บทที่ 4
ยูเอ็มแอล

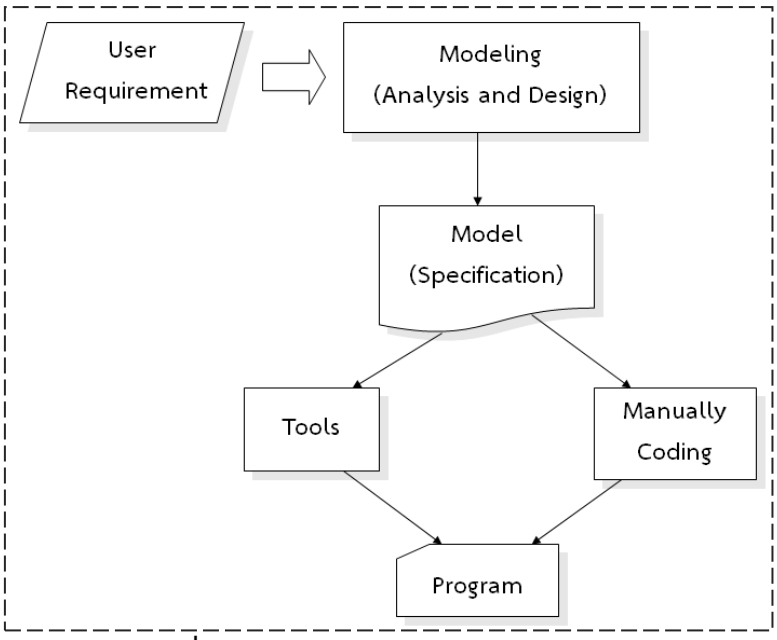
เกริ่นนำ

ในบทนี้จะอธิบายถึงความหมายของตัวแบบ แบบจำลอง ความหมายของยูเอ็มแอล มุมมอง ความสำคัญและประโยชน์ของยูเอ็มแอล แผนภาพต่าง ๆ ในยูเอ็มแอล โดยจะอธิบายแต่ละแผนภาพที่มีอยู่ในยูเอ็มแอลในเบื้องต้นก่อน ส่วนในบทต่อ ๆ ไปจะลงลึกในรายละเอียดของแต่ละแผนภาพและสัญลักษณ์ต่าง ๆ ที่ใช้ รวมถึงขั้นตอนวิธีการวิเคราะห์และออกแบบระบบเชิงวัตถุโดยใช้แผนภาพต่าง ๆ ของยูเอ็มแอล

ความหมายของตัวแบบและแบบจำลอง

ตัวแบบ (Model) หมายถึง การสร้างแบบจำลอง (Modeling) เป็นวิธีการวิเคราะห์และออกแบบ (Analysis and Design) วิธีการหนึ่งที่เน้นการสร้างแบบจำลองเพื่อให้สามารถเข้าใจปัญหาของระบบได้ โดยใช้เป็นเครื่องมือในการสื่อสารแนวความคิดในการพัฒนาระบบร่วมกันกับบุคคลอื่น ๆ ในทีม

แบบจำลอง (Modeling) หมายถึง วิธีการวิเคราะห์ออกแบบ (Analysis and Design) อย่างหนึ่งที่เน้นการใช้งานแบบจำลองเป็นหลักซึ่งแบบจำลองที่สร้างขึ้นมาจะสามารถช่วยให้เข้าใจในปัญหาได้ง่ายขึ้นอีก ทั้งยังสามารถนำแบบจำลองมาเป็นเครื่องมือในการสื่อสารถ่ายทอดความคิดกับบุคคลอื่น ๆ ที่เกี่ยวข้องในโครงการได้ เช่น เจ้าของกิจการ ลูกค้า เจ้าหน้าที่ พนักงาน นักวิเคราะห์ระบบนักออกแบบระบบ เป็นต้น (ชาคริต กุลไกรศรี, 2556) ส่วนแบบจำลองภาพคือการใช้สัญลักษณ์รูปภาพในการสร้างแบบจำลองของระบบที่จะพัฒนาเพื่อประโยชน์ที่คล้ายคลึงกันในการทำความเข้าใจกับความต้องการของลูกค้าการออกแบบระบบที่เป็นไปได้อย่างชัดเจนขึ้นและการบำรุงรักษาที่ง่ายยิ่งขึ้นแบบจำลองเกิดขึ้นโดยการนำเสนอส่วนต่าง ๆ ของระบบแต่เพียงส่วนที่สำคัญโดยไม่คำนึงถึงรายละเอียดปลีกย่อยต่าง ๆ ในการพัฒนาระบบซอฟต์แวร์ที่ซับซ้อน นักพัฒนาจำเป็นต้องทำความเข้าใจกับมุมมองด้านต่าง ๆ ของระบบก่อนทำการพัฒนาจริงโดยการสร้างแบบจำลองอันเปรียบเสมือนพิมพ์เขียวที่แสดงถึงภาพรวมทั้งหมดของระบบแบบจำลองที่สร้างขึ้นจะต้องมีความสอดคล้องกับความต้องการของผู้ใช้งานระบบเป็นสำคัญ ในส่วนของรายละเอียดต่าง ๆ จะถูกเพิ่มเติมลงไปในตัวแบบจำลองและในที่สุดแบบจำลองจะถูกนำไปพัฒนาขึ้นเป็นระบบจริง (ที่มา: https://msit5.wordpress.com/2013/09/04/uml-คืออะไร/) โดยตัวอย่างของแบบจำลองซอฟต์แวร์ (Software Modeling) ดังแสดงในภาพที่ 4.1



ภาพที่ 4.1 แสดงตัวอย่างแบบจำลองซอฟต์แวร์

จากรูปที่ 4.1 เป็นแบบจำลองของการพัฒนาซอฟต์แวร์ซึ่งสามารถอธิบายได้ว่าการพัฒนาซอฟต์แวร์เริ่มต้นจากกระบวนรวบรวมข้อมูลความต้องการของผู้ใช้ระบบแล้วนำมาวิเคราะห์เพื่อให้ได้ข้อกำหนดความต้องการ (Requirement Specification) จากนั้นก็ทำการออกแบบระบบด้วยตัวแบบต่าง ๆ แล้วลงมือพัฒนาระบบโดยใช้เครื่องมือหรือเขียนคำสั่ง จนได้ซอฟต์แวร์ที่ตรงกับความต้องการของผู้ใช้

การเปรียบเทียบแบบจำลองสำหรับเชิงโครงสร้างและแบบจำลองสำหรับเชิงวัตถุสามารถแสดงได้ดังตารางที่ 4.1

ตารางที่ 4.1 เปรียบเทียบแบบจำลองที่ใช้ในขั้นตอนการวิเคราะห์และออกแบบระบบระหว่างแบบจำลองเชิงโรงสร้างและแบบจำลองเชิงวัตถุ

ขั้นตอน / วิธีการ	แบบจำลองสำหรับเชิงโครงสร้าง	แบบจำลองสำหรับเชิงวัตถุ
ขั้นตอนการวิเคราะห์	1. แผนภาพกระแสการไหลข้อมูล (DFD: Data Flow Diagrams) 2. แผนภาพอีอาร์ (Entity Relationship Diagrams) 3.แผนภาพการเปลี่ยนสถานะ (State-Transition Diagrams)	1. Use-case Diagrams 2. Class and Object Diagrams 4. Sequence Diagram 5. Collaboration Diagram 6. State Diagram 7. Activity Diagram
ขั้นตอนการออกแบบ	1. ขั้นตอนวิธี (Algorithm) 2. รหัสจำลอง (Pseudo Code) 3. ผังงาน (Flowchart)	1. Component Diagram 2. Deployment Diagram

ความหมายของยูเอ็มแอล

ยูเอ็มแอล (UML: Unified Modeling Language) เป็นภาษาที่ใช้อธิบายแบบจำลองต่าง ๆ ซึ่งเป็นภาษาสัญลักษณ์รูปภาพมาตรฐานสำหรับใช้ในการสร้างแบบจำลองเชิงวัตถุ โดยยูเอ็มแอลเป็นภาษามาตรฐานสำหรับสร้างแบบพิมพ์เขียวให้แก่ระบบงาน นักวิเคราะห์ระบบสามารถใช้ยูเอ็มแอลในการสร้างมุมมอง กำหนดรายละเอียด สร้างระบบงานและจัดทำเอกสารอ้างอิงให้แก่ระบบงานได้ เนื่องจากยูเอ็มแอลเป็นภาษาที่มีการใช้สัญลักษณ์รูปภาพ จึงอาจมีผู้เข้าใจสับสนว่ายูเอ็มแอลเป็นการสร้างแผนภาพหรือเป็นเพียงการใช้สัญลักษณ์เพื่ออธิบายระบบงานเท่านั้น แต่ยูเอ็มแอลมีลักษณะของแบบจำลองข้อมูลคือเป็นแบบจำลองที่เอาไว้อธิบายแบบจำลองอื่น ๆ อีกที การใช้งานภาษายูเอ็มแอล นอกจากจะต้องเข้าใจในแนวความคิดเชิงวัตถุแล้ว ยังจำเป็นต้องมีพื้นฐานความเข้าใจเกี่ยวกับแบบจำลองภาพด้วยเช่นกัน (พนิดา พานิชกุล, 2548)

เป็นภาษาที่ใช้ในการสร้างแบบจำลอง (Modeling Language) ที่ประกอบด้วยองค์ความรู้ที่ใช้ในการนำเสนอและออกแบบเอกสารประกอบโปรแกรมโดยรวม 3 แนวคิดหรือวิธีการเชิงวัตถุซึ่งได้แก่ Booch Rumbaugh และ Jacobson รวมทั้งจากบุคคลอื่น ๆ มาจัดเป็นมาตรฐานสำหรับการแลกเปลี่ยนแนวคิดการออกแบบระบบและองค์ความรู้ในเชิงเทคนิคในรูปของแบบจำลองยูเอ็มเอลนั้นไม่ใช่ภาษาคอมพิวเตอร์ที่ใช้สำหรับการเขียนโปรแกรมแต่เป็นวิธีการหรือหรือกระบวนการ (Methodology) ซึ่งยูเอ็มเอลจะถูกใช้สำหรับการสร้างแบบจำลองในการวิเคราะห์และออกแบบระบบที่ไม่ขึ้นกับกรรมวิธีโดยแต่ละโครงงานสามารถที่จะเลือกกรรมวิธีการทำงานที่เหมาะสมกับสภาพความจริงของโครงงานได้และไม่ขึ้นกับภาษาคอมพิวเตอร์ภาษาใดภาษาหนึ่ง (Denis, A. Wixom, B. and Tegarden, D. 2005)

Grady Booch, Ivar Jacobson และ Jim Rumbaugh ทั้งสามได้ร่วมมือกันพัฒนาโมเดลในการพัฒนาระบบเชิงวัตถุและได้นิยามไว้ว่ายูเอ็มแอลเป็นภาษาสัญลักษณ์ที่ใช้อธิบายและแสดงรายละเอียดจำลองการสร้างและจัดการเอกสารต่าง ๆ ในระบบ เพื่อให้การออกแบบซอฟต์แวร์สามารถทำได้โดยง่ายและปรับปรุงวิธีการทำงานให้ดียิ่งขึ้น โดยก่อนหน้านั้นทั้งสามคนต่างก็มีโมเดลในการพัฒนาระบบเชิงวัตถุคนละแบบและเป็นในแบบของตัวเอง ซึ่งต่อมาบริษัท Rational ได้ร่วมมือให้บุคคลทั้งสามทำการพัฒนาโมเดลร่วมกันจึงเป็นที่มาของยูเอ็มแอลซึ่งเป็นโมเดลที่สื่อสารด้วยภาพ (Visual Modeling) โดยแต่ละโมเดลจะแสดงมุมมองที่มีต่อระบบแตกต่างกันดังนั้นยูเอ็มแอลก็เป็นวิธีการหรือกระบวนการ (Methodology) หนึ่งเช่นเดียวกับการวิเคราะห์ระบบเชิงโครงสร้างที่ใช้แผนภาพกระแสข้อมูลและแผนภาพอีอาร์

สรุปความหมายของยูเอ็มแอล คือ เป็นภาษาที่ใช้ในการอธิบายแบบจำลองต่าง ๆ หรือเป็นภาษาสัญลักษณ์ที่เป็นรูปภาพมาตรฐานสำหรับใช้ในการสร้างแบบจำลองตามหลักการเชิงวัตถุ โดยยูเอ็มแอลจัดเป็นภาษามาตรฐานสำหรับสร้างแบบพิมพ์เขียวให้แก่ระบบงานสารสนเทศสามารถใช้ยูเอ็มแอลในการสร้างมุมมอง กำหนดรายละเอียดสร้างระบบงานและจัดทำเอกสารอ้างอิงให้แก่ระบบงานได้ เนื่องจากยูเอ็มแอลเป็นภาษาที่มีการใช้สัญลักษณ์รูปภาพ จึงอาจมีผู้เข้าใจสับสนว่า ยูเอ็มแอลเป็นการสร้างแผนภาพหรือเป็นเพียงการใช้สัญลักษณ์เพื่ออธิบายระบบงานเท่านั้นแต่แท้จริงแล้ว ยูเอ็มแอลมีลักษณะของแบบจำลองข้อมูลคือเป็น

แบบจำลองที่เอาไว้อธิบายแบบจำลองอื่น ๆ อีกทีการใช้งานภาษายูเอ็มแอล นอกจากจะต้องเข้าใจในแนวความคิดเชิงวัตถุแล้วยังจำเป็นต้องมีพื้นฐานความเข้าใจเกี่ยวกับแบบจำลองภาพด้วยเช่นกัน

ความสำคัญของยูเอ็มแอล

ยูเอ็มแอลเป็นเครื่องมือใหม่ที่ได้รับการยอมรับเพิ่มขึ้นตลอดเวลา โดยเริ่มประยุกต์ใช้กับระบบงานมากขึ้นเพราะเป็นเครื่องมือที่มีความหลากหลายในการแสดงแบบซอฟต์แวร์เป็นโมเดลมาตรฐานที่ใช้หลักการออกแบบเชิงวัตถุ รูปแบบของภาษาที่มีสัญลักษณ์สำหรับสื่อความหมายมีกฎระเบียบที่มีความหมายต่อการเขียนโปรแกรม (Coding) ดังนั้นการใช้ยูเอ็มแอลจะต้องทราบความหมายของสัญลักษณ์ต่าง ๆ เช่น เจเนอรัลไลเซชัน (Generalization) แอสโซซิเอชัน (Association) การขึ้นต่อกัน (Dependency) คลาส และเพ็กเกจ (Package) สิ่งเหล่านี้มีความจำเป็นต่อการตีความการออกแบบก่อนนำไปประยุกต์ใช้ในระบบงานจริง (Object Management Group, 1997)

แบบจำลอง (Modeling) เป็นวิธีการวิเคราะห์ออกแบบ (Analysis and Design) อย่างหนึ่งที่เน้นการใช้งานแบบจำลองเป็นหลัก ซึ่งแบบจำลองที่สร้างขึ้นมาจะสามารถช่วยให้เข้าใจในปัญหาได้ง่ายขึ้น อีกทั้งยังสามารถนำแบบจำลองมาเป็นเครื่องมือในการสื่อสารถ่ายทอดความคิดกับบุคคลอื่น ๆ ที่เกี่ยวข้องในโครงการได้ เช่น ลูกค้า นักวิเคราะห์ระบบ นักออกแบบระบบ เป็นต้น ส่วนแบบจำลองภาพ คือการใช้สัญลักษณ์รูปภาพในการสร้างแบบจำลองของระบบ ที่จะพัฒนาเพื่อประโยชน์ที่คล้ายคลึงกันในการทำความเข้าใจกับความต้องการของลูกค้า การออกแบบระบบที่เป็นไปได้อย่างชัดเจนขึ้นและการบำรุงรักษาที่ง่ายยิ่งขึ้น (Schach, S. 2004) แบบจำลองเกิดขึ้นโดยการนำเสนอส่วนต่าง ๆ ของระบบแต่เพียงส่วนที่สำคัญโดยไม่คำนึงถึงรายละเอียดปลีกย่อยต่าง ๆ ในการพัฒนาระบบซอฟต์แวร์ที่ซับซ้อน นักพัฒนาจำเป็นต้องทำความเข้าใจกับมุมมองด้านต่าง ๆ ของระบบก่อนทำการพัฒนาจริง โดยการสร้างแบบจำลองอันเปรียบเสมือนพิมพ์เขียวที่แสดงถึงภาพรวมทั้งหมดของระบบ แบบจำลองที่สร้างขึ้นจะต้องมีความสอดคล้องกับความต้องการของผู้ใช้งานระบบเป็นสำคัญ ในส่วนของรายละเอียดต่าง ๆ จะค่อย ๆ ถูกเพิ่มเติมลงไปในตัวแบบจำลอง และในที่สุดแบบจำลองจะถูกนำไปพัฒนาขึ้นเป็นระบบจริง (วิกิตำรา, 2556)

ประวัติของยูเอ็มแอล

ประวัติของยูเอ็มแอล (History of UML) มีคร่าว ๆ ดังนี้ ในปี ค.ศ. 1970 ได้มีการเสนอวิธีเชิงวัตถุ (Object Oriented Method) ต่อมาในปี ค.ศ. 1970 ถึงปี ค.ศ. 1990 ได้เกิดยุคที่เรียกว่าสงครามเมธอด (Method Wars) ซึ่งมี Grady Booch Jim Rumbaugh และ Ivar Jacobson โดยทั้ง 3 คนเสนอแนวคิดหรือวิธีการเชิงวัตถุที่เป็นที่นิยมใช้ในช่วงกลางทศวรรษ 1990 ได้แก่

1) OOSE (Object-Oriented Software Engineering) นำเสนอโดย Ivar Jacobson
2) OMT (Object Modelling Technique) นำเสนอโดย Jim Rumbaugh

3) Booch Method นำเสนอโดย Grady Booch

แต่ทั้งสามวิธีมีปัญหาด้านการสื่อสารกัน เนื่องจากสัญลักษณ์ที่ใช้ในแต่ละวิธีการไม่เหมือนกันดังนั้นในปี 1994-1995 Booch, Rumbaugh และ Jacobson (เรียกตัวเองว่า “Three amigos”) ได้ร่วมกันทำการรวบรวมแนวคิดองค์ความรู้และเทคนิคต่าง ๆ เข้าด้วยกันที่ Rational Software ต่อมา Three Amigos ได้เสนอ Unified Modeling Language ไปยังหน่วยงาน OMG (Object Management Group) เพื่อให้เป็นภาษามาตรฐานสำหรับสร้างแบบจำลองเชิงวัตถุจากนั้น UML Version 1.1 ได้ถูกพัฒนาขึ้นในปี 1997 เพื่อเป็นมาตรฐานสำหรับสร้างแบบจำลองเชิงวัตถุ และได้มีการปรับปรุงพัฒนาในรุ่นต่อมาเรื่อย ๆ ดังนี้

ปี 1998 พัฒนา UML Version 1.2

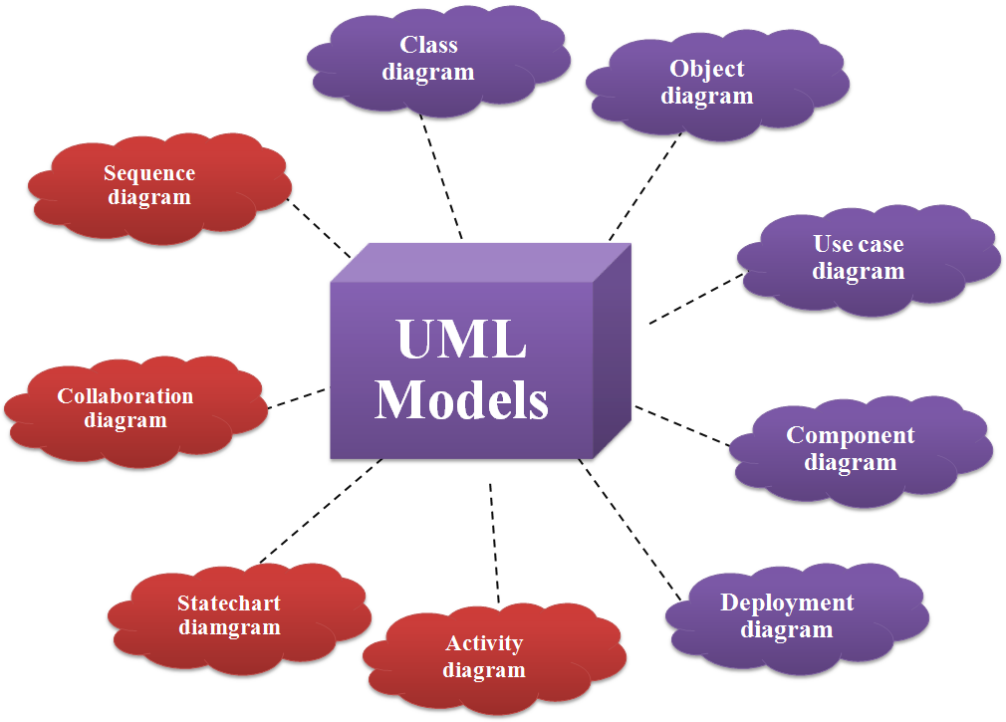
ปี 1999 พัฒนา UML Version 1.3

ปี 2000 พัฒนา UML Version 1.4

ปี 2001 พัฒนา UML Version 2.0

ซึ่งในปัจจุบันสามารถศึกษาข้อมูลเกี่ยวกับยูเอ็มแอลเพิ่มเติมได้ที่เว็บไซต์หลักคือ http://www.uml.org/

ภาพแสดงตัวแบบและแผนภาพ (Model and Diagrams) ทั้ง 9 ชนิด



ภาพที่ 4.2 แสดงตัวแบบและแผนภาพต่าง ๆ

ประโยชน์ของยูเอ็มแอล

มีเหตุผลหลายประการที่ทำให้ยูเอ็มแอลได้รับความนิยมในการพัฒนาระบบเชิงวัตถุ แต่เหตุผลหลัก ๆ สามารถสรุปเป็นข้อ ๆ ได้ดังนี้

1. ยูเอ็มแอลได้รวมข้อดีของโมเดลต่าง ๆ เอาไว้ ได้แก่

1) Data Model มาจากโมเดล OMT ของ James Rambauge ซึ่งเน้นในเรื่องข้อมูล โดยเอาแนวคิดมาจาก ER -Diagram

2) Business Model หรือ กระแสงาน (Workflow) คล้ายกับแผนภาพกระแสข้อมูลแต่ดีกว่าในเรื่องของ Sequence, Loop, Check If Condition

3) Object Model สามารถที่จะสร้างวัตถุในแบบต่าง ๆ ได้

4) Component Model เป็นโมเดลที่มีแนวคิดว่าทำอย่างไรจึงจะผลิตซอฟต์แวร์ให้เหมือนการผลิตฮาร์ดแวร์

2. ครอบคลุมทุกส่วนในวัฏจักร (Life Cycle) ของการพัฒนาระบบ (Development Life Cycle) โดยสามารถถูกใช้ตั้งแต่ขั้นตอนการหาความต้องการ (Requirements) จนถึงขั้นตอนการทดสอบติดตั้งระบบ (Test & Installation)

3. Application Domains สนับสนุนการสร้าง Application ในหลาย Domain เช่นinformation Systems, Embedded Real-Time Systems, Technical Systems, Distributed Systems, System Software

4. เป็นภาษาที่เป็นมาตรฐานเปิดของทุก ๆ ภาษา (Implementation Languages and Platforms) ไม่ผูกติดกับภาษาและแพลตฟอร์ม (Platform) ของการนำไปประยุกต์ใช้ (Implement) ง่ายต่อการทำความเข้าใจและสามารถแปลงเป็นโปรแกรมคำสั่งได้ สนับสนุนทั้งหลักการเชิงวัตถุ (Pure OO Languages) เช่น Smalltalk Java, C#. หรือหลักการผสม (Hybrid) เช่น C++ หรือภาษาที่ไม่ใช่หลักการเชิงวัตถุ (Non-OO) เช่น C

5) Development Process สามารถใช้ยูเอ็มแอลร่วมกับกรรมวิธีพัฒนาระบบใด ๆ ก็ได้ มีบริษัทชั้นนำและอุตสาหกรรมต่าง ๆ ให้การยอมรับและสนับสนุน

6) Internal Concepts มีหลักการที่ชัดเจนและมั่นคง เป็นภาษาที่มีความสมดุลในแง่ของความเรียบง่ายและความซับซ้อน

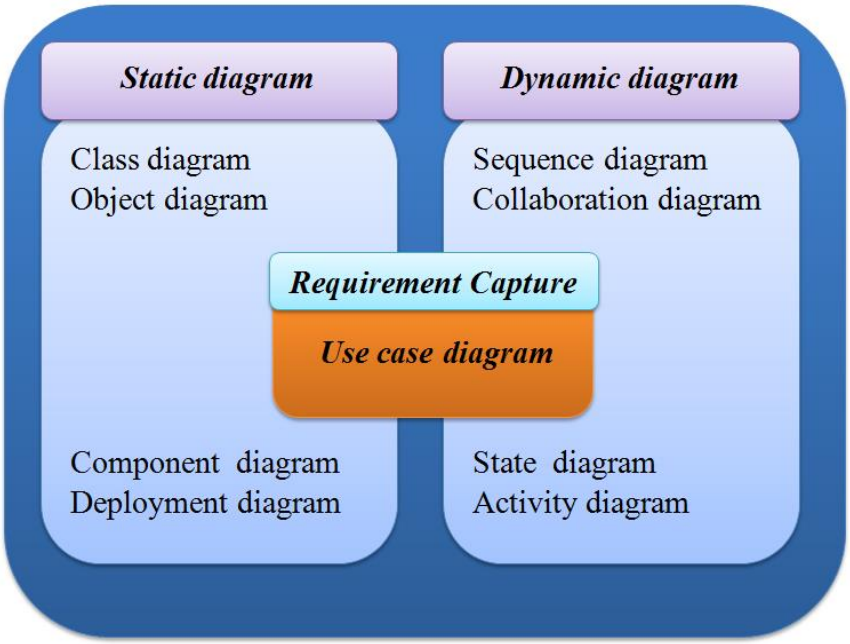
ยูเอ็มแอลเป็นเครื่องมือใหม่ที่ได้รับการยอมรับเพิ่มขึ้น และมีการประยุกต์ใช้กับระบบงานมากขึ้น และเป็นเครื่องมือที่มีความหลากหลายในการแสดงแบบซอฟต์แวร์ และเป็นโมเดลมาตรฐานที่ใช้หลักการออกแบบเชิงวัตถุ โดยรูปแบบของยูเอ็มแอลมีสัญลักษณ์สำหรับสื่อความหมาย และมีกฎระเบียบหรือข้อบังคับที่มีความหมายต่อการเขียนโปรแกรม

ประเภทของแผนภาพในยูเอ็มแอล

ยูเอ็มแอลแบ่งแผนภาพได้เป็น 2 ประเภทใหญ่ ๆ ดังนี้คือ

1) แผนภาพเชิงโครงสร้าง (Structural Diagram) หรือแผนภาพเชิงสถิต (Static Diagram) ใช้สำหรับการออกแบบโครงสร้างของระบบงาน

2) แผนภาพเชิงพฤติกรรม (Behavioral Diagram) หรือแผนภาพเชิงพลวัต (Dynamic Diagram) ใช้สำหรับการออกแบบการทำงานขององค์ประกอบต่าง ๆ ของระบบงาน ว่ามีการทำงานในตัวเองและทำงานประสานงานกันอย่างไร



ภาพที่ 4.3 แผนภาพต่าง ๆ ในยูเอ็มแอล

แผนภาพต่าง ๆ ในยูเอ็มแอล

1. แผนภาพยูสเคสและคำอธิบายยูสเคส

แบบจำลองยูสเคส (Use Case Model) แบบจำลองความต้องการของระบบที่นำเสนอความต้องการเชิงฟังก์ชัน (Functional Requirement) ของระบบโดยรวม จากมุมมองของผู้ใช้ภายนอก หรือ ระบบภายนอก ระบุพฤติกรรม หรือหน้าที่การทำงานของระบบโดยเน้นสิ่งที่ระบบต้องมี (Bahrami, A. 1999) เพื่อใช้ในการทดสอบการตรวจสอบโครงสร้างและหน้าที่การทำงานของระบบ ในยูเอ็มแอลจะแบ่งยูสเคสออกเป็น 2 ชนิดคือ

1) คำอธิบายยูสเคส (Use Case Description)

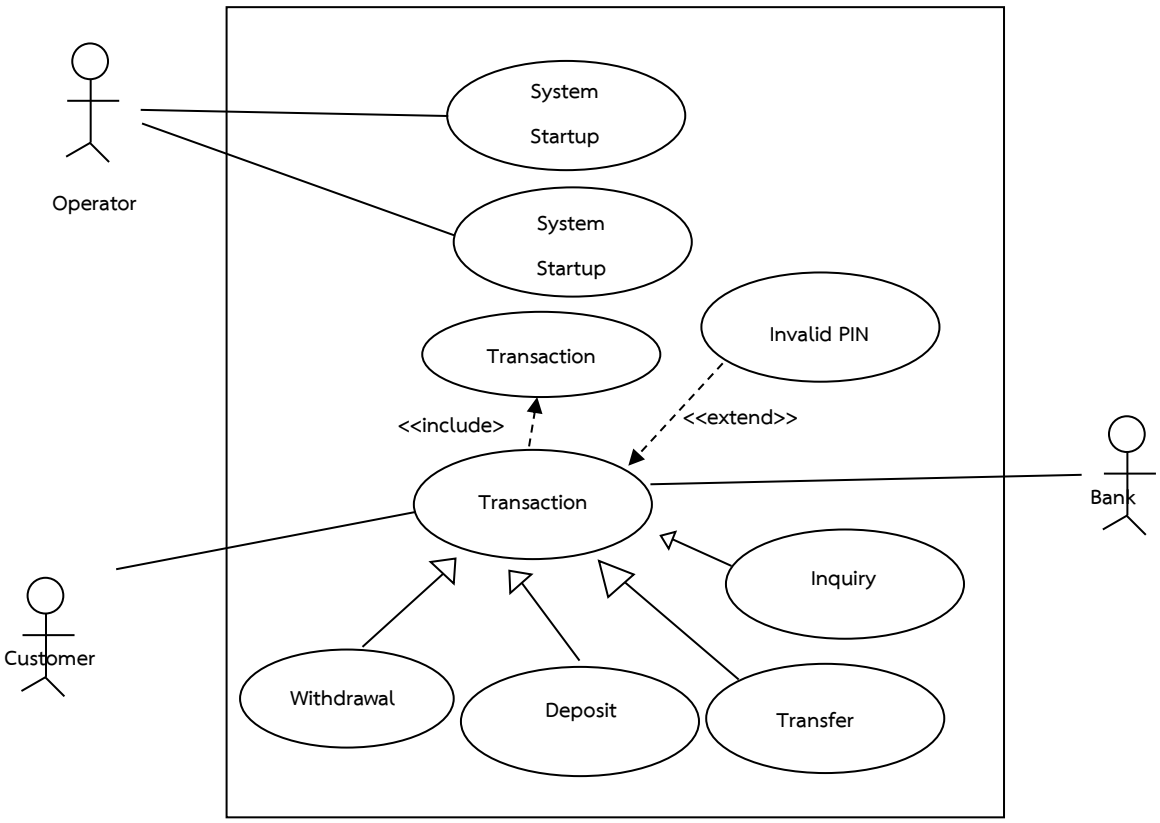
2) แผนภาพยูสเคส (Use Case Diagram)

แผนภาพยูสเคส (Use Case Diagram)

เป็นแผนภาพหนึ่งที่ใช้สื่อสารกันระหว่างนักพัฒนาระบบกับผู้ใช้ มีองค์ประกอบ 2 ส่วนคือ

1) ยูสเคส (Use Case) เป็นส่วนที่แสดงถึงขอบเขตของระบบที่กำลังสนใจ

2) แอกเตอร์ (Actor) เป็นส่วนที่อยู่นอกระบบ แต่เป็นผู้ที่กระทำอะไรบางอย่างกับระบบ และเป็นผู้ที่ได้รับผลจากระบบด้วย (Gordon Collect. 2004)



ภาพที่ 4.4 ตัวอย่างแผนภาพยูสเคสของการใช้งานตู้กดเงินอัตโนมัติ

(ที่มา: http://www.math-cs.gordon.edu/courses/cs211/ATMExample/UseCases.html)

2. แผนภาพคลาส

แผนภาพคลาส (Class Diagram) คือ แผนภาพที่ใช้แสดงคลาสและความสัมพันธ์ในแง่มุมต่าง ๆ ระหว่างคลาสเหล่านั้น ซึ่งความสัมพันธ์ที่กล่าวถึงในแผนภาพคลาสเป็นความสัมพันธ์เชิงสถิตย์ (Static Relationship) หมายถึง ความสัมพันธ์ที่มีอยู่แล้วเป็นปกติในระหว่างคลาสต่าง ๆ ไม่ใช่ความสัมพันธ์ที่เกิดขึ้นเนื่องจากกิจกรรมอื่น ๆ ซึ่งจะเรียกแบบนี้ว่าความสัมพันธ์เชิงกิจกรรม (Dynamic Relationship) สิ่งที่ปรากฏในแผนภาพคลาสนั้นจะประกอบไปด้วยกลุ่มของคลาสและกลุ่มของความสัมพันธ์ระหว่างคลาสโดยสัญลักษณ์ที่ใช้ในการแสดงคลาสนั้นจะแทนด้วยสี่เหลี่ยม ภายในสี่เหลี่ยมจะแบ่งออกเป็น 3 ส่วนย่อย โดยแต่ละส่วนนั้น (จากบนลงล่าง) จะใช้ในการแสดง ชื่อของคลาส แอตทริบิวท์ และฟังก์ชันต่าง ๆ ตามลำดับดังแสดงในภาพที่ 4.5

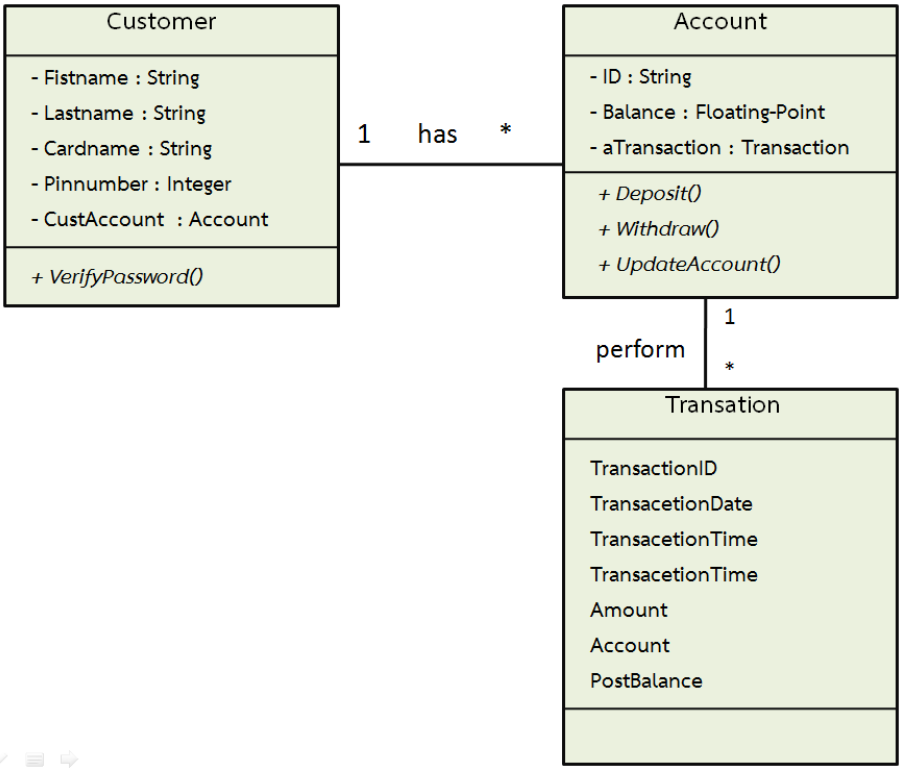
ในการเขียนสัญลักษณ์แทนคลาสสิ่งที่ต้องคำนึงถึงอีกสิ่งหนึ่งคือ ระดับการเข้าถึงเรียกสัญลักษณ์ที่ใช้แทนการเข้าถึงนี้ว่า Visibility แบ่งออกได้เป็น 3 ประเภท ได้แก่

1. Private เขียนแทนด้วยสัญลักษณ - (เครื่องหมายลบ) หมายถึง แอตทริบิวท์หรือฟังก์ชันที่ไม่สามารถมองเห็นได้จากภายนอกคลาส แต่สามารถมองเห็นหรือเข้าถึงได้จากภายในตัวของคลาสเองเท่านั้น

2. Protect เขียนแทนด้วยสัญลักษณ์ # (เครื่องหมายสี่เหลี่ยม) หมายถึงแอตทริบิวต์หรือฟังก์ชัน ที่สงวนไว้สำหรับกลไกลในการรับทอดมรดก (Inheritance) โดยเฉพาะแอตทริบิวต์หรือฟังก์ชันเหล่านี้จะเป็นของซุปเปอร์คลาส (Super class) เมื่อทำการรับทอดมรดกแล้วรับทอดมรดกที่มี Visibility แบบ Protect จะกลายไปเป็น Private รับทอดมรดก หรือ Protected ขึ้นอยู่กับภาษา Programming ที่นำไปใช้

3. Public เขียนแทนด้วยสัญลักษณ์ + (เครื่องหมายบวก) หมายถึงแอตทริบิวต์หรือฟังก์ชันที่สามารถมองเห็นได้จากภายนอก และสามารถเข้าไปเปลี่ยนค่า อ่านค่าหรือเรียกใช้งานแอตทริบิวต์หรือฟังก์ชันนั้นได้ทันทีโดยอิสระจากภายนอก (โดยทั่วไปแล้การเข้าถึงแบบ Public มักจะใช้สำหรับฟังก์ชันมากกว่าแอตทริบิวท์)

แผนภาพคลาสนั้นสามารถนำไปใช้สำหรับแสดงเอนทิตี้ต่าง ๆ ในระบบหรือภายในโดเมนหนึ่ง ๆ โดยอธิบายว่าเอนทิตี้เหล่านั้นมีความสัมพันธ์กันอย่างไร ซึ่งแผนภาพคลาสนั้นสามารถนำมาใช้อธิบาย Classes , Interfaces , Collaborations และความสัมพันธ์ระหว่างกันได้ด้วยดังแสดงใน ภาพที่ 4.5



ภาพที่ 4.5 ตัวอย่างแผนภาพคลาสของระบบงานธนาคาร

สิ่งที่เป็นผลลัพธ์ที่ได้จากขั้นตอนการออกแบบระบบโดยจะสามารถใช้อธิบายถึงมุมมองเชิงสถิตย์(Static View หรือ Static Structure) ของระบบที่กำลังจะพัฒนานั้น ในแง่ของรายละเอียดของการแก้ปัญหา (Software Solution) ของระบบมากกว่าจะอธิบายสภาพหรือลักษณะปัญหาที่พบของระบบดังกล่าว (Problem Domain) กล่าวคือจะอธิบายว่าระบบนั้นถูกแก้ปัญหาอย่างไรมีรายละเอียดและหนทางแก้ปัญหาอย่างไรบ้างซึ่งสิ่งดังกล่าวก็ล้วนเกิดจากการกำหนดขึ้นมาโดยผู้พัฒนาระบบนั้นเองซึ่งจะเรียกว่าข้อกำหนดขององค์ประกอบภายในระบบหรือเรียกว่า Specification of Software Class

1) แสดงรายละเอียดของคลาส และความสัมพันธ์ระหว่างคลาส ในมุมมองแบบ Logical View

2) องค์ประกอบของแผนภาพคลาสมีดังนี้

3) คลาสโครงสร้างของคลาส และพฤติกรรมของคลาส

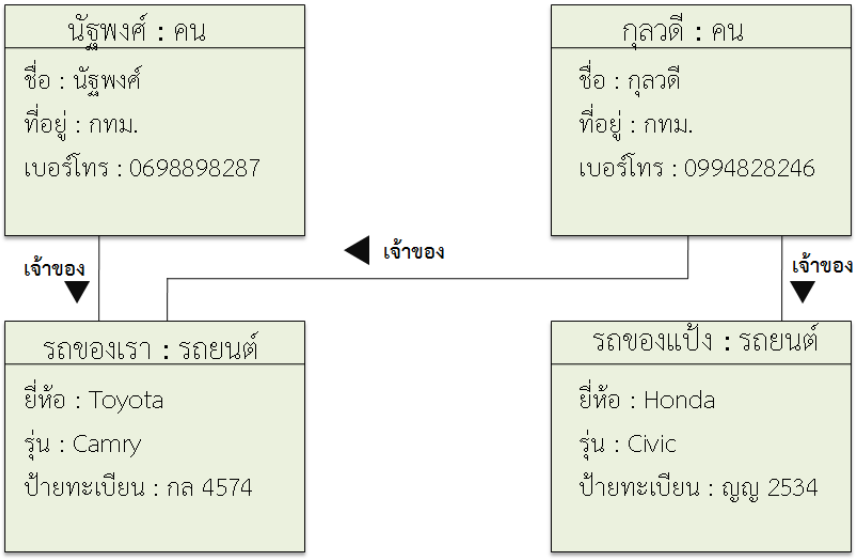
4) ตัวบ่งชี้ Multiplicity และ Navigation

5) ชื่อของโรว์ (Role)

6) ความสัมพันธ์แบบ Association, Aggregation, Dependency, และ Inheritance

3. แผนภาพวัตถุ

แผนภาพวัตถุ (Object Diagram) เป็นแผนภาพที่แสดงวัตถุที่ถูกสร้างขึ้นจากคลาส และจำลองความสัมพันธ์ระหว่างวัตถุที่ได้ออกแบบความสัมพันธ์ไว้ในแผนภาพคลาสนั้นสามารถเกิดขึ้นได้จริงในระบบงานใหม่หรือไม่ โดยสัญลักษณ์ที่ใช้จะมีลักษณะเช่นเดียวกับแผนภาพคลาสต่างกันที่ชื่อของแผนภาพวัตถุจะมีขีดเส้นใต้ดังแสดงในภาพที่ 4.6

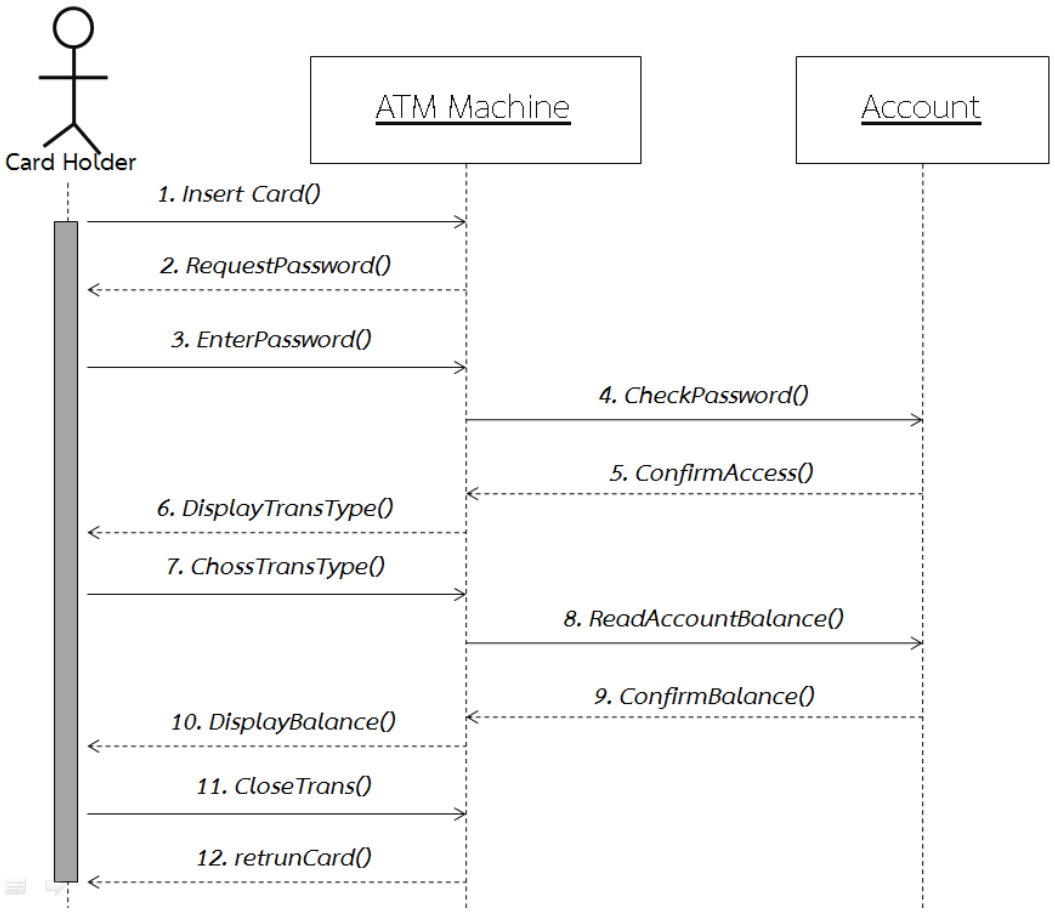


ภาพที่ 4.6 ตัวอย่างแผนภาพวัตถุ

จากภาพที่ 4.6 เป็นตัวอย่างของแผนภาพวัตถุสามารถจะเห็นว่าประกอบไปด้วย You และ Me ซึ่งเป็นวัตถุของคลาส Person ในขณะที่ Ours และ Mine เป็นวัตถุของคลาส Car จากแผนภาพนี้สิ่งที่ใช้แสดงความสัมพันธ์ระหว่างแต่ละวัตถุนั้นจะถูกเรียกว่าจุดเชื่อม (Links)

4. แผนภาพลำดับ

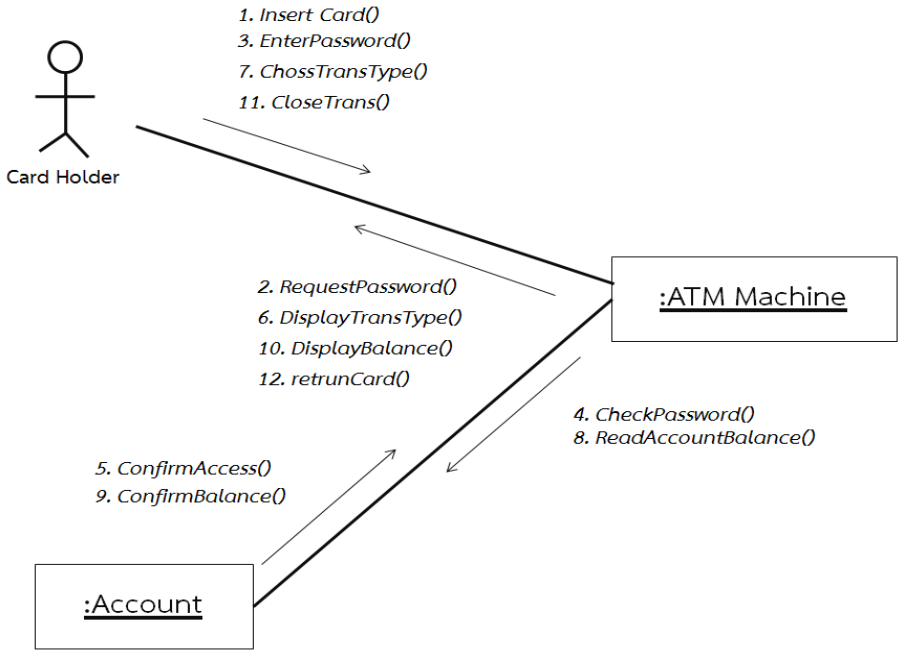
แผนภาพลำดับ (Sequence Diagram) แผนภาพลำดับเป็นแผนภาพที่ใช้แสดงการทำงานระหว่างวัตถุต่าง ๆ เมื่อเกิดการส่งข่าวสารระหว่างกัน



ภาพที่ 4.7 ตัวอย่างแผนภาพลำดับการสอบถามยอดบัญชีจากตู้กดเงินอัตโนมัติ (ATM)

5. แผนภาพประสานงาน

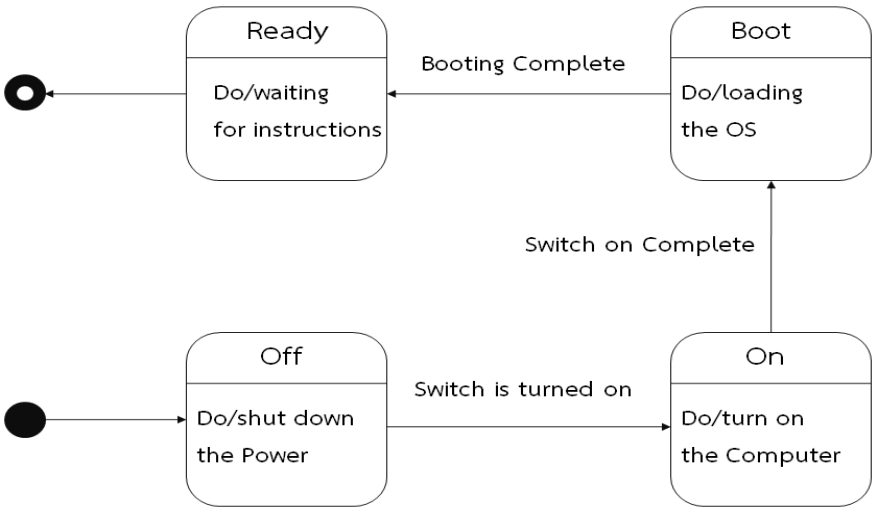
แผนภาพประสานงาน (Collaboration diagram) เป็นไดอะแกรมแบบเดียวกันกับ Sequence Diagram โดย Sequence Diagram จะเป็นไดอะแกรมที่แสดงถึงการแลกเปลี่ยนข่าวสาร แต่ Collaboration Diagram จะนำเสนอแผนภาพการทำงานร่วมกันระหว่างออบเจ็กต์เป็นสำคัญ นอกจากนี้ก็ยังแสดงลำดับการทำงานก่อน และหลังด้วยซึ่งจะเห็นได้ว่า Collaboration Diagram จะแสดงให้เห็นภาพโครงสร้างระบบมากกว่าการเน้นเพียงข่าวสารที่สื่อสารกัน หากต้องการแผนภาพที่มุ่งเน้นด้านเวลาเป็นสำคัญและแสดงลำดับก่อนหลัง ให้เลือกใช้แผนภาพลำดับ แต่หากต้องการแผนภาพที่ให้ความสัมพันธ์ภายในออบเจ็กต์ ก็ให้เลือกใช้แผนภาพประสานงาน ซึ่งแผนภาพประสานงานเป็นแผนภาพชนิดเดียวกับแผนภาพลำดับ โดยแผนภาพลำดับจะเป็นแผนภาพที่แสดงถึงการสื่อสาร แต่แผนภาพประสานงานจะนำเสนอการทำงานร่วมกันระหว่างวัตถุเป็นหลัก แต่ก็สามารถแสดงถึงลำดับก่อนหลังด้วย ดังแสดงในภาพที่ 4.8



ภาพที่ 4.8 ตัวอย่างแผนภาพประสานงานการสอบถามยอดบัญชีจากตู้กดเงินอัตโนมัติ ATM

6. แผนภาพสถานะ

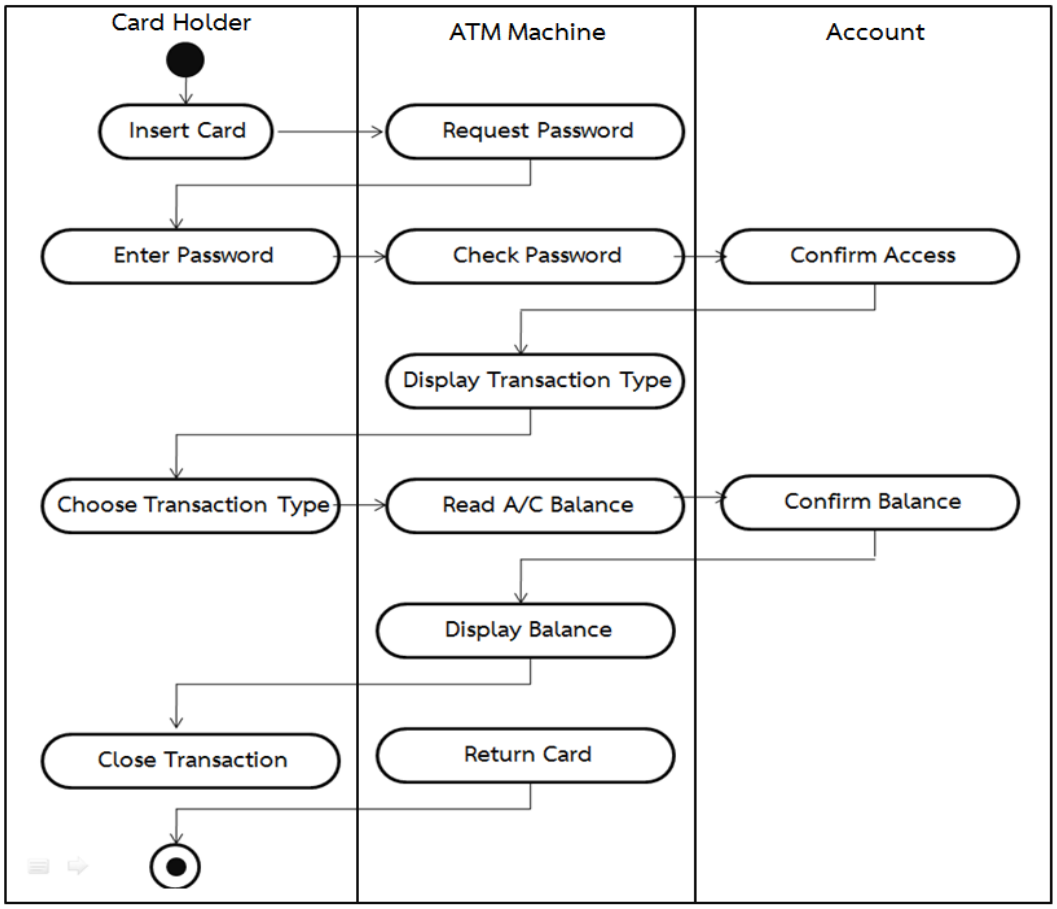
แผนภาพสถานะ (State Diagram หรือ State Transition Diagram) เป็นแผนภาพใช้แสดงสถานะ (state) ต่างๆ ของวัตถุที่เป็นได้ในระหว่างช่วงชีวิต(Life Time) ในการตอบสนองต่อเหตุการณ์ที่เกิดขึ้น โดยทั่วไปแล้ว State Diagram จะไม่ถูกใช้กับคลาสทั้งหมด แต่จะใช้อธิบายเฉพาะคลาสที่มีความซับซ้อนสูงเท่านั้น เพื่อที่จะช่วยให้การออกแบบขั้นตอนวิธี (Algorithm) ง่ายขึ้น โดยแผนภาพสถานะ เป็นการแสดงวงจรชีวิตของวัตถุ ระบบย่อยต่าง ๆ และระบบโดยรวม โดยบ่งบอกว่าเหตุการณ์ต่าง ๆ จะส่งผลกระทบให้เกิดอะไรขึ้นบ้าง ซึ่งอาจจะมีจุดเริ่มต้นและจุดสิ้นสุดได้หลาย ๆ จุด ดังตัวอย่างในภาพที่ 4.9



ภาพที่ 4.9 ตัวอย่างแผนภาพสถานะแสดงการเปิดใช้งานคอมพิวเตอร์

7. แผนภาพกิจกรรม

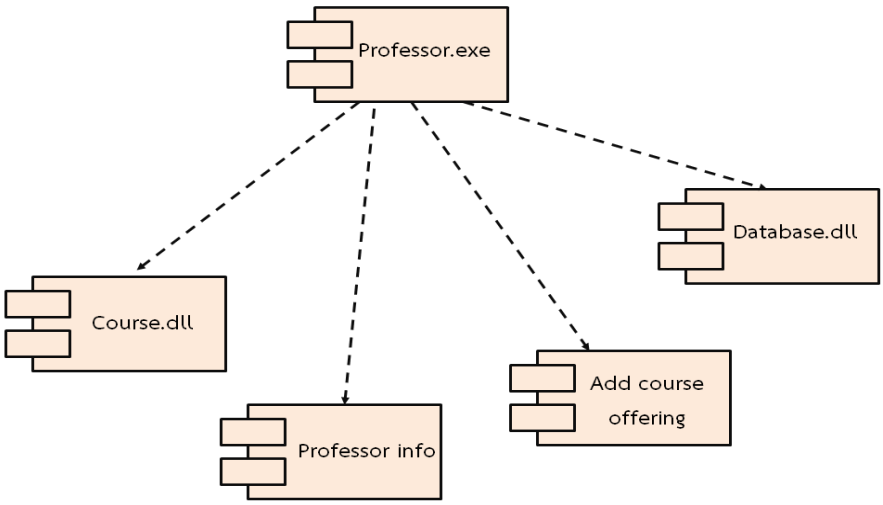
แผนภาพกิจกรรม (Activity Diagram) เป็นการแสดงถึงขั้นตอน และจุดที่ต้องมีการตัดสินใจที่เกิดภายในวัตถุ หรือภายในกระบวนการทำงาน



ภาพที่ 4.10 ตัวอย่างแผนภาพกิจกรรมของการสอบถามยอดบัญชีจากตู้กดเงินอัตโนมัติ

8. แผนภาพคอมโพเน้นท์

แผนภาพคอมโพเน้นท์ (Component Diagram) เป็นแผนภาพที่แสดงโครงสร้างและความสัมพันธ์ระหว่างส่วนประกอบ (Components) ต่าง ๆ ของ ซอฟต์แวร์ซึ่งองค์ประกอบดังกล่าวอาจเป็น รหัสต้นฉบับ (Source Code) โปรแกรมประมวลผล (Executable Program) ไบนารี (Binary) รวมถึงข้อความ (Text) และส่วนที่ติดต่อผู้ใช้งาน (User Interface)

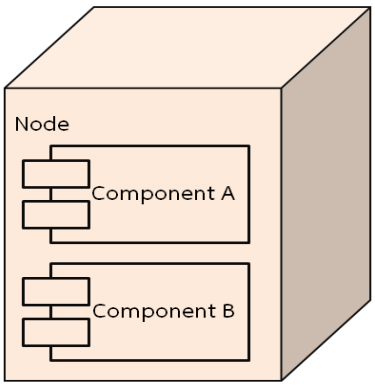


ภาพที่ 4.11 ตัวอย่างแผนภาพส่วนประกอบของระบบการลงทะเบียน

แผนภาพส่วนประกอบเป็นแผนภาพที่ใช้ในการแบ่งระบบงาน (System) ขนาดใหญ่ออกเป็นระบบย่อยๆ (Subsystems) ซึ่งแต่ละระบบย่อยก็จะมีส่วนประกอบ (Component) ต่าง ๆ ประกอบอยู่ภายใน การแบ่งระบบออกเป็นระบบย่อยเป็นที่ยอมรับโดยทั่วไปแล้วว่าสามารถทำให้การพัฒนาระบบอย่างมีประสิทธิภาพ และสนับสนุนหลักการพัฒนาระบบงานแบบเป็นทีมงานที่สามารถแบ่งส่วนย่อยต่าง ๆ ให้แต่ละส่วนงานย่อยไปรับผิดชอบได้

9. แผนภาพดีพลอยด์เม้นท์

แผนภาพดีพลอยด์เม้นท์ (Deployment Diagram) เป็นแผนภาพที่ใช้แสดงสถาปัตยกรรมของระบบในลักษณะที่เป็นสถาปัตยกรรมเชิงกายภาพ (Physical Architecture) เพื่อแสดงให้เห็นว่าระบบที่พัฒนามีคอมพิวเตอร์และอุปกรณ์อะไรบ้างที่จำเป็นต้องใช้ในระบบ



ภาพที่ 4.12 แผนภาพแจกจ่าย

สำหรับในยูเอ็มแอลรุ่น 2.0 ได้มีการเพิ่มเติมแผนภาพและปรับเปลี่ยนแผนภาพทั้งหมดเป็น 13 แผนภาพโดย สามารถแบ่งออกเป็นกลุ่มต่าง ๆ ได้ดังนี้

แผนภาพประเภทโครงสร้าง

1) แผนภาพคลาส (Class diagram)
2) แผนภาพคอมโพเน้นท์ (Component diagram)
3) แผนภาพ Composite structure diagram
4) แผนภาพดีพลอยด์เม้นท์ (Deployment diagram)
5) แผนภาพวัตถุ (Object diagram)
6) แผนภาพแพ็กเก็จ (Package diagram)

แผนภาพประเภทพฤติกรรม

1) แผนภาพกิจกรรม (Activity diagram)
2) แผนภาพสถานะ (State Machine diagram)
3) แผนภาพยูสเคส (Use case diagram)

แผนภาพประเภทการโต้ตอบ

1) แผนภาพการสื่อสาร (Communication diagram)
2) แผนภาพปฏิสัมพันธ์ (Interaction overview diagram ,UML 2.0)
3) แผนภาพลำดับ (Sequence diagram)
4) แผนภาพช่วงเวลา (UML Timing diagram ,UML 2.0)

ในยูเอ็มแอลรุ่นที่ 2 (UML 2.0) ได้มีการกำหนดแผนภาพเพิ่มจากเดิม เป็น 14 แผนภาพดังนี้ (All supported by Enterprise Architect)

1. Package diagrams
2. Class or Structural Diagrams
3. Object Diagrams
4. Composite Structure
5. Component Diagrams
6. Deployment diagrams
7. Use Case Diagrams
8. Activity diagrams
9. State Machine diagrams
10. Communication diagrams

11. Sequence diagrams
12. Timing diagrams
13. Interaction Overview diagrams
14. Profile diagrams

มุมมองต่าง ๆ ของยูเอ็มแอล (UML Views)

ยูเอ็มแอลสามารถมองได้แบบต่าง ๆ ได้ดังตารางที่ 4.2

ตารางที่ 4.2 ตารางแสดงมุมมองเชิงโครงสร้างของยูเอ็มแอล

Major Area	มุมมอง (View)	แผนภาพ (Diagrams)
โครงสร้าง (Structural)	Static View	Class Diagram
	Use Case View	Use Case Diagram
	Implementation View	Component Diagram
	Deployment View	Deployment Diagram

ตารางที่ 4.3 มุมมองเชิงไดนามิคของยูเอ็มแอล

ไดนามิค (dynamic)	State Machine View	State Chart Diagram
	Activity View	Activity Diagram
	Interaction View	Sequence Diagram
		Collaboration Diagram

ตารางที่ 4.4 มุมมองเชิงการจัดการของยูเอ็มแอล

การจัดการโมเดล (model management)	Model Management View	Class Diagram (Package, Subsystem)

5 มุมมองหลักของ UML

1) Use-case View เป็นมุมมองที่เน้นหน้าที่การทำงานของระบบซอฟต์แวร์ โดยพิจารณาจากมุมมองของผู้ใช้ภายนอกหรือระบบภายนอกโดยแผนภาพที่ใช้ในมุมมองนี้คือแผนภาพยูสเคส (Use case diagram)

2) Logical View เป็นมุมมองที่เน้นหน้าที่การทำงานของระบบว่ามีโครงสร้างอย่างไรโดยมองในรูปของโครงสร้างสถิตและโครงสร้างแบบไดนามิก (Static Structure and Dynamic Behavior) โดยแผนภาพที่ใช้สำหรับมุมมองนี้คือ แผนภาพคลาส (Class Diagram), แผนภาพวัตถุ (Object Diagram), แผนภาพสถานะ (State Diagram), แผนภาพลำดับ (Sequence Diagram), แผนภาพประสาน (Collaboration Diagram), Activity Diagram

3) Component View เป็นมุมมองที่เน้นองค์ประกอบย่อยในการ Implement ที่ประกอบเป็นระบบและ Dependency ระหว่างองค์ประกอบเหล่านั้นโดยแผนภาพที่ใช้สำหรับมุมมองนี้คือแผนภาพคอมโพเน้นท์ (Component Diagram)

4) Concurrency View เป็นมุมมองที่เน้นการแบ่งแยก Process และ Processors โดยพิจารณาทั้ง Communication และ Synchronization โดยแผนภาพที่ใช้สำหรับมุมมองนี้คือ

4.1) Dynamic Diagrams (State, Sequence, Collaboration Activity)

4.2) Implementation Diagrams (Component และ Deployment)

5) Deployment View เป็นมุมมองที่เน้นโครงสร้างทางกายภาพเกี่ยวกับการติดตั้งและใช้งานระบบโดยแผนภาพที่ใช้สำหรับมุมมองนี้คือแผนภาพแจกจ่าย

ประโยชน์ของยูเอ็มแอล (UML)

ยูเอ็มแอลนั้นมีประโยชน์ในหลายเรื่องดังนี้

1) Development Life Cycle - สนับสนุนทั้ง Life Cycle ของการพัฒนาระบบ สามารถถูกใช้ตั้งแต่ขั้นตอนการหาความต้องการ (Requirements) จนถึงขั้นตอนการทดสอบติดตั้งระบบ (Test & Installation)

2) Application Domains - สนับสนุนการสร้าง Application ในหลาย Domain เช่น Information Systems, Embedded Real-Time Systems, Technical Systems, Distributed Systems, System Software

3) Implementation Languages and Platforms - ไม่ผูกติดกับภาษาและแพลตฟอร์ม (Platform) ของการ implement สนับสนุนทั้ง Pure OO Languages เช่น Smalltalk Java, C#. หรือ Hybrid เช่น C++ หรือ non-OO เช่น c

4) Development Process – สามารถใช้ UML ร่วมกับกรรมวิธีพัฒนาระบบใดๆก็ได้ (Prefer UP)

5) Internal Concepts – มีหลักการที่ชัด มั่นคง

การใช้งานยูเอ็มแอลในวงจรการพัฒนาระบบเชิงวัตถุ

การใช้งานยูเอ็มแอลในขั้นตอนการพัฒนาระบบเชิงวัตถุ (Object-Oriented System Analysis and Design) จะถูกแบ่งออกตามขั้นตอนการทำงานต่าง ๆ ดังนี้

1. การวิเคราะห์ความต้องการ (Requirement Analysis)

วัตถุประสงค์ของเฟสแรกนี้ คือ การทำความเข้าใจกับขอบเขตของปัญหา (Problem Domain) การกำหนดขอบเขตของการพัฒนาและฟังก์ชัน (Function) การทำงานต่าง ๆ ของระบบที่พัฒนา

สิ่งที่ผู้พัฒนาต้องทำในขั้นตอนนี้ คือ การนำเอาความต้องการแบบเชิงฟังก์ชัน (Functional Requirements) ซึ่งบ่งบอกถึงความสามารถที่ผู้ใช้สามารถเรียกใช้จากระบบ มาแปลงเป็นโมเดลความต้องการของผู้ใช้งาน ซึ่งจะใช้แผนภาพยูสเคส (Use Case Diagram) ที่ขาดไม่ได้ในยูสเคส คือคำบรรยายยูสเคส (Use Case Description) ซึ่งเป็นรายละเอียดของแต่ละฟังก์ชันว่าเริ่มต้นอย่างไร มีการดำเนินเหตุการณ์เกิดขึ้นอย่างไรและสิ้นสุดลงอย่างไร เรียกรวม ๆ ว่า ลำดับการเกิดเหตุการณ์ (Flow of Event) รวมถึงเหตุการณ์ยกเว้น (Exception Flow of Event) ที่อาจเกิดขึ้นระหว่างการปฏิบัติฟังก์ชันดังกล่าวของระบบ ก็ต้องถูกบันทึกด้วยเช่นกัน (Bennett, S. Mcrobb, S. and Farmer, R. 2006)

2. การวิเคราะห์ระบบ (System Analysis)

ในเฟสที่สองนี้เป็นการบรรยายถึงโครงสร้างและพฤติกรรมของระบบที่กำลังพัฒนา ซึ่งกิจกรรมสำคัญต่างๆที่ผู้พัฒนาต้องกระทำในเฟสนี้ ได้แก่

2.1) การสร้างแผนภาพคลาส (Class Diagram) เพื่อศึกษาโครงสร้างหรือส่วนที่เป็นโครงสร้างของระบบ โดยการ

2.1.1 หาคลาสทั้งหมดในระบบโดยใช้เทคนิค Heuristic Mapping คือ ให้คำนามที่พบในคำบรรยายยูสเคส (ระยะแรก) แปลงเป็นคลาส

คำกิริยา จะถูกแปลงเป็น Operation และ

คำวิเศษณ์ จะถูกแปลงเป็น Attribute

ซึ่งคำนามทุกคำอาจไม่ถูกแปลงเป็นคลาสเสมอไป เช่น คำนามบางคำเขียนต่างกันแต่หมายถึงสิ่งเดียวกัน ซึ่งขั้นตอนนี้ประสบการณ์ถือเป็นสิ่งสำคัญที่ช่วยให้ผู้พัฒนาค้นพบคลาสเป้าหมายและรายละเอียดของคลาสได้อย่างถูกต้อง

2.1.2 ค้นหาความสัมพันธ์ (Relationships) ระหว่างคลาส

อันได้แก่ การขึ้นต่อกัน (Dependency) การถ่ายทอดคุณสมบัติ (Inheritance) ความสัมพันธ์แบบเชื่อมโยง (Association) ทั้งแบบสัมพันธ์แบบองค์ประกอบร่วม (Composition) และสัมพันธ์แบบมีส่วนร่วม (Aggregation) ด้วยเช่นกัน

2.2) การสร้างแผนภาพลำดับ (Sequence Diagram) เพื่อศึกษาพฤติกรรม (Behavior) หรือส่วนที่มีความเป็นพลวัต (Dynamic) ของระบบ โดยการสร้างแผนภาพลำดับสำหรับแต่ละยูสเคส ในแผนภาพยูสเคสจากเฟสแรก นอกจากนี้การสร้างแผนภาพลำดับยังช่วยให้พบ Operation เพิ่มเติมด้วยวิธีการพัฒนาซอฟต์แวร์เชิงวัตถุ เป็นไปในลักษณะของการวนรอบทำซ้ำและทำเพิ่ม (Interactive and Incremental) ขึ้นเรื่อย ๆ จนสมบูรณ์

ซึ่งลักษณะดังกล่าวเริ่มปรากฏในขั้นตอนนี้ คือ แทนที่ต้องทำการวิเคราะห์ ทุกยูสเคส จากแผนภาพยูสเคส (Use Case Diagram) ในครั้งเดียว ซึ่งหากเป็นระบบที่ซับซ้อนและมีขนาดใหญ่ก็อาจประกอบไปด้วยยูสเคส นับร้อย ดังนั้นในขั้นตอนนี้ ผู้พัฒนาสามารถเลือกเพียงหนึ่งยูสเคส มาทำการวิเคราะห์ หาคลาสและความสัมพันธ์ระหว่างคลาส รวมถึงลักษณะพลวัตของระบบ และเมื่อวงจรการพัฒนากลับมาที่ขั้นตอนนี้อีกครั้ง ก็เป็นการวิเคราะห์ออกแบบยูสเคสต่อไป สรุปขั้นตอนของการวิเคราะห์เป็นการรูปแบบ (Model) ของปัญหา ซึ่งผลที่ได้คือเอกสารที่บันทึกรูปแบบการวิเคราะห์ระบบ (Analysis Model Document) ซึ่งประกอบไปด้วยแผนภาพคลาสและแผนภาพลำดับ นอกจากนี้อาจมีการใช้สัญลักษณ์ แพ็กเกจ (Package) หรือแผนภาพอื่น ๆ เช่น แผนภาพประสานงาน แผนภาพการเปลี่ยนสถานะ และแผนภาพกิจกรรมมาช่วยในการสร้างรูปแบบด้วยเช่นกัน (soundmk. 2562)

3. การออกแบบ (Design)

ในเฟสการออกแบบระบบ ผู้พัฒนาจะทำการกำหนดรายละเอียดเชิงเทคนิคของระบบให้พร้อม เพื่อนำไปใช้ (Implement) จริงในเฟสถัดไป

อาจกล่าวได้ว่า ขั้นตอนนี้เป็นการค้นหาวิธีการแก้ปัญหาภายหลังจากการทำความเข้าใจปัญหา สิ่งที่ต้องเตรียมสำหรับใช้ในขั้นตอนนี้ คือ รูปแบบการวิเคราะห์จากขั้นตอนที่สอง ซึ่งอาจรวมถึงความต้องการแบบไม่เชิงฟังก์ชัน (Non-Functional Requirement) ก็จะถูกนำมาพิจารณาในการออกแบบในขั้นตอนนี้ด้วยเช่นกัน กิจกรรมในการขั้นตอนการออกแบบระบบมีดังต่อไปนี้

3.1) การเพิ่มเติม คลาส หรือ แพ็กเกจ ลงไปภายในรูปแบบ

การวิเคราะห์ระบบ (ขั้นตอนที่สอง) เช่น การเพิ่มแพ็กเกจที่เกี่ยวข้องกับฐานข้อมูล การติดต่อสื่อสารโดย แพ็กเกจ หรือ คลาส เหล่านี้จะทำงานร่วมกันกับแพ็กเกจ เดิมที่มีอยู่ในรูปแบบการวิเคราะห์ระบบ นอกจากนี้ยังรวมถึงการแก้ไขปรับปรุงเพิ่มเติมแอตทริบิวต์ (Attribute) หรือฟังก์ชัน (Function) ของคลาสต่าง ๆ ในรูปแบบการวิเคราะห์ระบบด้วยเช่นกัน

3.2) จัดหาลำดับชั้นของคลาส (Class Hierarchy) หรือคอมโพเน้นท์ (Component) จากที่อื่นเพื่อนำมาใช้อีกครั้ง เพื่อช่วยลดเวลาในการพัฒนาระบบ

3.3) กำหนดรายละเอียดส่วนติดต่อกับผู้ใช้ของระบบ (User Interface Design)

3.4) หาวิธีจัดการกับข้อผิดพลาดที่อาจจะเกิดขึ้นในการใช้งานระบบ (Exception Handling)

3.5) ออกแบบสถาปัตยกรรมระบบ (System Architecture Design) เพื่อทำการพิจารณาที่อยู่ หรือตำแหน่งการติดตั้งของส่วนประกอบ (Component) หรือแพ็กเกจ (Package) ต่าง ๆ พิจารณาว่าคลาสใดควรอยู่ในส่วนประกอบ (Component) หรือไฟล์ใดควรติดตั้งไว้ที่ระบบคอมพิวเตอร์ส่วนใด นั่นคือการสร้างแผนภาพส่วนประกอบ (Component Diagram) และแผนภาพการปรับใช้ (Deployment Diagram) สำหรับรูปแบบของสถาปัตยกรรมที่ใช้กันโดยทั่วไป คือ สถาปัตยกรรมแบบกระจาย (Distributed System) ซึ่งได้แก่ 2-เทียร์ (2-Tier) 3-เทียร์ (3- Tier) และ มัลติเทียร์ (Multi-Tier System)

3.6) ออกแบบส่วนที่เป็นความต้องไม่เชิงฟังก์ชัน (Nonfunctional Requirement) เช่น กลุ่มของผู้ใช้งานมีความต้องการใช้งานระบบใหม่ร่วมกับฐานข้อมูลหรือระบบเดิมที่มีอยู่ (Legacy System Integration)

3.7) นอกจากนี้ การออกแบบยังรวมถึงการตัดสินใจเลือกใช้เทคโนโลยีต่าง ๆ ที่มีอยู่ให้เหมาะสมกับความต้องการและงบประมาณของผู้ใช้งานด้วยเช่นกัน

กล่าวโดยสรุปแผนภาพยูเอ็มแอล (UML Diagram) ที่ถูกสร้างในขั้นตอนนี้ได้แก่ แผนภาพคอมโพเน้นท์ (Component Diagram) และแผนภาพดีพลอยด์เม้นท์ (Deployment Diagram) ในส่วนของแผนภาพคลาส (Class Diagram) และแผนภาพลำดับ (Sequence Diagram) จะถูกเพิ่มรายละเอียดเชิงเทคนิค เรียกผลลัพธ์รวมของขั้นตอนนี้ว่า โมเดลการออกแบบระบบ (Design Model) ซึ่งประกอบไปด้วยแผนภาพต่าง ๆ ข้างต้น รวมถึงข้อกำหนดด้านอื่น ๆ ซึ่งระบุถึงรายละเอียดของความต้องการแบบ Nonfunctional เทคนิควิธีการแก้ปัญหา และเอกสารการออกแบบ (UML Design Document) นี้จะถูกส่งต่อไปให้โปรแกรมเมอร์เพื่อนำไปพัฒนาในขั้นตอนถัดไป (Bruegge, B. and Dutdit, A. 2000)

4. การสร้างโปรแกรมระบบ (Construction)

วัตถุประสงค์หลักของเฟสนี้ คือ การแปลงผลที่ได้จากขั้นตอนการออกแบบไปเป็นรหัสคำสั่ง (Code) กล่าวคือ นักวิเคราะห์ออกแบบระบบจะต้องทำการสร้างรูปแบบ (Model) ระบบที่สมบูรณ์อันเป็นข้อมูลสำคัญสำหรับโปรแกรมเมอร์

5. การทดสอบระบบ (Testing)

ในขั้นตอนนี้ ผู้พัฒนาจะต้องทำการค้นหาข้อผิดพลาด (Error) ภายในระบบที่กำลังพัฒนาซึ่งโดยปกติการทดสอบระบบจะอ้างอิงผลการวิเคราะห์ความต้องการผู้ใช้งานระบบ(ขั้นตอนแรก)เป็นหลักว่าเป็นไปตามความต้องการของผู้ใช้งานจริงอย่างครบถ้วนหรือไม่ทั้งด้านความต้องการแบบเชิงฟังก์ชันและไม่เชิงฟังก์ชัน

สรุป

ในบทนี้ได้อธิบายถึงความหมายประวัติความเป็นมา หลักการ มุมมอง และประโยชน์ของแผนภาพต่าง ๆ ที่มีอยู่ในยูเอ็มแอล และ ขั้นตอนการพัฒนาระบบเชิงวัตถุ ไปแล้ว ในบทต่อ ๆ ไปจะได้เจาะลึกลงใน

รายละเอียดแต่ละแผนภาพของยูเอ็มแอลและจะอธิบายถึงหลักการและวิธีการนำแผนภาพแต่ละแบบมาประยุกต์ใช้ทั้งในขั้นตอนของการวิเคราะห์ระบบและขั้นตอนการออกแบบระบบตามหลักการเชิงวัตถุต่อไป

ชื่อ-นามสกุล	รหัส	สาขาวิชา	รุ่น/หมู่	คะแนน	ลายเซ็นต์อาจารย์

แบบฝึกหัดท้ายบทที่ 4

1. จงอธิบายความหมายของแบบจำลองมาพอสังเขป

.....

.....

.....

.....

2. จงเปรียบเทียบแบบจำลองที่ใช้ในหลักการเชิงวัตถุและหลักการเชิงโครงสร้าง

.....

.....

.....

.....

3. ข้อดีของยูเอ็มแอลมีอะไรบ้าง จงอธิบายมาพอสังเขป

.....

.....

.....

.....

4. จงอธิบายประโยชน์ของยูเอ็มแอลในการพัฒนาระบบเชิงวัตถุ

.....

.....

.....

.....

5. จงอธิบายการใช้แผนภาพแต่ละอย่างของยูเอ็มแอล

.....

.....

.....

.....

6. จงอธิบายขั้นตอนของการพัฒนาระบบเชิงวัตถุ

.....

.....

.....

.....

เอกสารอ้างอิง

พนิดา พานิชกุล. (2548). *Object-Oriented ฉบับพื้นฐาน.* กรุงเทพฯ: เคพีทีคอมพแอนดคอนซัลทจำกัด.

ชาคริต กุลไกรศรี. (2556). *UML คืออะไร.* สืบค้น 29 มีนาคม 2563, จาก https://msit5.wordpress.com/2013/09/04/uml-คืออะไร/

วิกิตำรา. (2556). *กระบวนการพัฒนาซอฟต์แวร์.* สืบค้น 29 มีนาคม 2563, จาก https://th.wikibooks.org/w/index.php?title=กระบวนการพัฒนาซอฟต์แวร์&oldid=23201

Bahrami, A. (1999). *Object Oriented Systems Development.* Singapore: McGraw-Hill.

Bennett, S. Mcrobb, S. and Farmer, R. (2006). *Object-Oriented Systems Analysis And Design Using UML.* 3rd Ed. Berkshire, UK. McGraw-hill.

Bruegge, B. and Dutdit, A. (2000). *Object-Oriented Software Engineering: Conquering Complex and Changing Systems.* Upper Saddle River, N.J.: Prentice Hall.

Denis, A. Wixom, B. and Tegarden, D. (2005). *Systems Analysis and Design with UML Version 2.0.2nd ed.* USA: Wiley.

Gordon Collect. (2004). *Use Cases for Example ATM System.* Retrieved 14 January 2020, from http://www.math-cs.gordon.edu/courses/cs211/ATMExample/UseCases.html

Object Management Group. (1997). *WHAT IS UML.* Retrieved 18 September 2020, from https://www.uml.org/

Schach, S. (2004). *An Introduction to Object-oriented systems analysis and design with UML and the unified process.* New York : McGraw-hill.

soundmk. (2562). *การวิเคราะห์และออกแบบระบบเชิงออปเจ็คต์.* สืบค้น 21 มกราคม 2563, จาก https://soundmk.com/ooad