รายงานการฝึกอบรม

Object-Oriented Software Development with UML

จัดโดย

เขตอุตสาหกรรมซอฟต์แวร์ประเทศไทย (Software Park Thailand)

ณ อาคาร Software Park นนทบุรี

วันที่ 18-22 มิถุนายน 2555

ผู้จัดทำ

รองศาสตราจารย์ณัฐพร เห็นเจริญเลิศ

อาจารย์ ดร.ขจิตพรรณ มกระธัช

สาขาวิชาวิทยาศาสตร์และเทคโนโลยี

โครงการนี้ได้รับการสนับสนุนจากทุนพัฒนาบุคลากร 2555

1. ชื่อ นางสาวณัฐพร นามสกุล เห็นเจริญเลิศ อายุ 45 ปี

ตำแหน่ง รองศาสตราจารย์ ระดับ 9 และ

ชื่อ นางสาวขจิตพรรณ นามสกุล มกระธัช อายุ 35 ปี

ตำแหน่ง อาจารย์ ดร. ระดับ 6

สังกัด สาขาวิชาวิทยาศาสตร์และเทคโนโลยี โทร 025048191-3

ไป เข้าร่วมฝึกอบรมทางวิชาการ

เรื่อง Object-Oriented Software Development with UML

ณ. Software park นนทบุรี ตั้งแต่วันที่ 18-22 มิถุนายน พ.ศ. 2555

รวมระยะเวลา 5 วัน

2. รายงานการฝึกอบรม

(1) วิธีการฝึกอบรม มีการเรียนภาคทฤษฎีและฝึกปฏิบัติไปพร้อมๆกัน มีการแบ่งกลุ่มทำ work shop กรณีศึกษาที่วิทยากรกำหนดให้ โดยทำการดาวน์โหลดและติดตั้งซอฟต์แวร์ STARUML ซึ่งเป็นโอเพนซอร์ส เป็นซอฟต์แวร์ในการฝึกอบรม

(2) สาระสำคัญของการฝึกอบรม เป็นการอบรมทั้งภาคทฤษฎีและภาคปฏิบัติ เกี่ยวกับการพัฒนาระบบแบบ OOA โดยซอฟต์แวร์ UML โดยเนื้อหาของการฝึกอบรมแสดงไว้ในเอกสารรายงานที่แนบด้านท้าย

(3) มีผู้เข้าร่วมฝึกอบรม ประมาณ 20 คน ส่วนใหญ่เป็นผู้ทำงานบริษัทเอกชนที่เกี่ยวข้องกับงานด้านไอที และอาจารย์จากมหาวิทยาลัย

(4) ประโยชน์ที่ได้รับ

(4.1) ประโยชน์ที่ผู้รับทุนได้รับ

- ได้รับความรู้ใหม่เกี่ยวกับการพัฒนาระบบเชิงวัตถุ
- สามารถฝึกหัดใช้ซอฟต์แวร์และทำกรณีศึกษาตามที่วิทยากรกำหนดได้

(4.2) ประโยชน์ที่มหาวิทยาลัยได้รับ

- สามารถนำความรู้ที่ได้ไปใช้กับการเรียนการสอนชุดวิชา 99702 การพัฒนาระบบสารสนเทศ การบริหารโครงการและการประยุกต์ ได้
- สามารถนำโจทย์กรณีศึกษาไปปรับใช้เป็นแบบฝึกปฏิบัติสำหรับนักศึกษาได้

(5) ข้อเสนอแนะ

เนื่องจากมีหลักสูตรประยุกต์ด้านเทคโนโลยีสารสนเทศและการสื่อสารจำนวนหลายหลักสูตรที่จัดขึ้นโดยองค์กรภายนอก ทางมหาวิทยาลัยควรพิจารณาอนุมัติให้สาขาวิชาวิทยาศาสตร์และเทคโนโลยีสามารถเชิญผู้ทรงคุณวุฒิภายนอกมาเป็นวิทยากรให้ความรู้แก่คณาจารย์และบุคลากรที่สนใจในวิชาการดังกล่าว เพื่อการประหยัดค่าใช้จ่ายการลงทะเบียนที่ค่อนข้างสูง และเปิดโอกาสให้บุคลากรอื่นๆ สามารถเข้าฝึกอบรมด้วยได้ เนื่องจากการขออนุมัติไปฝึกอบรมในปัจจุบันอนุญาตให้เฉพาะบุคลากรของแต่ละสาขาฯ สามารถลงทะเบียนและเข้าฝึกอบรมเดียวกันได้ไม่เกิน 2 คน ทำให้ขาดโอกาสในการแสวงหาความรู้ใหม่ๆ โดยเฉพาะด้านเทคโนโลยีสารสนเทศและการสื่อสารที่มีการปรับเปลี่ยนอย่างรวดเร็วตลอดเวลา

Object-Oriented Software Development with UML

ความเป็นมาของUML

UML มาจากคำเต็มว่า “Unified Modeling Language” เป็นการนำสัญลักษณ์มาใช้ในยุคที3 ของ Object Oriented ในช่วงปี1994 – 1996 จิม รัมบอ (*Jim Rumbaugh*) นำทีมวิจัยที่บริษัท General Electric พัฒนาแนวคิดเรื่อง Object Modeling Technique (OMT) และในปี 1994 ไอวาร์ จาคอบสัน(*Ivar Jacobson*) จากบริษัท Ericsson ได้นำเสนอแนวความคิดในเรื่องของ Use case เป็นคนแรก ซึ่งช่วงนั้นมีระเบียบวิธีที่มีลักษณะคล้ายคลึงกันดังกล่าวเกิดขึ้นมากมาย และในขณะเดียวกันก็มีรายละเอียดปลีกย่อยที่แตกต่างกันออกไป แนวความคิดพื้นฐาน ที่เหมือนกันถูกแสดงด้วยสัญลักษณ์ที่ต่างกัน ซึ่งทำให้เกิดความสับสนในการสื่อความหมาย องค์การ OMG พยายามที่จะสร้าง มาตรฐานขึ้นมา เกรดี บูช (Grady Booch) พยายามประสานงานอย่างไม่เป็นทางการ ซึ่งก็ไม่ประสบผลสำเร็จแต่อย่างใด ต่อมามีการประชุม OOPSLA ในปี 1994 โดยมีการประกาศว่า Jim Rumbuagh ได้ร่วมกับ Grady Booch ที่ Rational Software เพื่อร่วมกันพัฒนาระเบียบวิธีของที่ต่างคนต่างคิดขึ้นด้วยกัน ในการประชุม OOPSLA

ปี 1995 Grady และ Jim ได้เตรียมการเผยแพร่เป็นครั้งแรกเกี่ยวกับระเบียบวิธีที่ร่วมกันพัฒนา ในรูปเอกสาร Unified Method version 0.8 ที่ได้รับความสนใจเป็นอย่างมาก ในช่วงนั้น Rational Software ได้เข้าซื้อกิจการของ Objectory และทำให้ Ivar Jacobson เข้าร่วมใน Unified team ของ Grady และ Jim ด้วย

ดังนั้น ในปี 1996 Grady, Jim และ Ivar ได้ดำเนินการพัฒนาระเบียบวิธีในชื่อใหม่ว่า Unified Modeling Language (UML) ในเดือนมกราคมปี 1997 หลายๆองค์กรได้ เสนอข้อเสนอ (Proposal) ที่ระบุถึงมาตรฐานสำหรับการแลกเปลี่ยนแบบจำลองในระหว่างแต่ละระเบียบวิธี (Methods standard to facilitate the interchange of models) ข้อเสนอเหล่านี้ให้ความสำคัญกับ Meta-model และสัญลักษณ์ประกอบ (Optional notations) บริษัทRational Software ก็ได้ออกเอกสาร UML version 1.0 เพื่อเป็นข้อเสนอต่อ OMG เช่นกัน ข้อเสนอต่างๆ จึงถูกรวมเข้าด้วยกัน และ OMG (Object Management Group) ได้พัฒนามาตรฐาน OMG อย่างเป็นทำงการใน version 1.1 (Official OMG standard) นับแต่นั้นเป็นต้นมา กลุ่มงาน Revision Task Force (RTF) นำโดย Cris Kobryn ก็ได้ดำเนินการพิจารณาปรับปรุงมาเป็นลำดับ โดย version 1.2 นั้นเป็นการปรับปรุงเพียงเล็กน้อย จนถึง version 1.3 ซึ่งได้นำเสนอสู่สาธารณะในปี 1999 และนับว่าเป็นการปรับปรุงครั้งสำคัญ

การวิเคราะห์และออกแบบระบบ

การวิเคราะห์ระบบคือ การระบุปัญหาที่เกิดขึ้น (what) เป็นการวิเคราะห์โมเดลของหลักการที่จะอำนวยความสะดวกในการทำความเข้าใจกับระบบงานเดิม โดยจะยังไม่พิจารณาเรื่องซอฟต์แวร์หรือโปรแกรมที่จะนำมาใช้ และยังไม่คำนึงถึงสมรรถนะของระบบ การวิเคราะห์เปรียบได้กับเครื่องมือที่ช่วยให้เข้าใจปัญหาและการทำงานของระบบเดิม

การออกแบบระบบ คือการอธิบายว่าการแก้ปัญหาของระบบเดิมจะทำอย่างไร (how) เป็นการจัดทำโมเดลเพื่อสนับสนุนการพัฒนาและติดตั้งระบบงานคอมพิวเตอร์ หรืออาจเป็นการปรับปรุงซอฟต์แวร์ในระบบเดิม และการปรับปรุงสมรรถนะของระบบงานเดิมหรือซอฟต์แวร์ระบบงานเดิม การออกแบบเป็นกระบวนการที่เปลี่ยนการวิเคราะห์ให้เป็นคุณลักษณะของซอฟต์แวร์ ในการพัฒนาซอฟต์แวร์ไม่ควรใช้ SDLC ควรนำหลักการ *แนวปฏิบัติที่ดีของวิธีวิศวกรรมซอฟต์แวร์* ทั้ง 6 ข้อมาประยุกต์

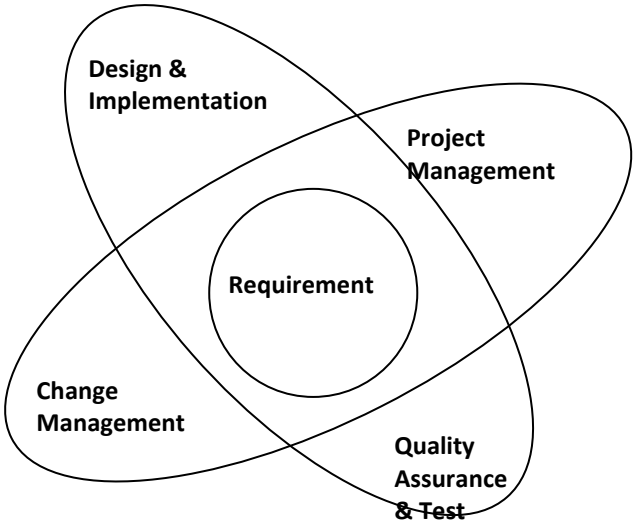
แนวปฏิบัติที่ดีของวิธีวิศวกรรมซอฟต์แวร์(Best Practices of Software Engineering) ประกอบด้วย 6 ข้อ **ดังนี้**

1. พัฒนาแบบวนซ้ำ (Develop Iterative) เป็นการพัฒนาซอฟต์แวร์ที่ทำขั้นตอนทุกขั้นตอนย่อยซ้ำ ๆ กันตั้งแต่การเก็บรวบรวมความต้องการ การออกแบบ การเขียนโปรแกรมการทดสอบโปรแกรม และการบูรณาการโปรแกรมเข้าด้วยกัน ซึ่งในการวนซ้ำแต่ละรอบจะช่วยลดความเสี่ยงที่ระบบจะล้มเหลวลงได้มากกว่าแบบวงจรพัฒนาระบบ (SDLC) สามารถจัดทำรายการสาเหตุหลัก (root causes) และคำตอบในการแก้ปัญหา (solutions) ได้ เช่น

Root Causes	Solutions
✔ O Insufficient requirement ←	Enables and encourages user feedback
✔ O Ambiguous Communications ←	Serious misunderstandings evident early in the life cycle
✔ O Overwhelming Complexity ←	Develop focuses on critical issues
✔ O Subjective Assessment ←	Objective Assessment thru testing

✔ Undetected inconsistencies ←	Inconsistencies detected early
✔ Poor testing ←	Testing stars earlier
✔ Waterfall development ←	Risks identified and addressed early

2. จัดการความต้องการของผู้ใช้ (Manage User Requirements) เป็นการรวบรวมความต้องการของผู้ใช้จากการสัมภาษณ์ และเอกสารที่เกี่ยวข้อง นำมาจัดลำดับความสำคัญและวิเคราะห์หน้าที่และข้อจำกัด ประเมินการเปลี่ยนแปลงและกำหนดผลกระทบที่เกิดจากการเปลี่ยนแปลงนั้น ๆ ความต้องการของผู้ใช้เป็นสิ่งที่มีการเปลี่ยนแปลงได้ตลอดเวลาในระหว่างที่พัฒนาซอฟต์แวร์ ความต้องการเป็นศูนย์กลางและเป็นสิ่งสำคัญต่อการพัฒนาและองค์ประกอบหลาย ๆ องค์ประกอบของโครงการพัฒนาฯ



3. ใช้สถาปัตยกรรมแยกเป็นองค์ประกอบย่อย ๆ (Use Component-Based Architectures) เป็นการพัฒนาซอฟต์แวร์โดยแยกระบบออกเป็นองค์ประกอบย่อย ๆ โดยองค์ประกอบดังกล่าวสามารถทำงานประสานร่วมกันได้ โครงสร้างซอฟต์แวร์ที่ออกแบบต้องคำนึงถึง การใช้ หน้าที่การทำงาน สมรรถนะการทำงาน ความยืดหยุ่น การนำมาใช้ซ้ำ ความสามารถในการทำความเข้าใจ ประหยัดค่าใช้จ่ายและข้อจำกัดของเทคโนโลยีและสุนทรียศาสตร์

ซอฟต์แวร์รูปแบบการมองเห็น (Visually Model Software) หมายถึง สถาปัตยกรรมที่ออกแบบต้องสามารถแสดงให้เห็นถึงการเข้ากันได้ขององค์ประกอบย่อยต่าง ๆ ซ่อนหรือเปิดเผยรายละเอียดได้ตามความเหมาะสมกับงานแต่ละงาน(task) รักษาความสม่ำเสมอระหว่างการออกแบบกับการพัฒนาและติดตั้งใช้งาน ลดความคลุมเครือและช่วยเพิ่มความสามารถในการจัดการความซับซ้อนของซอฟต์แวร์ มีการใช้ภาษายูเอ็ม

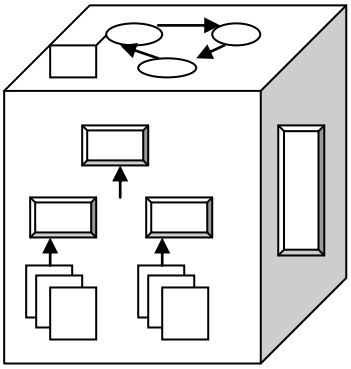
แอล (UML- Unified Modeling Language) ซึ่งเป็นภาษาสำหรับกำหนดคุณลักษณะเฉพาะ สามารถมองเห็นได้เพราะเป็นภาษาที่สื่อด้วยภาพ และยังเป็นภาษาที่ใช้ในการสร้างและจัดทำเอกสารประกอบการพัฒนาซอฟต์แวร์ด้วย การพัฒนาระบบสารสนเทศจะใช้เพียงทักษะด้านภาษาย่อมไม่เพียงพอ ทีมงานพัฒนาระบบนอกจากจะต้องมีทักษะพื้นฐานความรู้เกี่ยวกับฮาร์ดแวร์ ซอฟต์แวร์แล้ว ยังต้องใช้ Modeling Language และ Process ในการพัฒนาระบบด้วย

The Unified Modeling Language (UML) ภาษามาตรฐานสำหรับการจำลองระบบสารสนเทศ ด้วยภาพ UML เป็นภาษาในการวาดแบบจำลองเพื่อจุดประสงค์ 4 ประการ

- จำลองด้วยภาพ (visualizing)
- แสดงข้อกำหนด (specifying)
- ช่วยสร้างระบบ (constructing)
- ช่วยบันทึก (documenting) ชิ้นงานในขั้นตอนต่างๆ ของการพัฒนาระบบสารสนเทศ

4. ตรวจสอบคุณภาพซอฟต์แวร์ (Verify Software Quality) เป็นการตรวจสอบคุณภาพในด้านต่าง ๆ ของซอฟต์แวร์ก่อนที่จะนำไปติดตั้งใช้งาน เนื่องจากการนำซอฟต์แวร์ที่ไม่มีคุณภาพไปใช้นั้นมักจะทำให้ต้องเสียค่าใช้จ่ายและเวลาในการปรับปรุงภายหลัง ซึ่งส่งผลกระทบอย่างมากต่อการดำเนินงานของหน่วยงานด้วย
5. ควบคุมการเปลี่ยนแปลงของซอฟต์แวร์ (Control Changes to Software) ซอฟต์แวร์มีการเปลี่ยนแปลงเนื่องจากสาเหตุต่าง ๆ เช่น มีผู้พัฒนาหลายคน มีทีมพัฒนาหลายทีม มีหน่วยงานที่ต้องติดตั้งใช้งานหลายแห่ง มีการพัฒนาวนซ้ำหลายรอบ มีหลายแพล็ตฟอร์ม ฯลฯ การจัดการการเปลี่ยนแปลงมี 3ด้านหลัก ได้แก่ การจัดการการคำร้องขอของผู้ใช้ที่เปลี่ยนแปลง (Change Request Management) การจัดการองค์ประกอบ (Configuration Management) การวัด (Measurement)

Change Request Management (CRM)



Management

Configuration Management (CM)

Introduction to Unified Process

การสร้างระบบสารสนเทศจะใช้เพียงทักษะด้านภาษาย่อมไม่เพียงพอ ทีมงานพัฒนาระบบนอกจากจะต้องมีทักษะพื้นฐานความรู้เกี่ยวกับฮาร์ดแวร์ ซอฟต์แวร์แล้ว ยังต้องใช้ Modeling Language และ Process ในการพัฒนาระบบด้วย

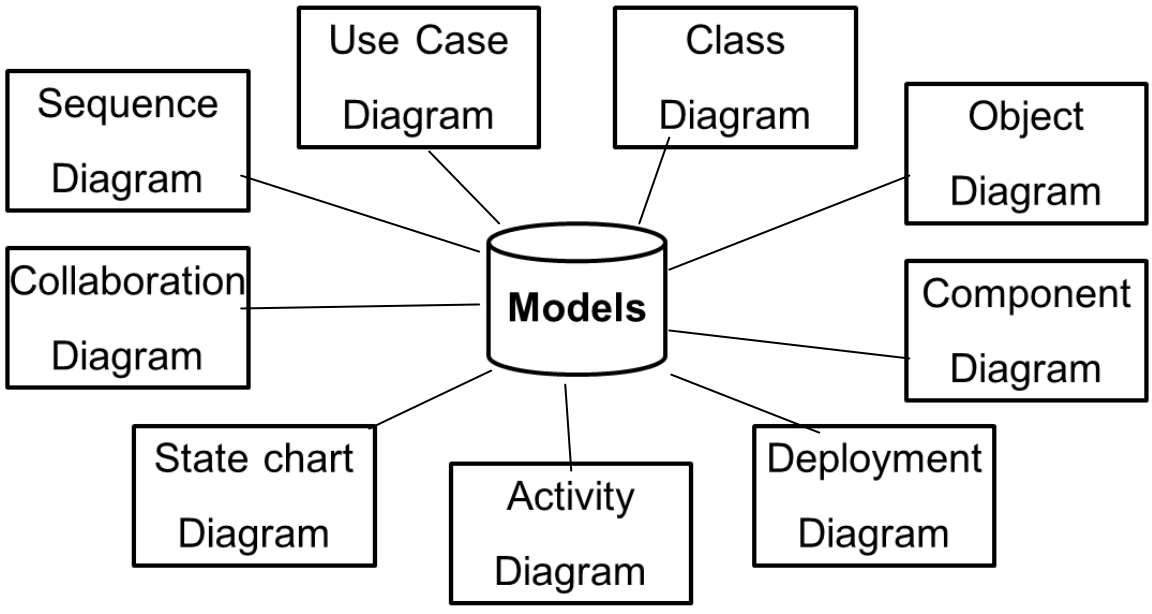
The Unified Modeling Language (UML) ภาษามาตรฐานสำหรับการจำลองระบบสารสนเทศ ด้วยภาพ UML เป็นภาษาในการวาดแบบจำลองเพื่อจุดประสงค์ 4 ประการ

- จำลองด้วยภาพ (visualizing)
- แสดงข้อกำหนด (specifying)
- ช่วยสร้างระบบ (constructing)
- ช่วยบันทึก (documenting)

ชิ้นงานในขั้นตอนต่างๆ ของการพัฒนาระบบสารสนเทศ

ประวัติ UML พัฒนาโดย Booch และ Rumbaugh เป็น Unified Method 0.8 ในปี 1997 และมี Jacobson มาร่วมพัฒนาด้วย เป็น UML 0.9 -> UML 1.0 จนในปัจจุบันเป็น UML 2.0 UML

UML ประกอบด้วย 9 แผนภาพดังนี้



กระบวนการในการทำ Software Engineering นั่นคือ สามารถบอกได้ว่าใครทำอะไร ทำเมื่อใดและทำได้อย่างไร หนึ่งในกระบวนการคือ Rational Unified Process ซึ่งเป็น Best Practices ที่ประกอบด้วย 6 ขั้นตอนคือ

1. Manage Requirement การจัดการความต้องการ
2. Develop Iteratively แบ่งแยกการพัฒนาออกเป็นส่วนย่อยๆ
3. Model Visually มีรูปแบบ
4. Verify Quality การตรวจสอบคุณภาพ
5. Use Component Architecture มีองค์ประกอบในเชิงสถาปัตยกรรม
6. Control Changes การควบคุมการเปลี่ยนแปลง

Rational Unified Process จะใช้วิธีการของ Use-Case Driven คือแสดงด้วยแผนภาพเหตุการณ์ 5 มุมมองหลักของ UML

1.Use-case view : หน้าที่การทำงานของระบบซอฟต์แวร์ โดยพิจารณาจากมุมมองของผู้ใช้ภายนอก หรือ ระบบภายนอก

- use-case diagram

2. Design View : หน้าที่การทำงานของระบบมีโครงสร้างอย่างไร มองในรูปของ static structure และdynamic behavior

- class diagram, object diagram, state, sequence, collaboration, activity diagrams

3. Implementation View: องค์ประกอบย่อยในการ implement ที่ประกอบเป็นระบบ และ dependency ระหว่างองค์ประกอบเหล่านั้น

- component diagram

4. Process view: การแบ่งแยก process และ processors โดยพิจารณาทั้ง communication และ synchronization

-dynamic diagrams (state, sequence, collaboration activity)

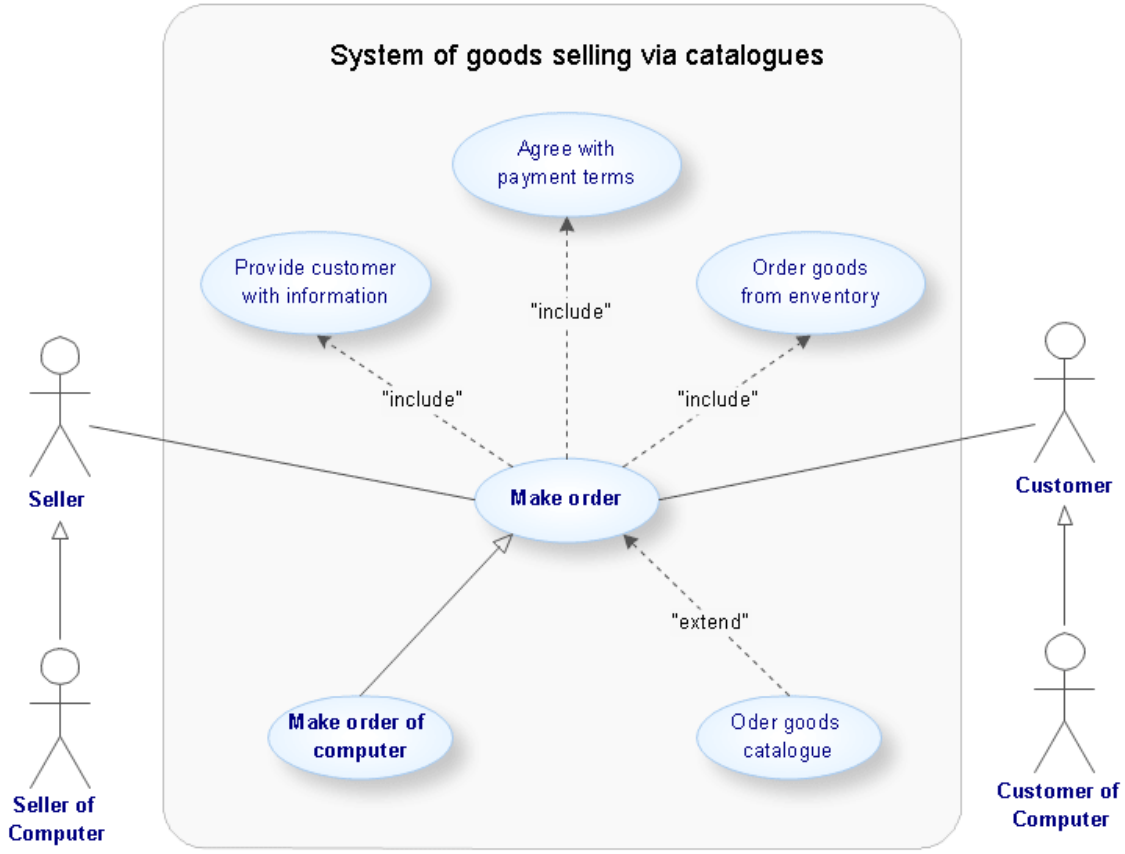
-implementation diagrams (component และ deployment)

5.Deployment view : โครงสร้างทางกายภาพเกี่ยวกับ การติดตั้ง และใช้งานระบบ

- deployment diagram

- Use Case Diagram ใช้เพื่อแสดงขอบเขตของระบบ และ ฟังก์ชั่นการทำงานหลักของระบบ

ตัวอย่างแผนภาพของ UML ที่แสดงด้วย USE-CASE Diagram



ทั้งนี้แผนภาพที่ใช้ในการอธิบายระบบส่วนใหญ่มีดังต่อไปนี้

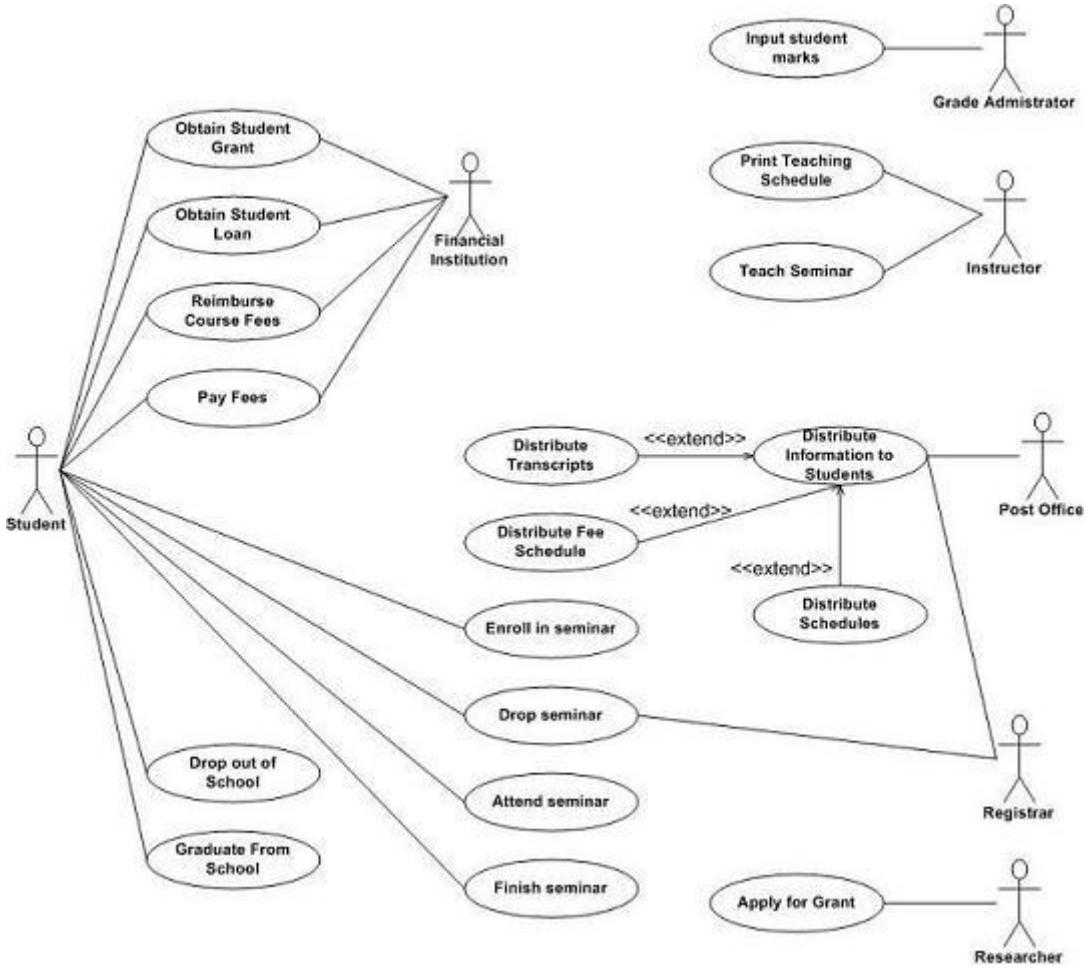
1. **Use Case Diagram** ใช้เพื่อแสดงขอบเขตของระบบ และ ฟังก์ชั่นการทำงานหลักของระบบ

สัญลักษณ์ที่ใช้ใน Use Case แสดงดังตาราง

สัญลักษณ์	ความหมาย
1. Use Case Name 	Use Case คือ หน้าที่ที่ระบบจะต้องทำ
2. Actor Name 	Actor คือ ผู้เกี่ยวข้องกับระบบ (แสดงบทบาทเป็นผู้คาดหวังผลลัพธ์จากระบบ หรือทำหน้าที่ผลักดัน ให้เกิดกิจกรรมของระบบหรือทำหน้าที่ควบคุมดูแลกิจกรรมของระบบ หรือสัมผัสกับระบบโดยตรง)
3. System Name 	System Boundary คือ เส้นแบ่งขอบเขตระหว่างระบบกับ Actor
4. Connection	Connection คือ เส้นเชื่อมระหว่าง Actor กับ Use Case

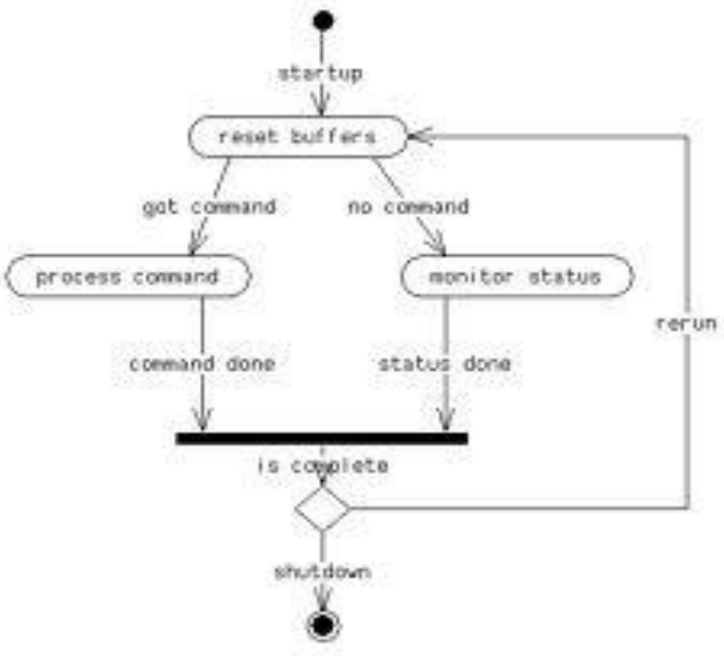
การพัฒนา Use Case แสดงถึงการวิเคราะห์ระบบแบบ logical โดยไม่ต้องระบุถึงอุปกรณ์ หรือ อินเทอร์เฟสต่างๆ โดยแสดงถึงสิ่งที่ต้องการจากระบบโดยไม่ต้องสนใจว่าจะทำได้อย่างไร

ตัวอย่างระบบลงทะเบียนที่แสดงด้วย Use Case diagram



2. **Activity Diagram** ใช้เพื่ออธิบายกระบวนการทำงานในภาพรวม หรือ กระบวนการในรายละเอียด

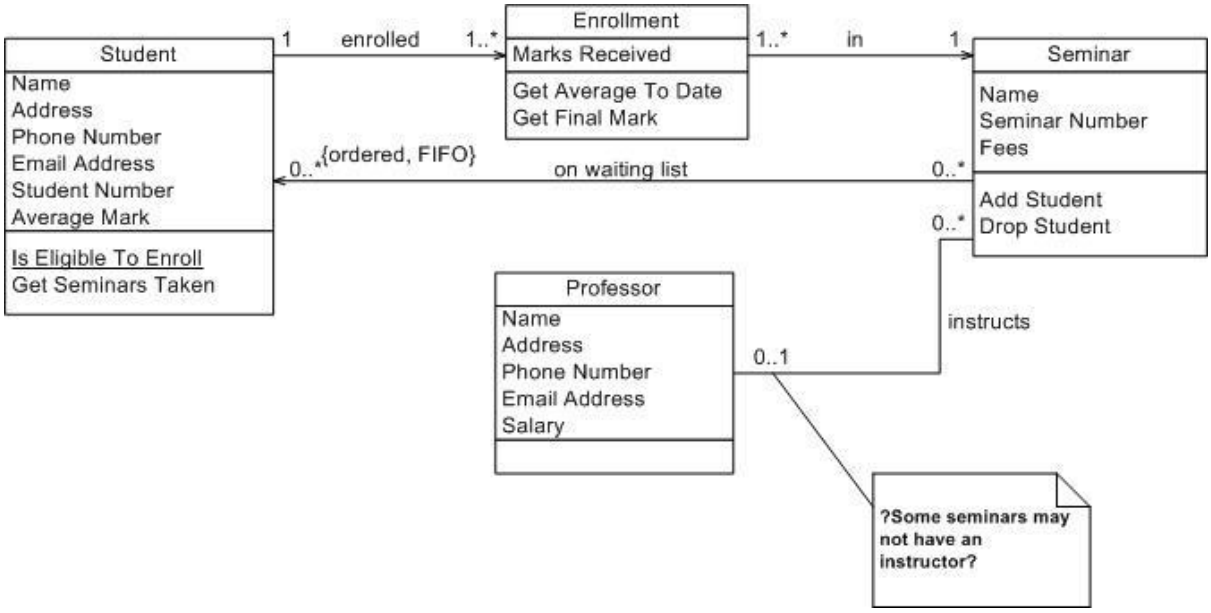
ตัวอย่าง Activity Diagram



3. **Class diagram** ใช้เพื่ออธิบายโครงสร้างคงที่ของระบบสารสนเทศทั้งด้านข้อมูล และกิจกรรมหลักที่กระทำกับข้อมูลนั้น

สัญลักษณ์ Class diagram เป็นรูปสี่เหลี่ยมประกอบด้วย 3 ส่วน คือส่วนที่เป็นชื่อ (name) ส่วนที่เป็นรายละเอียด (attribute) และส่วนที่เป็นการทำงาน (operation) ในสัญลักษณ์ไม่จำเป็นต้องแสดงหมดทั้ง 3 ส่วน

ตัวอย่าง Class diagram ของระบบลงทะเบียน



เมื่อสร้าง Class diagram ของระบบได้ครบถ้วนแล้วจึงสร้าง Data dictionary เพื่อเป็นที่เก็บคำศัพท์และคำอธิบายของระบบทั้งหมด เพื่ออธิบายความหมายของคำต่างๆ ที่ใช้ในระบบให้ชัดเจนเป็นที่เข้าใจทั้ง 2 ฝ่ายคือ ฝ่ายผู้พัฒนาระบบ และผู้ใช้ระบบ

4. **Sequence diagram** เป็นแผนภาพแสดงเหตุการณ์ (scenarios) ที่เกี่ยวข้องของแต่ละ use case และแสดงให้เห็นถึงการปฏิสัมพันธ์ระหว่าง Object ณ เวลาต่างๆ ทั้งนี้ use case หนึ่งๆ สามารถมีหลายเหตุการณ์ได้ และเมื่อพิจารณาระดับความสำคัญของแต่ละแผนภาพ พบว่า แผนภาพ use case แสดงกรอบงาน (framework) แผนภาพ class diagram ระบุธุรกรรมต่างๆที่เกี่ยวข้อง

Sequence diagram ประกอบด้วย

- Class/Object

- เส้นเพื่อใช้แสดงลาดับเวลา
- เส้นเพื่อแสดงกิจกรรมที่เกิดขึ้นจาก Object/Class
- มีการใช้สี่เหลี่ยมแทน Class/Object ภายในกรอบสี่เหลี่ยมมีชื่อของ Object/Class ประกอบอยู่ในรูปแบบ {Object}:Class
- กิจกรรมที่เกิดขึ้นแทนด้วยลูกศรแนวนอนจาก Class/Object หนึ่งไปยังอีก Class/Object ตัวต่อไป ระบุชื่อกิจกรรมในรูปแบบ {[Conditional]} Operation
- ชื่อของกิจกรรมต้องเป็น Operation ที่อยู่ใน Class/Object ที่ลูกศรชี้ไป

การปฏิสัมพันธ์ระหว่าง object ผ่าน Sequence diagram มีองค์ประกอบดังต่อไปนี้

- Object name บอกชื่อของออบเจ็กต์ ออบเจ็กต์ที่อยู่ทางซ้ายมือจะทำงานก่อนออบเจ็กต์ที่อยู่ทางขวามือ
- Lifeline เส้นประที่ลากในแนวดิ่งจากออบเจ็กต์
- Activation สี่เหลี่ยมเล็กๆ ที่อยู่บนเส้น lifeline แทนการทำงานต่างๆ ของออบเจ็กต์ของ activation นั้น

การส่ง message หรือ ข้อความ ระหว่างออบเจ็กต์ มีลักษณะดังต่อไปนี้

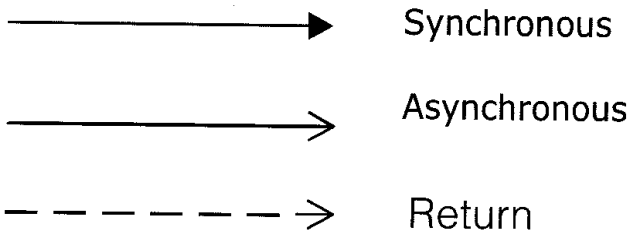
- Synchronous เป็นการส่งข้อความหรือติดต่อแบบรอคอยคำตอบ หรือการตอบกลับก่อนที่จะทำงานอื่นๆ ต่อไป
- Asynchronous เป็นการส่งข้อความหรือติดต่อแบบไม่รอคอยคำตอบ ไม่มีการหยุดทำงานของผู้ส่ง ผู้ส่งสามารถทำงานต่อได้
- Return เป็นข้อความที่เกิดขึ้นในกรณีที่ต้นทางเริ่มการติดต่อแล้วปลายทางต้องมีการติดต่อกลับด้วย โดยการส่งข้อความจะเขียนข้อความกำกับไว้ด้วย
- Time หรือช่วงเวลา เป็นการแสดงเวลาในลักษณะแนวตั้ง หรือจากบนลงล่าง ข้อความที่อยู่ด้านบนจะเป็นส่วนที่เกิดขึ้นก่อนข้อความที่อยู่ด้านล่าง
- ถ้าหากเป็นข้อความเงื่อนไข จะเขียนเงื่อนไขไว้ในวงเล็บก้ามปู [ ] โดยข้อความจะถูกส่งก็ต่อเมื่อเงื่อนไขนั้นเป็นจริง

ประโยชน์ของ Sequence diagram

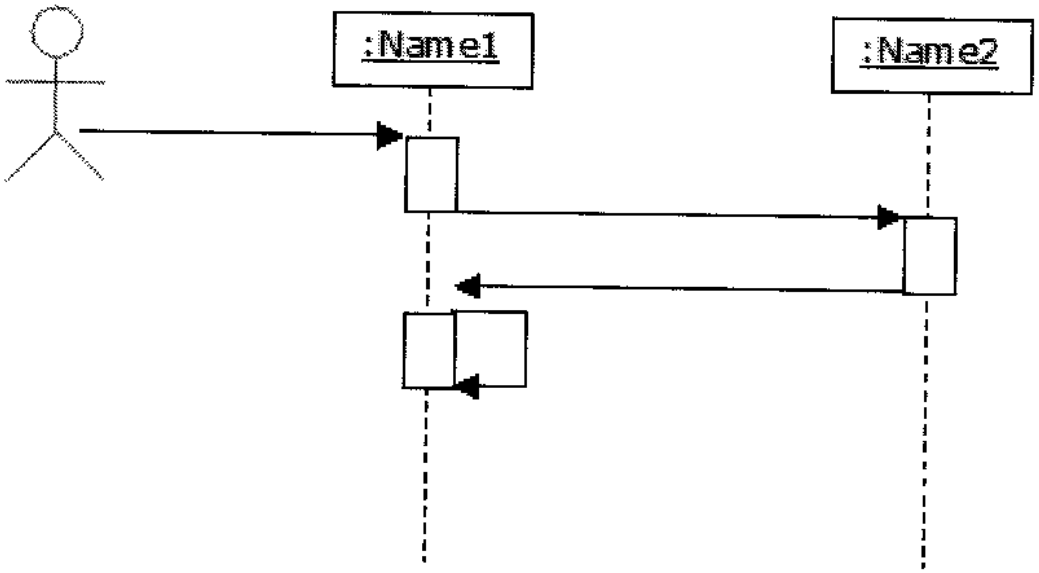
- ช่วยให้รู้ว่า มี Function ขาดหายไปจาก Class Diagram หรือไม่
- ช่วยให้พิจาณาได้ว่าควรเพิ่มเติม Function ใน Class Diagram อีกหรือไม่

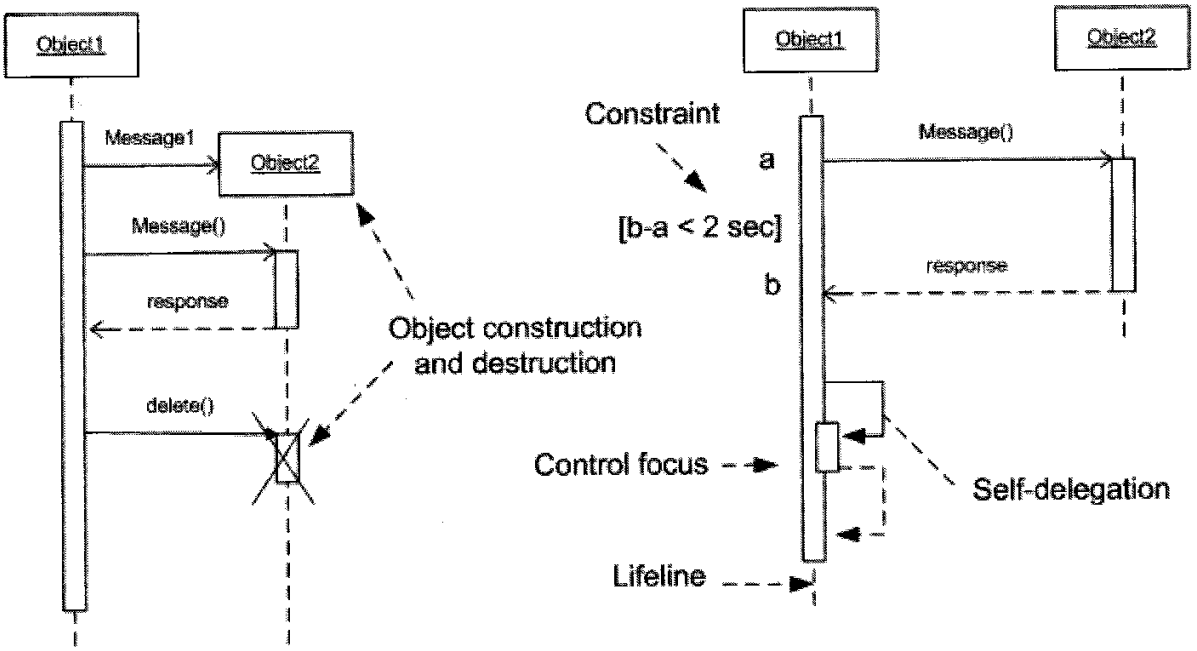
- ช่วยให้ปรับปรุง Class Diagram ได้สมบูรณ์ยิ่งขึ้น

สัญลักษณ์ที่ใช้แทนข้อความทั้ง 3 แบบ มีลักษณะดังต่อไปนี้



สัญลักษณ์มาตรฐานของ Sequence Diagram แสดงได้ดังนี้





เทคนิคการสร้าง Sequence Diagramจาก Use Case และ Class Diagram

1. พิจารณาทีละ Use Case โดยยังไม่ต้องคำนึงถึงความสัมพันธ์ที่แต่ละ Use Case มีต่อกัน
2. 2.พิจารณาแต่ละ Use Case ว่ามี Class/Object ใดร่วมทำให้เกิดกิจกรรมใน Use Case นั้นๆ บ้าง
3. นำเอา Class/Object ต่างๆ มาเรียงต่อกันในแนวนอน โดยนำ Actor (ในกรณีที่ Use Case นั้นมี Actor ด้วย) ไว้ที่ด้านซ้ายมือสุดเสมอ แล้วนำ Class/Object ต่างๆ เรียงต่อกันจากซ้ายไปขวาตามความเหมาะสม
4. หาก Use Case นั้นมี Actor โดยปกติแล้วกิจกรรมแรกที่ถูกเรียกมักจะเกิดจาก Actor ก่อนเสมอ ดังนั้นเมื่อเกิดกิจกรรมที่ไปยัง Class/Object ใด ให้ย้าย Class/Object นั้นมาทางซ้าย ทำเช่นนี้ไปเรื่อยๆ จนกระทั่งกิจกรรมทั้งหมดครบถ้วน
5. กรณีที่มีกิจกรรมเกิดขึ้นใหม่ แต่ Operation ที่เกิดขึ้นไม่มีใน Class/Object ที่ลูกศรชี้ไป ให้เข้าไปเพิ่ม Operation นั้นๆ ลงไปที่ Class นั้นใน Class Diagram
6. หากต้องการเพิ่ม Class ใหม่เข้าไปใน Sequence Diagram ต้องเข้าไปเพิ่มเติม Class นั้น และ Relationship ที่มีทั้งหมดใน Class Diagram ด้วย (แต่ Class ที่เพิ่มเข้าไปนั้นเป็น Class เพื่อจำลองกิจกรรมที่เกิดขึ้นจริงๆ ของระบบเท่านั้น ไม่ใช่ Class ที่เพิ่มเข้าไปเพื่อการ Implement เช่น User Interface ต่างๆ)
7. ทำขั้นตอน 1 –6 จนครบทุก Use Case
8. การสร้างความสัมพันธ์ของ Sequence Diagram จาก Use Case ที่มีการ Uses/Extends ทำได้โดยการนำ Class และกิจกรรมที่เกิดขึ้นใน Use Case ที่ถูก Uses/Extends มาแทรกเข้าไปใน Use Case ที่เรียกใช้ และใช้กิจกรรมเพื่อเชื่อมโยง Sequence Diagram ทั้งสอง

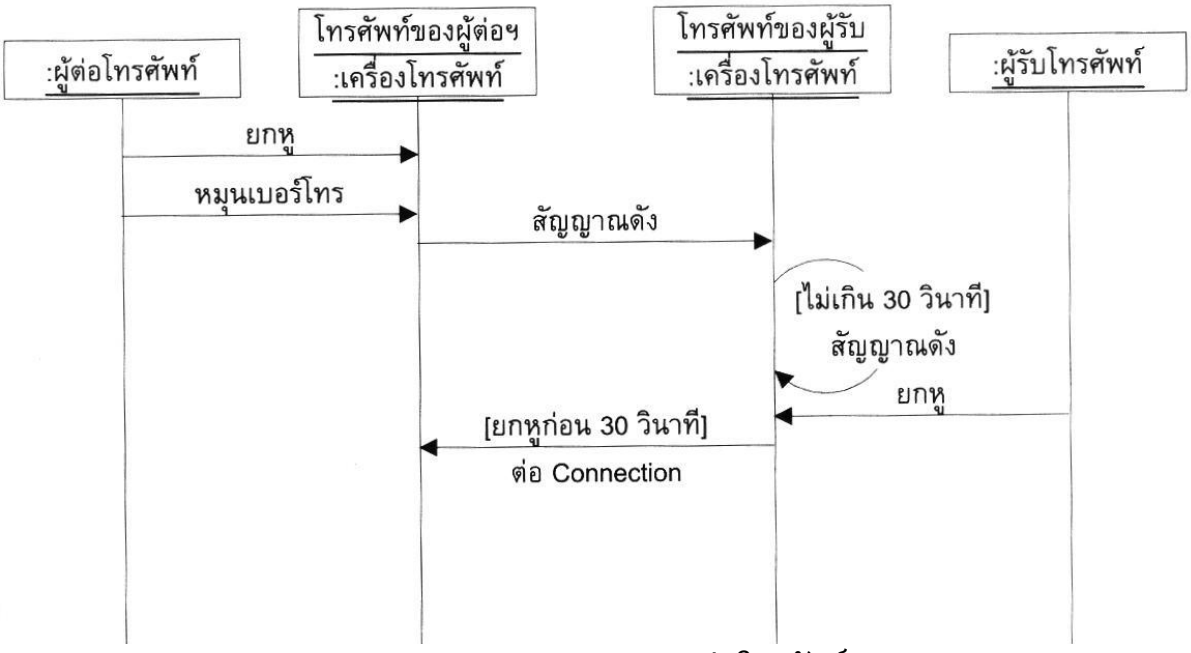
ตัวอย่าง Sequence diagram ของการคุยโทรศัพท์

Sequence Diagram ของการคุยโทรศัพท์ ประกอบด้วย Use Case ต่างๆ ดังนี้

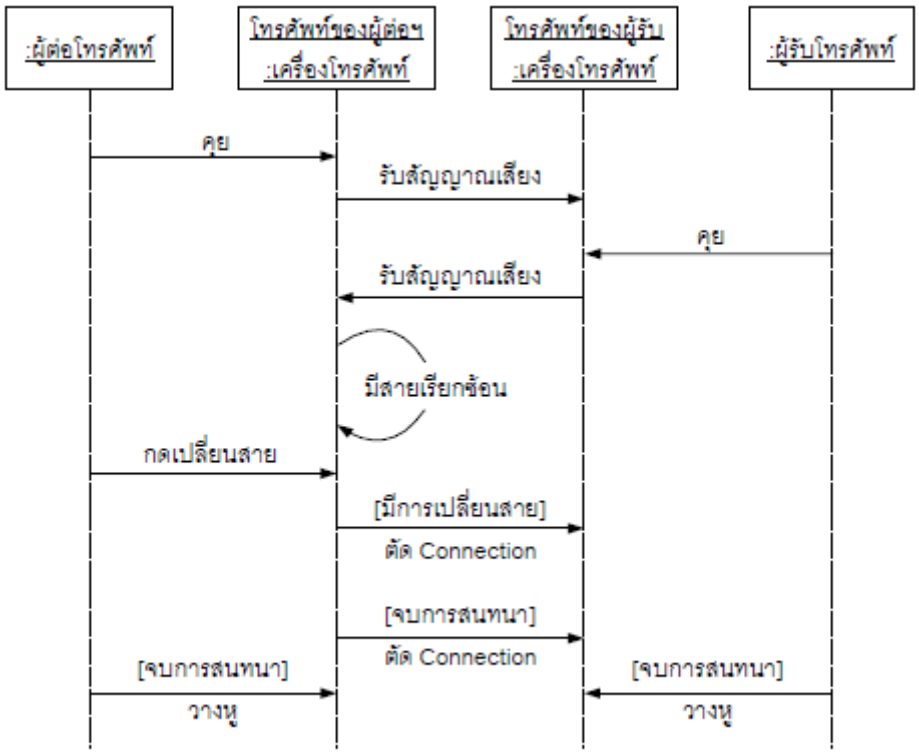
- การต่อโทรศัพท์
- การคุยโทรศัพท์
- การมีสายซ้อน (เป็นUse Case ที่ Extends การคุยโทรศัพท์)

ประกอบด้วย Class ต่างๆ ดังนี้

- ผู้ติดต่อโทรศัพท์ (Actor)
- ผู้รับโทรศัพท์ (Actor)
- เครื่องโทรศัพท์



Sequence diagram **ของการต่อโทรศัพท์**



Sequence diagram **ของสายเรียกซ้อน**

Communication Diagram

- เน้นที่การแสดงการทำงานของข้อความต่างๆ ในระบบ อธิบายการโต้ตอบในระบบ
- เป็นไดอะแกรมที่นามาแทน Collaboration Diagram ใน UML 1.x

สัญลักษณ์ของออบเจ็กต์ : Object หรือ : Object

ความสัมพันธ์

ลูกศรสำหรับบอกทิศทางการส่งเมสเสจ

Synchronous message

Asynchronous message

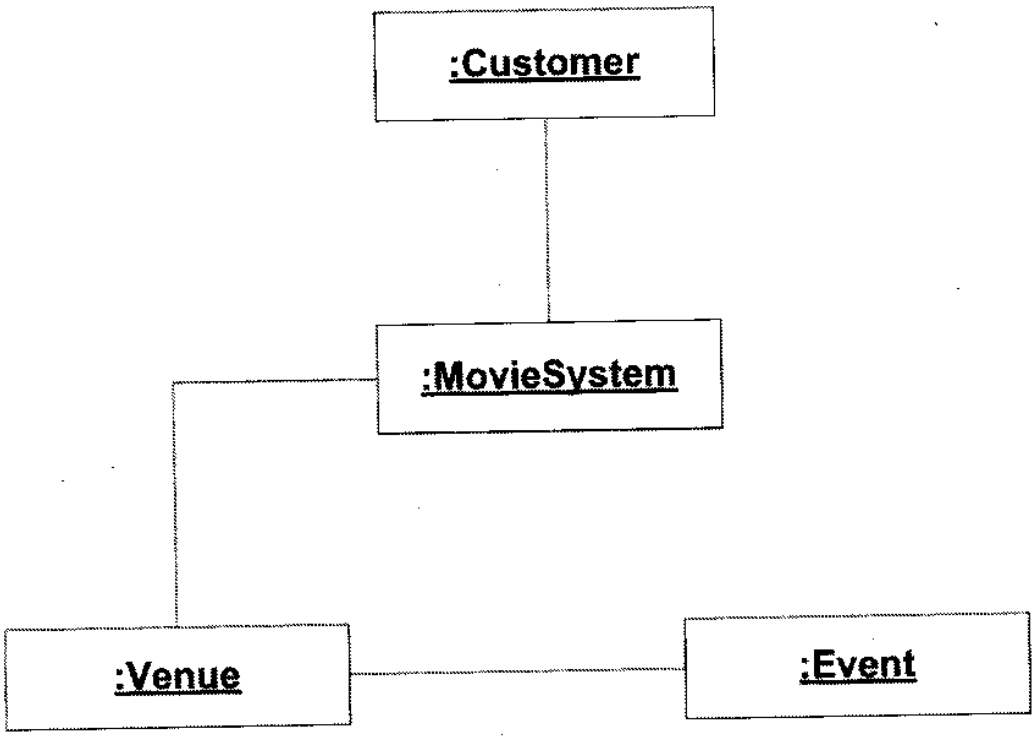
Return message

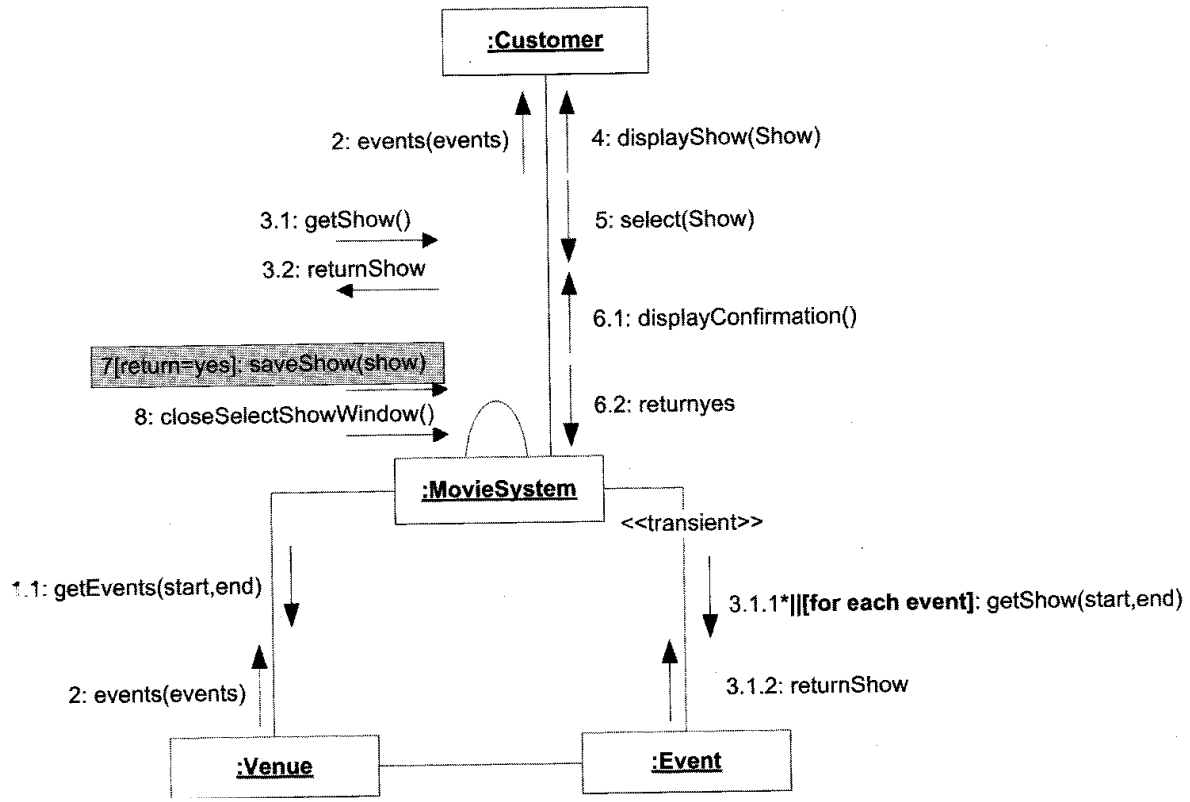
Procedure call

ตัวอย่าง Communication Diagram

การกำหนดลำดับการทำงานของข้อความควรเริ่มต้นด้วยการสร้างลิงค์ระหว่างออบเจ็กต์ก่อน จากนั้นทำการระบุข้อความต่างๆ ของระบบที่เรากำลังสนใจ ระบุลำดับให้กับข้อความเหล่านั้นโดยการใช้ตัวเลขจำนวนเต็ม เช่น 1.1, 3.2, 3.2.4 เป็นต้น นอกจากนี้มีการใช้ตัวอักษรร่วมด้วย เพื่อแสดงระดับการทำงานที่เท่ากันของข้อความมากว่า 1 ข้อความ เช่น 1.1a และ 1.1b เป็นข้อความที่อยู่ระดับเดียวกัน

- ในกรณีต้องการระบุว่าข้อความใดมีการทำซ้ำ (ภายใต้เงื่อนไข) ให้ใช้สัญลักษณ์ดอกจัน (*) ตามด้วยเงื่อนไขการทำซ้ำ และข้อความที่ต้องการให้ทำซ้ำ
- การกำหนดให้ข้อความที่มีการทำซ้ำให้ทำงานไปพร้อมๆ กันแบบคู่ขนาน ให้ใช้สัญลักษณ์ || ตามหลังเครื่องหมายดอกจัน
- กรณีต้องการกำหนดเงื่อนไขการทำงานให้กับข้อความ ด้วย Guard-Condition ซึ่งเป็นเงื่อนไขที่เป็นจริงหรือเท็จช่วยเราในการกำหนดเงื่อนไขดังกล่าว
- สัญลักษณ์ที่ใช้ จะเหมือนสัญลักษณ์การวนซ้ำแต่ไม่มีเครื่องหมายดอกจันนำหน้า





ดังนั้นสรุปได้ดังนี้ แผนภาพยูเอ็มแอลที่อยู่ในกลุ่มพฤติกรรม ประกอบด้วย 4 แผนภาพ ดังนี้

(1) แผนภาพยูสเคส (Use Case Diagrams) แสดงการปฏิสัมพันธ์ (interact) ในสองรูปแบบคือ ระหว่าง (1) ระบบกับผู้ใช้ หรือ (2) ระบบกับระบบภายนอก (External Systems) กล่าวคือ ยูสเคสจะแสดงว่าใครบ้างที่เป็นผู้ใช้ระบบ และผู้ใช้เหล่านั้นมีการติดต่อกับระบบในลักษณะใด ยูสเคสมีจุดเด่นคือเป็นแผนภาพที่สามารถทำความเข้าใจได้ง่าย นักวิเคราะห์ระบบมักจะใช้ยูสเคสในช่วงการรวบรวมความต้องการของระบบ (System Requirement) ซึ่งจะช่วยให้นักวิเคราะห์สามารถกำหนดขอบเขตของระบบได้อย่างถูกต้องครบถ้วน

(2) แผนภาพกิจกรรม (Activity Diagrams) แสดงให้เห็นกลุ่มของการกระทำ หรือการปฏิบัติงานที่เรียกว่า แอกชัน (action) ซึ่งประกอบขึ้นตามลำดับขั้นรวมกันเป็นกิจกรรม (activity) ของงาน แผนภาพกิจกรรมจะแสดงให้เห็นถึงขั้นตอนการปฏิบัติงาน ทางเลือกและเงื่อนไขของทางเลือกที่มีในแต่ละกิจกรรม

(3) แผนภาพสเตจแมชีน (State Machine Diagrams) แสดงสถานะของระบบตามช่วงชีวิต รวมทั้งแสดงเหตุการณ์ใด ๆ ที่สามารถเปลี่ยนสถานะของระบบเพื่อการเสริมสร้างความเข้าใจ โดยเฉพาะในการอธิบายการทำงานที่ซับซ้อน แผนภาพนี้สามารถใช้ในการอธิบายพฤติกรรม

ระบบทั้งหมด หน่วยย่อยของระบบ (Subsystem) หรือแม้แต่เพียงวัตถุเดียว (Single Object) ในระบบ

(4) แผนภาพปฏิสัมพันธ์ (Interaction Diagrams) แผนภาพที่จัดในกลุ่มนี้ออกเป็น 4 แบบดังนี้

- แผนภาพลำดับ (Sequence Diagrams) แสดงการปฏิสัมพันธ์ระหว่างอ็อบเจกต์ตามลำดับของเหตุการณ์ที่เกิดขึ้น แผนภาพลำดับมีลักษณะที่ง่ายต่อการทำความเข้าใจ จึงมักนำมาช่วยให้ผู้ใช้งานสามารถเห็นภาพการทำงานของระบบ และมักจะถูกนำมาใช้เสริมกับแผนภาพกิจกรรม เนื่องจากแผนภาพกิจกรรมอาจจะประกอบไปด้วยเงื่อนไขจำนวนมาก ทำให้มีทางเลือกในการทำงานได้หลายเส้นทาง เราจึงนำแผนภาพลำดับมาใช้ในเฉพาะเส้นทางที่เลือกบางเส้นที่ต้องการอธิบายขยายความ
- แผนภาพการสื่อสาร (Communication Diagrams) แสดงวิธีที่อ็อบเจกต์ใช้ในการปฏิสัมพันธ์ รวมทั้งการเชื่อมโยง (connections) ที่ใช้ในการสนับสนุนการปฏิสัมพันธ์นั้น ในยูเอ็มแอลเวอร์ชันแรกเรียกแผนภาพแบบนี้ว่า แผนภาพการร่วมมือ (Collaboration Diagrams)
- แผนภาพเวลา (Timing Diagrams) แสดงการเปลี่ยนแปลงสถานะ (state) เงื่อนไข (condition) หรือบทบาท (role) ของอ็อบเจกต์ตามลำดับของเวลา โดยปกติสถานะที่เปลี่ยนแปลงไปนั้นมีสาเหตุเนื่องมาจากการตอบสนองต่อเหตุการณ์ใด ๆ ที่เกิดขึ้น
- แผนภาพแสดงการปฏิสัมพันธ์อย่างหยาบ (Interaction Overview Diagrams) ใช้ในการรวบรวมแผนภาพลำดับ แผนภาพการสื่อสาร และแผนภาพเวลา เข้าด้วยกันเพื่อแสดงการปฏิสัมพันธ์ที่สำคัญที่เกิดขึ้นในระบบ

การพัฒนาซอฟต์แวร์

การพัฒนาซอฟต์แวร์ให้ประสบความสำเร็จ ควรให้ความสำคัญต่อ 4 ด้าน คือ

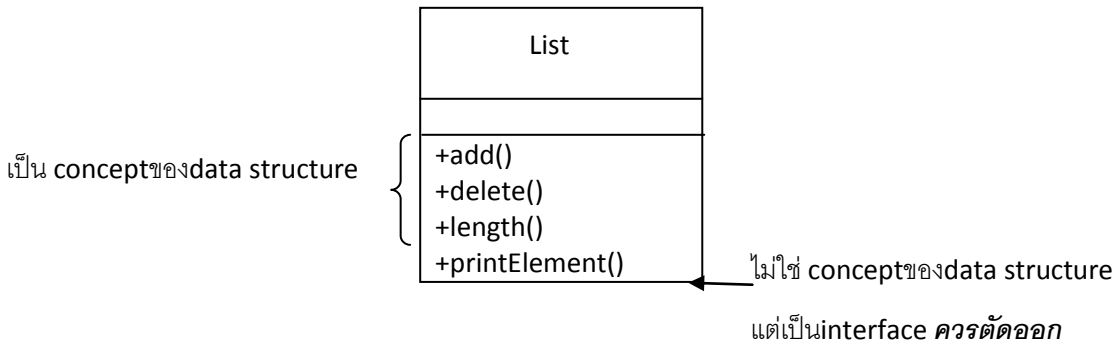
1. บุคลากรในทีมงาน ผู้บริหารโครงการพัฒนาซอฟต์แวร์ควรให้ความสำคัญต่อบุคลากรในทีมงานฯ มากกว่ากระบวนการและเครื่องมือ
2. เน้นการมีระบบซอฟต์แวร์ที่ใช้งานได้ดีมากกว่าเอกสารคู่มือการใช้งานซอฟต์แวร์
3. ความร่วมมือของผู้ใช้สำคัญกว่าข้อตกลงในสัญญา ทั้งนี้เนื่องจากความต้องการของผู้ใช้ในระหว่างที่พัฒนาซอฟต์แวร์ที่อาจแตกต่างจากที่ระบุในสัญญาทั้งหมด ทีมงานพัฒนาฯควรรับฟังความต้องการดังกล่าว หากไม่แตกต่างจากที่ระบุในสัญญามากก็ควรดำเนินการตามที่ผู้ใช้ร้องขอมา
4. ซอฟต์แวร์สามารถตอบสนองต่อความต้องการที่เปลี่ยนแปลงมีความสำคัญมากกว่าการดำเนินการตามแผนอย่างเคร่งครัด แต่ซอฟต์แวร์ที่พัฒนาไม่สามารถตอบสนองต่อความต้องการของผู้ใช้

การปรับปรุงรูปแบบของซอฟต์แวร์ที่ออกแบบไว้ในภาพรวม ต้องยึดหลักที่ทำให้ซอฟต์แวร์มีความสามารถดัง ต่อไปนี้

- ยืดหยุ่นสามารถปรับขนาดได้ (scalable)
- มีความแข็งแร่งในการทำงาน (robust) ไม่ล่ม (shutdown)
- สามารถนำกลับมาใช้ได้อีก (reusable)

ในระยะการออกแบบ (Design Phase) ควรใช้หลักการ 5 ข้อ ต่อไปนี้ ได้แก่

1. ครบถ้วนสมบูรณ์ (Completeness) การออกแบบที่ดีจะต้องครอบคลุมชุดของ operation ที่ครบถ้วนในคลาสแต่ละคลาส
2. ความเพียงพอ (sufficiency) การออกแบบที่ดีจะต้องไม่รวม operation ที่ไม่จำเป็นในคลาสแต่ละคลาส มีเฉพาะเท่าที่ต้องมี
3. ความเรียบง่าย (primitive) การออกแบบ operation ควรเป็นแบบเรียบง่ายไม่ซับซ้อน
4. มีความเป็นอันหนึ่งอันเดียวกันสูง (high cohesion) บางครั้งเรียกว่า Single Responsibility Principle (SRP) หมายความว่า operation แต่ละ operation ควรสนับสนุนหลักการทำงานเดียว ไม่ควรทำงานหลาย ๆ งานโดยใช้ operation เดียว
5.

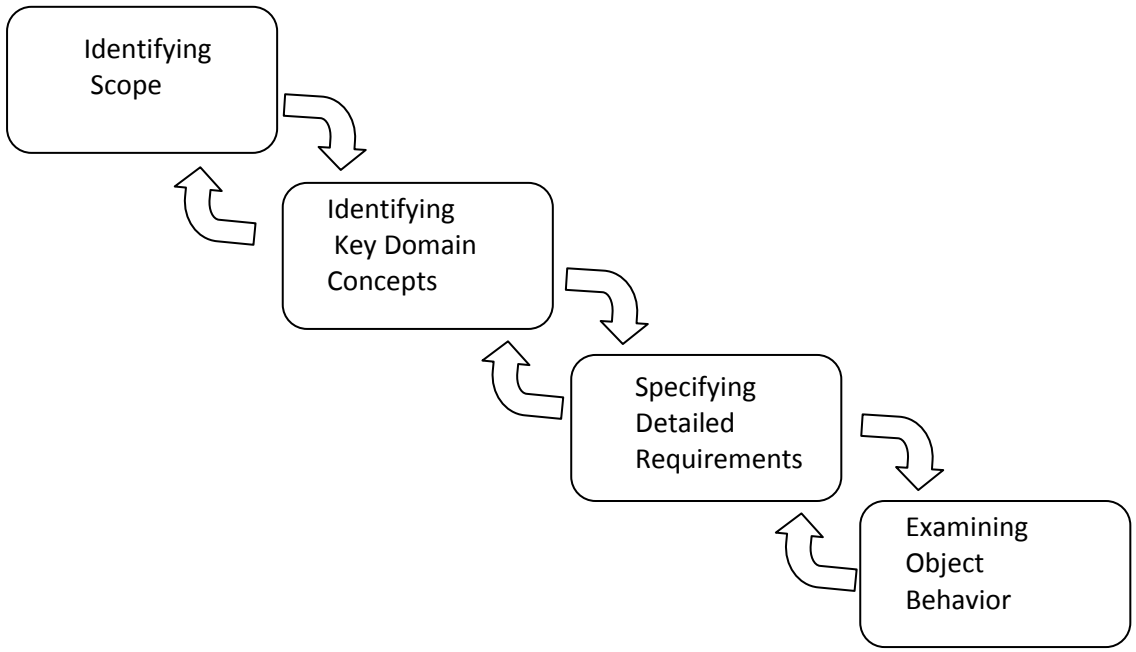


6. มีความเกี่ยวข้องหรือขึ้นต่อกันน้อย (low coupling) ระบบต่าง ๆ ไม่ควรทำงานเกินขอบเขตของตัวเองและเข้าไปทำงานในระบบอื่นมากเกินไป ควรเป็นลักษณะการทำงานที่ส่งต่อค่าพารามิเตอร์ระหว่างกันเท่านั้น

การออกแบบออปเจ็กต์(object design)

ในการออกแบบออปเจ็กต์ ควรทบทวนรายละเอียดการวิเคราะห์ เน้นหัวข้อการออกแบบที่สำคัญ ๆ แยกความแตกต่างระหว่างการวิเคราะห์กับการออกแบบให้ชัดเจน

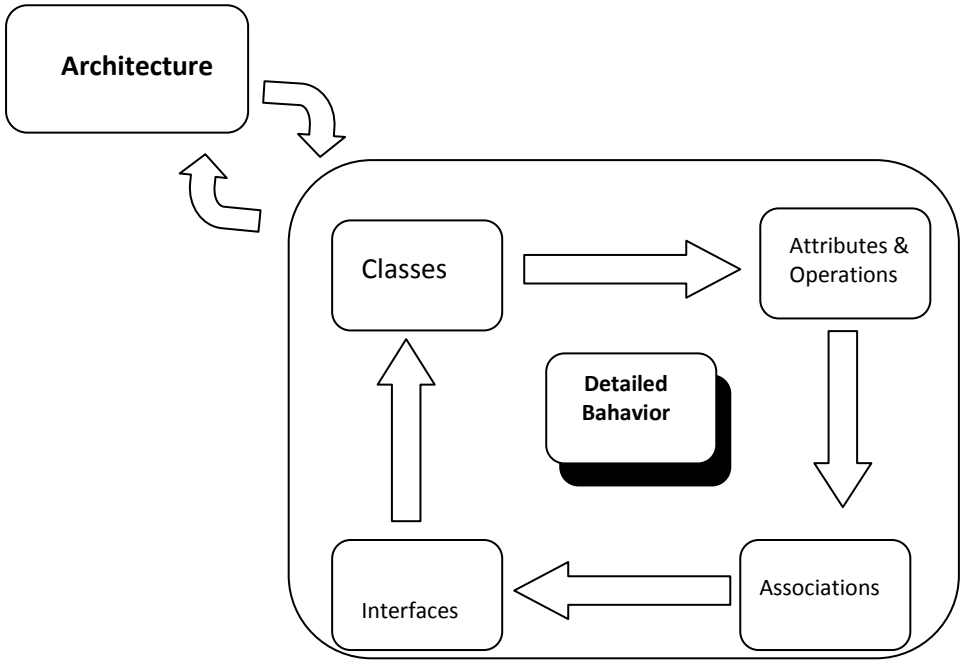
สรุปขั้นตอนการวิเคราะห์ ดังแสดงในภาพข้างล่างนี้



1. การกำหนดขอบเขต ควรเขียนแผนภาพuse case เพื่อระบุ
    - Actors
    - System behavior
    - System boundary
2. การกำหนดหลักการโดเมนสำคัญ โดยสร้างclass diagram ประกอบด้วย data dictionary ระบุclass และนิยาม class เพื่อเป็นกรอบงานสำหรับรายละเอียดความต้องการของผู้ใช้ สร้างที่จัดเก็บข้อมูลรายละเอียดของระบบทั้งหมด
3. สร้างscenarios และsequence diagram เพื่อระบุ
    - Logical system Interaction
    - Attributes & Operations
    - Associations
4. ตรวจสอบobject behavior state diagram เพื่อระบุ
    - Operation definition
    - Attributes

ทำการสำรวจความสมบูรณ์ครบถ้วน (completeness) ความสม่ำเสมอ (consistency)

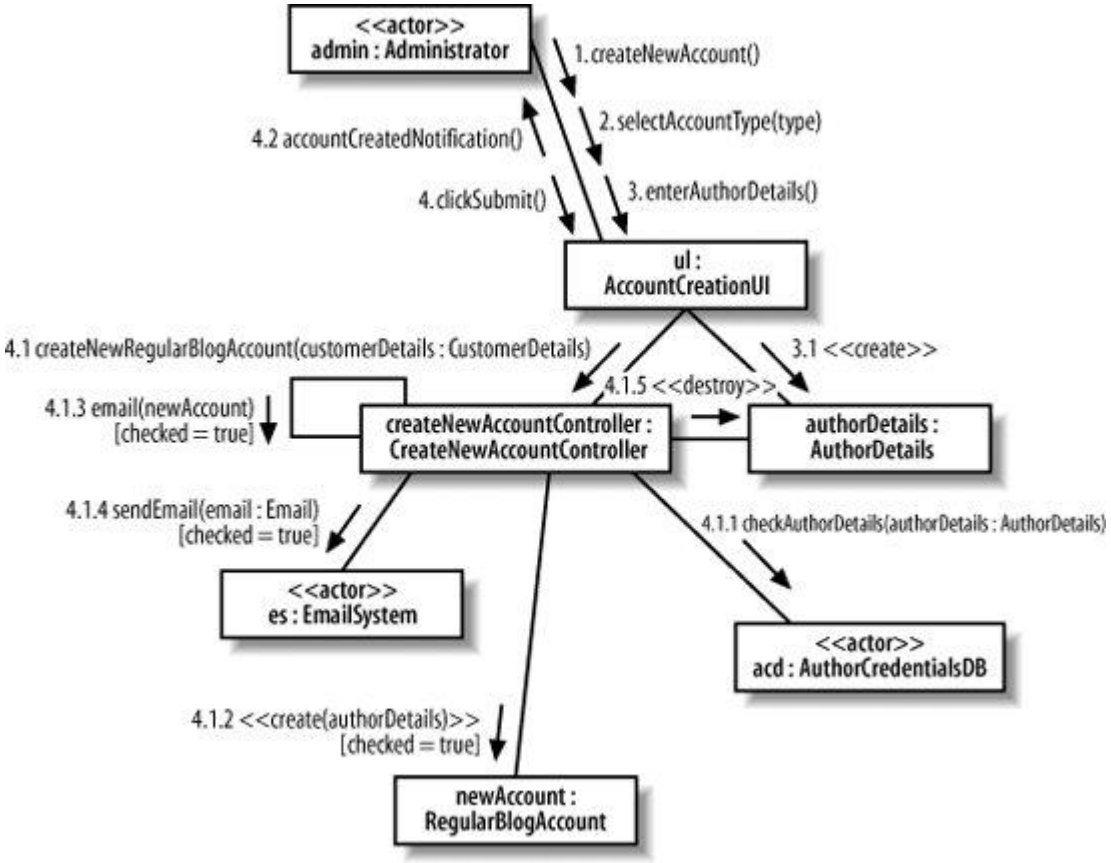
สรุปขั้นตอนการออกแบบ เป็นการมุ่งเน้นโครงสร้างซอฟต์แวร์ แทนที่จะเป็นการทำความเข้าใจเกี่ยวกับความต้องการของผู้ใช้ที่มีต่อซอฟต์แวร์ การออกแบบซอฟต์แวร์ที่ดีสามารถลดทรัพยากรที่ต้องใช้ในการดูแลบำรุงรักษาซอฟต์แวร์ในอนาคตลงได้ ในการออกแบบจะมีการนำข้อจำกัดต่าง ๆ มาพิจารณา เช่น เนื้อที่หน่วยความจำที่ใช้ในการประมวลผล เวลา ความซับซ้อนของซอฟต์แวร์ เป็นต้น ทำการเก็บรายละเอียดที่อาจละเลยไป ดำเนินการประยุกต์หลักการออกแบบ เช่น encapsulation reuse inheritance เป็นต้น และทำการออกแบบสถาปัตยกรรมของซอฟต์แวร์ ดังแสดงในภาพข้างล่างนี้



การระบุรายละเอียดขอ งsystem behavior เป็นการระบุว่า object มีการรับส่งข้อมูลอะไรบ้างระหว่างกัน ซึ่งนิยมใช้ Collaboration Diagram เป็นแผนภาพแสดงการติดต่อสื่อสารระหว่างกัน และมีลักษณะคล้ายกับ Sequence Diagram วัตถุประสงค์ของการเขียนแผนภาพ Collaboration Diagram คือ

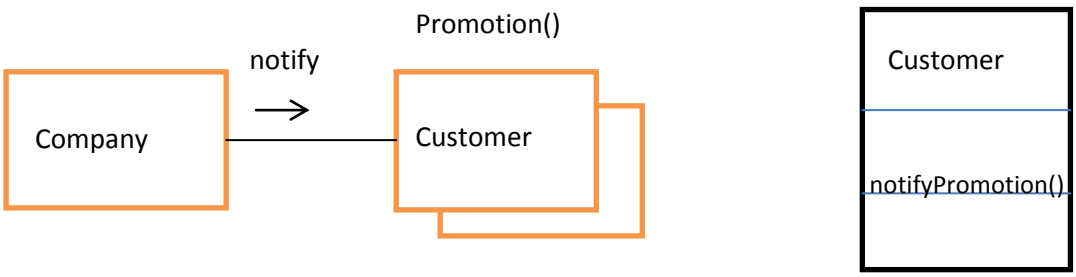
- ทำความเข้าใจการทำงานร่วมกันระหว่าง objectในระบบ
- สังเกตการทำงานของ object ทั้งหมดในระบบและ methods ที่มีสามารถทำงานเข้ากันได้มากน้อยเพียงใด
- ออกแบบรายละเอียดให้มีมากขึ้น เช่น การปฏิสัมพันธ์กันระหว่างคลาส กลไกของ association การใส่ qualifiers
- สิ่งสำคัญของการพัฒนาซอฟต์แวร์คือ การสื่อสารที่ดี (well communication) ระหว่างผู้ใช้กับทีมงานพัฒนาฯ ต้องมีการยืนยันรายละเอียดเป็นระยะ ๆ แม้แต่ในทีมงานเองก็ต้องเข้าใจตรงกัน ซึ่งสามารถทำได้โดยการประชุมร่วมกัน ทำเอกสารร่วมกัน โดยเฉพาะกรณีที่ซับซ้อนมาก ๆ ควรนำโมเดลมาช่วยในการสื่อสาร การออกแบบซอฟต์แวร์จะมีการใช้

Collaboration Diagram เพื่อแสดงรายละเอียด behavior ของซอฟต์แวร์ เป็นแผนภาพที่สร้างคล้ายกับSequence Diagram บางครั้งเรียกว่า Communication Diagram แสดงการรับส่งข้อมูลระหว่าง object ใช้ตรวจสอบว่า objectและ method ทั้งหมดมีความเหมาะสม จะใช้สัญลักษณ์คล้ายกันกับ Sequence Diagram แต่จะมีเส้นลูกศรกำกับบนเส้น association และมีตัวเลขกำกับโดยใช้ระบบดิวอี้คือเรียงตามลำดับและระดับ พร้อมคำอธิบาย operation ที่เกิดระหว่าง object ดังภาพ



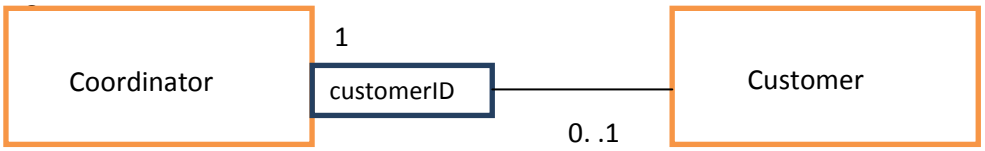
แผนภาพการสื่อสาร(communication diagram)

กรณีที่ต้องทำงานกับหลายobjectจะใช้สัญลักษณ์ภาพสี่เหลี่ยมซ้อนกัน ดังภาพต่อไปนี้



จากภาพบริษัทจะส่งข่าวสารการส่งเสริมการขายให้แก่ลูกค้าหลาย ๆ คน ซึ่งลูกค้า (Customer) เป็น class หนึ่งและมี notify promotion() เป็น operation อยู่แล้ว

กรณีที่ต้องการระบุว่า object 2 object ทำงานร่วมกันอย่างไร อ้างอิงผ่านอะไรจะใช้ Qualifier ซึ่งสัญลักษณ์ของ Qualifier คือกล่องสี่เหลี่ยมขนาดเล็กติดกับสัญลักษณ์ผู้ส่งและอยู่ที่ปลายเส้น association ภายในกล่องประกอบด้วยคีย์ (key) ดังภาพ



การใช้ Collaboration Diagram จะทำให้เห็น association((link) ที่ชัดเจนระหว่าง object เพราะ object ต้องมี link กันก่อนจึงจะรับส่ง message กันได้ การรับส่ง message ก็คือ การ call functionกันนั่นเอง การระบุประเภทของ link จะช่วยในการตรวจสัมพันธ์ ซึ่งประเภทของ linkมีด้วยกัน 5 ประเภท คือ

- Association จะใช้แอตทริบิวที่ถาวร
- Parameter เป็นการส่งผ่านการเรียก procedure
- Global เป็นการอ้างอิงไปยั งobjectที่ทุกคนรู้จักและอ้างอิงไปยัง class หรืออ้างอิงไปยัง classของผู้ที่ร้องขอมา
- Local เป็นแอ็ตทริบิวส์ชั่วคราว (ภายในกระบวนการที่มีการเรียก)
- Self เป็นการ link กับตัวเอง

โดยทั่วไป tools ที่ใช้เขียน UML จะสามารถแปลง (convert) Sequence Diagram ไปเป็น Collaboration Diagram ได้ แต่ใน Star UML เรียกว่า Interaction Instance

การออกแบบ interface เป็นการออกแบบการทำงานประสานกันระหว่างระบบย่อยๆของซอฟต์แวร์ เป็นคุณลักษณะเฉพาะของชุด object และ behavior การออกแบบ Interface ทำให้ได้คุณลักษณะเฉพาะที่สามารถนำไปประยุกต์ในการพัฒนาและติดตั้งระบบซอฟต์แวร์อื่น ๆ ได้ และทำให้ผู้ใช้เกิดความเข้าใจในการทำงานกับซอฟต์แวร์ได้ดียิ่งขึ้น ช่วยปรับปรุงการออกแบบและการพัฒนาซอฟต์แวร์ให้มีคุณภาพมากขึ้น