและลบความสัมพันธ์แอสโซซิเอชันระหว่างคลาส (Kung, 1994)

จากรูปที่ 2-13 จะเห็นได้ว่าคลาส A มีการเพิ่มการเชื่อมต่อและลบความสัมพันธ์แอสโซซิเอชันระหว่างคลาส ดังนั้นคลาส A จะถูกเปลี่ยนแปลง และได้รับผลกระทบจากการเปลี่ยนแปลงทั้งการเพิ่มและการลบความสัมพันธ์แอสโซซิเอชันระหว่างคลาส

2.5.2 การเปลี่ยนแปลงที่คลาส

การเปลี่ยนแปลงที่เกิดขึ้นที่คลาส จะพิจารณาเฉพาะการเพิ่มคลาส การลบคลาส และการแก้ไขส่วนโครงสร้างภายนอกของคลาสเท่านั้น ส่วนการแก้ไขส่วนโครงสร้างภายในคลาสนั้นจะแยกพิจารณาเป็นการเปลี่ยนแปลงที่เกิดขึ้นกับแอทริบิวต์ (Attributes) และเมธธอด (Method) ที่จะกล่าวถึงในหัวข้อถัดไป สำหรับการเปลี่ยนแปลงที่คลาส สามารถแสดงได้ ดังรูปที่ 2-14



รูปที่ 2-14 แสดงการเปลี่ยนแปลงที่คลาส (Li, 1996)

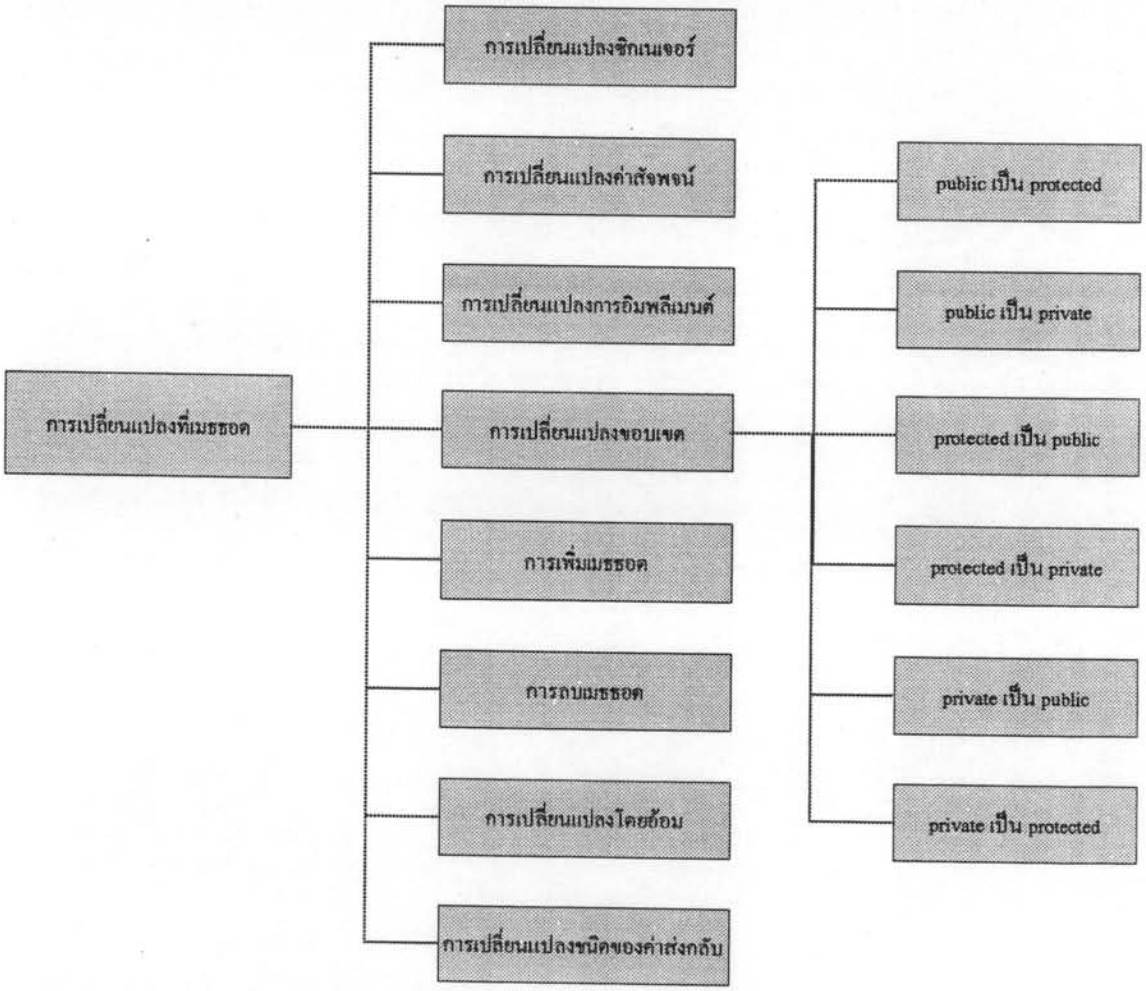
- การเพิ่มคลาส เป็นการเปลี่ยนแปลงที่ไม่มีผลกระทบกับคลาสอื่น ๆ ในระบบ เนื่องจากคลาสที่เพิ่มเข้าไปใหม่นั้น ยังไม่ถูกอ้างอิงจากคลาสใด ๆ
- การลบคลาส เป็นการเปลี่ยนแปลงที่ทำให้คลาสอื่น ๆ ที่เรียกใช้งานคุณสมบัติของคลาสที่ถูกลบทั้งหมดได้รับผลกระทบ เพราะไม่สามารถเรียกใช้งานแอทริบิวต์หรือเมธธอดในคลาสที่ถูกลบไปได้
- การเปลี่ยนแปลงโครงสร้าง ซึ่งเป็นการเปลี่ยนแปลงโครงสร้างภายนอกของคลาส ที่ประกอบด้วยการเปลี่ยนแปลงชื่อคลาสและขอบเขตของคลาส
    (1) การเปลี่ยนแปลงชื่อคลาส เป็นการเปลี่ยนแปลงที่เกิดขึ้นในลักษณะเดียวกันกับการลบคลาส ซึ่งจะไม่ปรากฏชื่อคลาสเดิมให้เรียกใช้งานอีกต่อไป
    (2) การเปลี่ยนแปลงขอบเขตของคลาส มี 2 ประเภท คือ การเปลี่ยนจากพับลิค (public) ไปเป็นโพรเทคท์ (protected) และการเปลี่ยนจากโพรเทคท์ (protected) ไปเป็นพับลิค (public) ซึ่งผลกระทบจะ

เกิดขึ้นในกรณีที่เปลี่ยนแปลงจากพับลิค (public) ไปเป็นโพรเทคท์ (protected) เท่านั้น ส่วนกรณีที่การเปลี่ยนแปลงเกิดจากการเปลี่ยนโพรเทคท์ (protected) ไปเป็นพับลิค (public) นั้นจะไม่ได้รับผลกระทบ

หมายเหตุ สำหรับขอบเขตของคลาสแบบไพรเวท (private) นั้น คลาสจะไม่สามารถประกาศให้คลาสอื่นเข้ามาใช้งานได้

2.5.3 การเปลี่ยนแปลงที่เมธธอด

การเปลี่ยนแปลงที่เมธธอด สามารถแสดงได้ ดังรูปที่ 2-15



รูปที่ 2-15 แสดงการเปลี่ยนแปลงที่เมธธอด (Li, 1996)

- การเปลี่ยนแปลงซิกเนเจอร์ (Signature Change) ซิกเนเจอร์ ประกอบด้วยชื่อของเมธธอด จำนวนพารามิเตอร์ และประเภทของพารามิเตอร์ ตัวอย่างการเปลี่ยนแปลง เช่น การเพิ่ม หรือลดจำนวนพารามิเตอร์ การเปลี่ยนแปลงชื่อเมธธอด เป็นต้น ซึ่งจะส่งผลกระทบไปยังเมธธอดหรือคลาสอื่น ๆ ที่มีการอ้างอิงเมธธอดนี้
- การเปลี่ยนแปลงสัจพจน์ (Axiom Change) เป็นการเปลี่ยนแปลงพรีคอนดิชันโพสคอนดิชัน หรือค่าความจริง โดยอาจจะเป็นการเปลี่ยนแปลงพฤติกรรมหรือความหมายของเมธธอด ซึ่งอาจจะมีผลกระทบกับคลาสที่อ้างอิงเมธธอดนี้ หรืออาจจะไม่เกิดผลกระทบก็ได้
- การเปลี่ยนแปลงอิมพลีเมนต์ (Implementation Change) ซึ่งการเปลี่ยนแปลงประเภทนี้จะกระทบกับรายละเอียดของการอิมพลีเมนต์ แต่จะไม่เกิดผลกระทบกับส่วนอินเทอร์เฟช (Interface) หรือส่วนที่ใช้ติดต่อกับเมธธอด โดยการเปลี่ยนแปลงนี้อาจจะมีผลกระทบกับคลาสที่อ้างอิงเมธธอดนี้หรืออาจจะไม่เกิดผลกระทบก็ได้
- การเปลี่ยนแปลงที่ขอบเขต (Scope Change) เป็นการเปลี่ยนแปลงขอบเขตของการเข้าถึงหรือการอ้างอิงเมธธอด ซึ่งสามารถเปลี่ยนแปลงได้ 6 รูปแบบ ดังตารางที่ 2-16

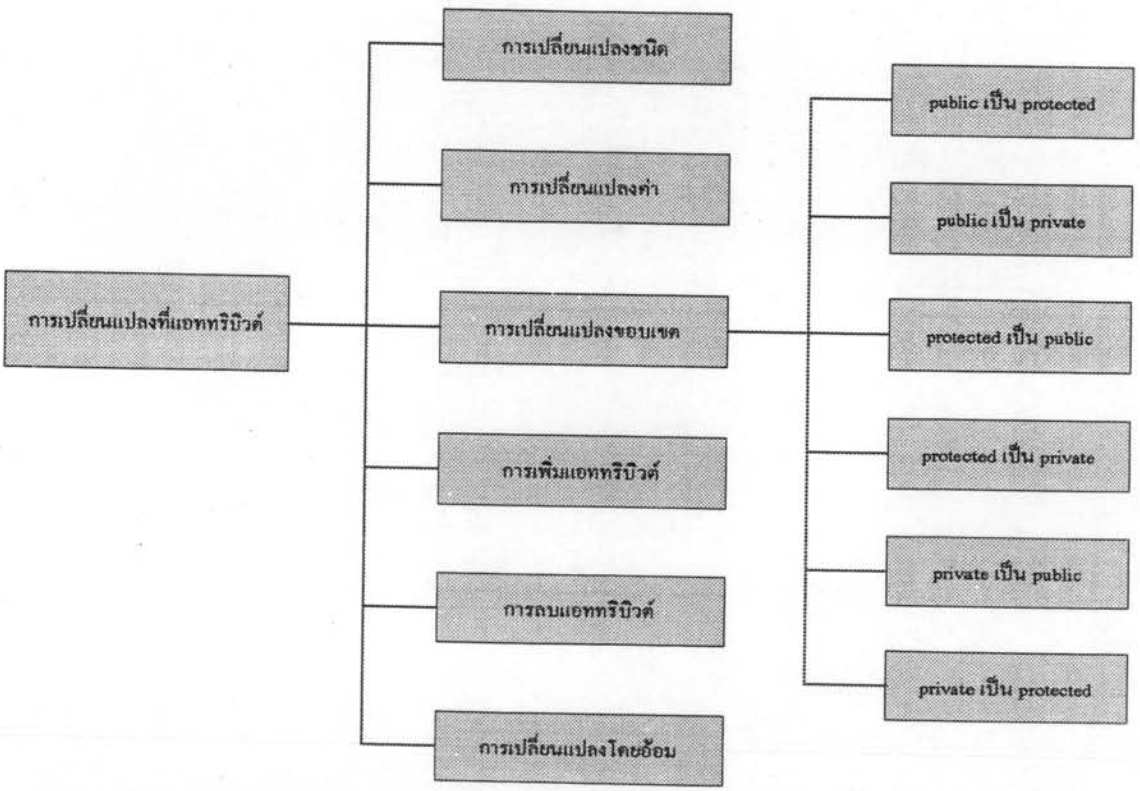
ตารางที่ 2-16 รูปแบบการเปลี่ยนแปลงที่ขอบเขตของเมธธอด

การเปลี่ยนแปลงขอบเขตของเมธธอด	คลาสที่ได้รับผลกระทบ
พับลิค (public) → ไพรเวท (private)	ทุกคลาสที่เรียกใช้เมธธอดนี้
พับลิค (public) → โพรเทคท์ (protected)	คลาสที่เรียกใช้เมธธอดนี้ทั้งหมด ยกเว้นคลาสลูก
โพรเทคท์ (protected) → ไพรเวท (private)	คลาสลูก
โพรเทคท์ (protected) → พับลิค (public)	ไม่ได้รับผลกระทบ
ไพรเวท (private) → พับลิค (public)	ไม่ได้รับผลกระทบ
ไพรเวท (private) → โพรเทคท์ (protected)	ไม่ได้รับผลกระทบ

- การเปลี่ยนแปลงโดยการลบเมธธอด ซึ่งทุก ๆ คลาสที่เรียกใช้เมธธอดนี้ จะได้รับผลกระทบ
- การเปลี่ยนแปลงโดยการเพิ่มเมธธอด เมื่อมีการเพิ่มเมธธอด จะยังไม่มีการถูกเรียกใช้งาน แต่คลาสลูกจะรับรู้ว่า จะมีเมธธอดใหม่เกิดขึ้น จึงไม่มีผลกระทบกับคลาสใด ๆ
- การเปลี่ยนแปลงชนิดของค่าส่งกลับ จะทำให้เมธธอดที่เรียกใช้งานได้รับผลกระทบ
- การเปลี่ยนแปลงโดยอ้อม เป็นการเปลี่ยนแปลงที่เกิดจากเมธธอดเรียกใช้แอททริบิวต์หรือเมธธอดอื่นที่ได้รับผลกระทบจากการเปลี่ยนแปลงเมธธอด จึงอาจได้รับผลกระทบได้

2.5.4 การเปลี่ยนแปลงที่แอททริบิวต์

การเปลี่ยนแปลงที่แอททริบิวต์ เป็นการเปลี่ยนแปลงที่เกิดขึ้นกับตัวแปร (Variable) ที่ใช้ในการเก็บข้อมูลของออบเจคท์ ซึ่งการเปลี่ยนแปลงที่แอททริบิวต์สามารถแสดงได้ ดังรูปที่ 2-16



รูปที่ 2-16 แสดงการเปลี่ยนแปลงที่แอททริบิวต์ (Li, 1996)

- การเปลี่ยนแปลงชนิด (Type Change) เป็นการเปลี่ยนแปลงที่จะทำให้คลาสที่อ้างอิงถึงแอททริบิวต์นี้ได้รับผลกระทบ
- การเปลี่ยนแปลงค่า (Value Change) เป็นการเปลี่ยนแปลงค่าของแอททริบิวต์ ซึ่งอาจจะทำให้เกิดผลกระทบกับคลาสอื่น ๆ หรือไม่เกิดผลกระทบก็ได้ ขึ้นอยู่กับว่าค่าที่เปลี่ยนแปลงนั้นทำให้สถานะของออบเจคท์เปลี่ยนแปลงหรือไม่ หากมีการเปลี่ยนแปลงสถานะอาจจะทำให้เส้นทาง (Path) ในการทำงานเปลี่ยนแปลงไป ซึ่งจะส่งผลกระทบกับคลาสนั้น ๆ ได้
- การเปลี่ยนแปลงขอบเขตของแอททริบิวต์ ซึ่งการเปลี่ยนแปลงนี้จะเป็นเช่นเดียวกันกับ การเปลี่ยนแปลงขอบเขตของเมธธอด โดยสามารถเปลี่ยนแปลงได้ 6 กรณีเช่นกัน ดังตารางที่ 2-17

ตารางที่ 2-17 รูปแบบการเปลี่ยนแปลงที่ขอบเขตของแอททริบิวต์

การเปลี่ยนแปลงขอบเขตของแอททริบิวต์	คลาสที่ได้รับผลกระทบ
พับลิค (public) → ไพรเวท (private)	ทุกคลาสที่เรียกใช้แอททริบิวต์นี้
พับลิค (public) → โพรเทคท์ (protected)	คลาสที่เรียกใช้แอททริบิวต์นี้ทั้งหมด ยกเว้นคลาสลูก
โพรเทคท์ (protected) → ไพรเวท (private)	คลาสลูก
โพรเทคท์ (protected) → พับลิค (public)	ไม่ได้รับผลกระทบ
ไพรเวท (private) → พับลิค (public)	ไม่ได้รับผลกระทบ
ไพรเวท (private) → โพรเทคท์ (protected)	ไม่ได้รับผลกระทบ

- การเปลี่ยนแปลงโดยการเพิ่มแอททริบิวต์ ซึ่งการเปลี่ยนแปลงโดยการเพิ่มแอททริบิวต์นี้ จะยังไม่มีการอ้างอิงหรือเข้าถึงแอททริบิวต์นั้น แต่คลาสลูกจะรับรู้ว่าจะมีแอททริบิวต์ใหม่เกิดขึ้น ทำให้ยังไม่มีผลกระทบกับคลาสใด เนื่องจากยังไม่มีการเรียกใช้งาน
- การเปลี่ยนแปลงโดยการลบแอททริบิวต์ การเปลี่ยนแปลงนี้จะมีความคล้ายคลึงกับการลบเมธธอด โดยผลกระทบที่ได้รับนั้นจะมีมากหรือน้อยนั้นขึ้นอยู่กับขอบเขตของแอททริบิวต์นั้น ๆ

- การเปลี่ยนแปลงโดยอ้อม เป็นการเปลี่ยนแปลงที่เกิดจากแอททริบิวต์เรียกใช้แอททริบิวต์หรือเมธธอดอื่น ๆ ที่ได้รับผลกระทบจากการเปลี่ยนแปลงแอททริบิวต์ ซึ่งอาจจะทำให้ได้รับผลกระทบหรือไม่ก็ได้

2.6 ประสิทธิภาพของซอฟต์แวร์ (Software Performance)

ประสิทธิภาพในการทำงานของซอฟต์แวร์ ได้ถูกกำหนดให้เป็นหนึ่งในคุณลักษณะของคุณภาพ (Quality Attribute) ซึ่งเป็นปัจจัยที่มีผลกับคุณภาพของซอฟต์แวร์ โดยในปัจจุบันคุณภาพของซอฟต์แวร์ ได้มีผู้ให้นิยามและความหมายไว้ในหลายมุมมองด้วยกัน ดังนี้

- **Kan (2000)** ได้อธิบายว่า "คุณภาพซอฟต์แวร์ไม่ใช่มุมมองเพียงด้านเดียวแต่ทว่าเป็นมุมมองหลายๆ ด้าน โดยมุมมองของคุณภาพนั้นคือคุณลักษณะของคุณภาพ (Quality Attributes) ของแต่ละด้าน"
- **ไอเอสโอ 9126 (ISO 9126)** (2001) ได้ให้นิยามความหมายว่า "คุณภาพซอฟต์แวร์เป็นคุณลักษณะของคุณภาพซอฟต์แวร์เป็นกลุ่มของคุณลักษณะในผลิตภัณฑ์ซอฟต์แวร์ (Product Quality Attributes) ได้แก่ ฟังก์ชันนอลลิตี้ (Functionality) ความน่าเชื่อถือ (Reliability) ความสามารถในการใช้งานง่าย (Usability) ความมีประสิทธิภาพ (Efficiency) การบำรุงรักษาง่าย (Maintainability) และความสามารถในการเคลื่อนย้าย (Portability) ซึ่งคุณลักษณะเหล่านี้นำมาใช้สำหรับการประเมินค่าคุณภาพซอฟต์แวร์"

จากนิยามความหมายต่าง ๆ ของคุณภาพของซอฟต์แวร์ที่กล่าวมาข้างต้น ทำให้เห็นคุณภาพของซอฟต์แวร์ในหลาย ๆ มุมมอง ซึ่งโดยส่วนใหญ่แล้วคุณภาพของซอฟต์แวร์ จะมีการกล่าวอ้างถึงคุณลักษณะของคุณภาพ (Quality Attributes) ของผลผลิตซอฟต์แวร์ (Software Product) โดยงานวิจัยนี้สนใจเพียงคุณลักษณะของคุณภาพในด้านประสิทธิภาพการทำงาน

ประสิทธิภาพในการทำงาน (Performance) คือ การตอบสนองของซอฟต์แวร์ระบบ หรืออาจจะหมายถึง เวลาที่ระบบจำเป็นต้องใช้ในการตอบสนองต่อเหตุการณ์ (Event) หนึ่ง ๆ หรือจำนวนของเหตุการณ์ที่ระบบสามารถประมวลผลได้ในช่วงเวลาหนึ่ง ๆ (Bass et al., 1998) ซึ่งจะเป็นคุณลักษณะของคุณภาพที่มองเห็น ณ ขณะซอฟต์แวร์ทำงาน (Run - Time) หรือขณะประมวลผล (Execution) (McCall, 1997)

โดยประสิทธิภาพในการทำงาน สามารถวัดได้โดยความเร็วในการประมวลผลข้อมูล (Grady and Caswell, 1987) ซึ่งมักจะถูกคำนวณโดยวัดจากจำนวนทรานแซคชัน (Transactions) ต่อหนึ่งหน่วยเวลา (Unit – Time) หรือระยะเวลาที่ทรานแซคชันหนึ่ง ๆ จะถูกประมวลผลโดยสมบูรณ์

ดังนั้น ถ้าทรานแซคชันที่มีการเชื่อมต่อข้อมูลกันมากระยะเวลาในการประมวลผลจะมากขึ้นด้วย (Bass et al., 1998)

Baker และ Shih (1992) ศึกษาและหาวิธีการและสร้างรูปแบบของซอฟต์แวร์ขนาดใหญ่ให้มีประสิทธิภาพ รวมถึงวิธีการประเมินค่าประสิทธิภาพของซอฟต์แวร์ (Software Performance) โดยรูปแบบประสิทธิภาพของซอฟต์แวร์ มีการเสนอให้พิจารณารูปแบบของการประมวลผลซอฟต์แวร์ในสภาวะแวดล้อมที่มีความซับซ้อน ซึ่งประสิทธิภาพของซอฟต์แวร์ของงานวิจัยนี้พิจารณาได้จากจำนวนปริมาณงานที่มีการทำงานในซอฟต์แวร์ (Throughput) หรือพิจารณาจากระยะเวลาของการตอบสนองการทำงานของซอฟต์แวร์ในขณะประมวลผล (Response Time)

Munson (1992) ต้องการวัดความซับซ้อนของซอฟต์แวร์ระหว่างที่ซอฟต์แวร์มีการประมวลผล (Dynamic Metric) แต่ประสิทธิภาพของซอฟต์แวร์มีการแปรผันขึ้นอยู่กับความซับซ้อนของซอฟต์แวร์ ดังนั้นมาตรวัดซอฟต์แวร์จึงเป็นวิธีการที่เหมาะสมในการทำนายประสิทธิภาพ เนื่องจากประสิทธิภาพของซอฟต์แวร์ขึ้นอยู่กับความซับซ้อนของซอฟต์แวร์ ดังนั้นงานวิจัยนี้จึงใช้มาตรวัดความซับซ้อนของซอฟต์แวร์เพื่อทำนายประสิทธิภาพของซอฟต์แวร์และความน่าเชื่อถือของซอฟต์แวร์ โดยตัวอย่างของมาตรวัดความซับซ้อนของซอฟต์แวร์ที่ใช้ในงานวิจัยนี้ เช่น

- มาตรวัดไซโคลเมติกของแมคแคบ (Cyclomatric Complexity : V (G)) เป็นมาตรวัดที่ใช้วัดความซับซ้อนของเส้นทางในการท่องโปรแกรมหรือจำนวนเส้นทางอิสระในโปรแกรมซึ่งพิจารณาจากจำนวนกิ่ง (Branch) ภายในโปรแกรม
- มาตรวัดการเข้าคู่กันระหว่างออบเจคท์ (Coupling between Object : CBO) การเข้าคู่กันหมายถึง การที่ออบเจคท์หรือคลาสหนึ่งมีการเรียกใช้ออบเจคท์หรือคลาสอื่น ๆ ซึ่งยังหมายรวมถึงการเรียกใช้เมทธอดหรือแอททริบิวต์ในคลาสอื่น ๆ ด้วย

ซึ่งงานวิจัยเชื่อว่าความซับซ้อนของซอฟต์แวร์มีผลในการทำนายประสิทธิภาพของซอฟต์แวร์และความน่าเชื่อถือของซอฟต์แวร์ระหว่างที่ซอฟต์แวร์มีการประมวลผลได้ โดยสอดคล้องกับงานวิจัยของ Sarker (2005) วัดความซับซ้อนของซอฟต์แวร์จากระดับความสัมพันธ์ระหว่างคลาส (Coupling) ในโครงสร้างของซอฟต์แวร์ ซึ่งขึ้นอยู่กับความซับซ้อนของการส่งผ่านข้อมูลระหว่างคลาส ได้แก่ การส่งข้อความหรือเมสเสจ (Message Calling) ระหว่างคลาสทั้งหมดที่มีความสัมพันธ์กัน

2.7 งานวิจัยที่เกี่ยวข้อง

ผลกระทบในการเปลี่ยนแปลงซอฟต์แวร์ที่ออกแบบด้วยโครงสร้างคลาสที่ต่างกัน โดยการพิจารณาถึงความเกี่ยวข้องกันระหว่างการออกแบบโครงสร้างหรือคุณลักษณะของคลาสและคุณภาพของซอฟต์แวร์ เพื่อวิเคราะห์ถึงผลกระทบและประสิทธิภาพของซอฟต์แวร์ขณะประมวลผลระหว่างการเปลี่ยนแปลงโครงสร้างคลาส มีงานวิจัยที่เกี่ยวข้องดังนี้

2.7.1 อัลกอริทึมในการวิเคราะห์ผลกระทบของการเปลี่ยนแปลงต่อซอฟต์แวร์เชิงวัตถุ (Algorithm Analysis of the Impact of Change to Object – Oriented Software) งานวิจัยของ Li และ Offutt (1996)

งานวิจัยนี้ ได้รวบรวมรูปแบบของการเปลี่ยนแปลงที่เกิดขึ้นในซอฟต์แวร์เชิงวัตถุ โดยการแบ่งการเปลี่ยนแปลงที่เกิดขึ้นกับแอททริบิวต์ การเปลี่ยนแปลงที่เกิดขึ้นกับเมธธอด และการเปลี่ยนแปลงที่เกิดขึ้นกับคลาส นอกจากนี้ ยังได้แบ่งลักษณะของการเปลี่ยนแปลงตามคลาสที่ถูกกระทบจากการเปลี่ยนแปลง ออกเป็น 5 แบบ ดังนี้

1. **การเปลี่ยนแปลงแบบกระทบทั้งหมด (Contaminate All)** เป็นการเปลี่ยนแปลงที่มีผลกระทบกับแอททริบิวต์และเมธธอดในทุก ๆ คลาสที่เกี่ยวข้องกับคลาสที่เกิดการเปลี่ยนแปลง
2. **การเปลี่ยนแปลงที่กระทบกับคลาสปัจจุบัน (Contaminate Current)** เป็นการเปลี่ยนแปลงที่มีผลกระทบกับสมาชิกข้อมูลและเมธธอดในคลาสที่เกิดการเปลี่ยนแปลงนั้น
3. **การเปลี่ยนแปลงกระทบกับคลาสลูก (Contaminate Children)** เป็นการเปลี่ยนแปลงที่มีผลกับคลาสที่มีการสืบทอดคุณสมบัติจากคลาสที่เกิดการเปลี่ยนแปลง
4. **การเปลี่ยนแปลงกระทบกับไคลเอนต์ (Contaminate Client)** เป็นการเปลี่ยนแปลงที่มีผลกับคลาสที่เรียกใช้งานคลาสที่เกิดการเปลี่ยนแปลง
5. **การเปลี่ยนแปลงที่ไม่กระทบ (Contaminate None)** เป็นการเปลี่ยนแปลงที่เกิดขึ้นแล้วไม่มีผลกระทบกับคลาสใด ๆ ทั้งสิ้น

จากงานวิจัยนี้ ผู้วิจัยได้นำลักษณะของการเปลี่ยนแปลงที่เกิดขึ้นกับซอฟต์แวร์เชิงวัตถุมาใช้เพื่อนำมาวิเคราะห์ถึงผลกระทบที่เกิดจากการเปลี่ยนแปลงความต้องการของซอฟต์แวร์ของคลาสระหว่างโครงสร้างคลาสที่แตกต่างกันว่ามีผลต่อการแก้ไข ปรับปรุงและบำรุงรักษาซอฟต์แวร์อย่างไร

2.7.2 การศึกษาผลกระทบของการสืบทอดคุณสมบัติที่มีต่อการบำรุงรักษาซอฟต์แวร์เชิงวัตถุ (The Effect of Inheritance on the Maintainability of Object – Oriented Software : An Empirical Study) งานวิจัยของ Daly และคณะ (1995)

ในงานวิจัยนี้ ได้ศึกษาการสืบทอดคุณสมบัติของคลาส ว่ามีผลต่อการบำรุงรักษา (Maintainability) ซอฟต์แวร์เชิงวัตถุหรือไม่ โดยการเปรียบเทียบระหว่างซอฟต์แวร์ที่มีลักษณะการสืบทอดคุณสมบัติกับซอฟต์แวร์ที่ไม่มีระดับความลึกในการสืบทอดคุณสมบัติ ซึ่งงานวิจัยได้กำหนดให้ระดับความลึกของการสืบทอดคุณสมบัติ (Depth of Inheritance) ของคลาสเพียง 3 ระดับ โดยตั้งสมมติฐานของการทดลอง ดังนี้

H_0 : การสืบทอดคุณสมบัติของคลาสที่มีความลึก 3 ระดับไม่มีผลกระทบกับการบำรุงรักษาซอฟต์แวร์เชิงวัตถุ

H_1 : การสืบทอดคุณสมบัติของคลาสที่มีความลึก 3 ระดับไม่มีผลกระทบกับการบำรุงรักษาซอฟต์แวร์เชิงวัตถุ

สำหรับขั้นตอนในการทดลอง แบ่งได้ดังนี้

1. การเลือกกลุ่มตัวอย่าง จะเลือกจากนักศึกษาที่ลงทะเบียนเรียนวิชาการโปรแกรมเชิงวัตถุ (Object – Oriented Programming) โดยภาษาโปรแกรมที่ใช้ในการทดลอง คือ ภาษาซีพลัสพลัส (C++ Programming Language)
2. แบ่งกลุ่มตัวอย่างออกเป็น 2 กลุ่ม โดยแต่ละกลุ่มจะทำการทดสอบโปรแกรม ดังตารางที่ 2-18

ตารางที่ 2-18 การแบ่งกลุ่มหน่วยตัวอย่างเพื่อทำการทดลองศึกษาผลกระทบของการสืบทอดคุณสมบัติที่มีต่อการบำรุงรักษาซอฟต์แวร์เชิงวัตถุ

กลุ่มตัวอย่าง	การทดสอบที่ 1	การทดสอบที่ 2
A	โปรแกรมที่ 1 : มีระดับความลึกในการสืบทอดคุณสมบัติ	โปรแกรมที่ 2 : ไม่มีระดับความลึกในการสืบทอดคุณสมบัติ
B	โปรแกรมที่ 1 : ไม่มีระดับความลึกในการสืบทอดคุณสมบัติ	โปรแกรมที่ 2 : มีระดับความลึกในการสืบทอดคุณสมบัติ

3. การทดสอบการบำรุงรักษาซอฟต์แวร์เชิงวัตถุ จะกระทำในขั้นตอนของเฟสการอิมพลีเมนต์โปรแกรม (Implement Phase) ซึ่งงานวิจัยได้กำหนดงานที่จะใช้

ในการแก้ไขเปลี่ยนแปลง (Maintenance Tasks) โปรแกรมแต่ละโปรแกรมไว้ให้กลุ่มตัวอย่างแก้ไขตามงานที่กำหนด

4. โดยผลกระทบจากการเปลี่ยนแปลงซอฟต์แวร์ งานวิจัยนี้พิจารณาจากระยะเวลาเฉลี่ยที่แต่ละกลุ่มตัวอย่างใช้ไปสำหรับการแก้ไขโปรแกรมให้ทำงานได้สมบูรณ์

จากผลการทดลองของงานวิจัยนี้สอดคล้องกับสมมติฐานที่ตั้งไว้ คือ โปรแกรมที่มีการสืบทอดคุณสมบัติที่ระดับความลึก 3 ระดับมีความสามารถในการปรับปรุงและแก้ไขซอฟต์แวร์ระหว่างการอิมพลีเมนต์ได้รวดเร็วกว่าซอฟต์แวร์ที่ไม่มีระดับความลึกในการสืบทอดคุณสมบัติ

ซึ่งในงานวิจัยนี้พิจารณาเพียงผลกระทบในการบำรุงรักษาซอฟต์แวร์จากระยะเวลาที่ใช้ในการแก้ไขโปรแกรม ซึ่งยังไม่มีการพิจารณาถึงประสิทธิภาพของซอฟต์แวร์ขณะประมวลผล และผลกระทบจากการเปลี่ยนแปลงความต้องการของซอฟต์แวร์จากการเปลี่ยนคุณลักษณะของคลาส ดังนั้น ผู้วิจัยจึงได้ทำการทดลองโดยการพิจารณาประสิทธิภาพของซอฟต์แวร์และผลกระทบจากการเปลี่ยนแปลงซอฟต์แวร์เพิ่มเติม