COS4101 Class Task #02

ความต้องการด้านซอฟต์แวร์

(Software Requirements)

จัดทำโดย

นายณัฐวุฒิ ปลอดสมบูรณ์ 5901505510

นายธนวัฒน์ ขุนพรหม 6305002500

นางสาววรกานต์ ผลพรหม 6405001840

นายศุภณัฐ แซ่เตีย 6505000270

นายอโณทัย วิชาไพบูลย์ 6505501038

นายศรธรรศ ตรีทิพยรักษ์ 6605005476

นายณัฐพงษ์ เข็มโคตร์ 6605501573

เสนอ

ผู้ช่วยศาสตราจารย์ ดร.อัจฉรา มหาวีรวัฒน์

รายงานฉบับนี้เป็นส่วนหนึ่งของการศึกษา

กระบวนวิชา COS4101 วิศวกรรมซอฟต์แวร์

ภาคเรียนที่ 1 ปีการศึกษา 2568

ภาควิชาวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์ มหาวิทยาลัยรามคำแหง

สารบัญ

วิศวกรรมความต้องการ (Software Requirement Engineering)

เป็นกระบวนการในการสร้างและบำรุงเอกสารข้อกำหนดความต้องการ เพื่อทำให้นักพัฒนาระบบสามารถเข้าใจและเข้าถึงความต้องการของผู้ใช้ได้อย่างแท้จริง ด้วยการสกัดความต้องการ ตรวจสอบ และนิยามความต้องการ เพื่อนำไปสร้างเป็นข้อกำหนดความต้องการด้านระบบหรือซอฟต์แวร์ ซึ่งจะถูกนำไปใช้เป็นจุดเริ่มต้นในการพัฒนาระบบในขั้นตอนต่อไป

กระบวนการด้านวิศวกรรมความต้องการจะแบ่งงานออกเป็นงานย่อย ๆ ทั้งหมด 7 ด้าน ได้แก่ การเริ่มต้นวิเคราะห์, การเจาะลึกความต้องการ, การขยายรายละเอียด, การเจรจาต่อรอง, การจัดทำข้อกำหนด, การตรวจรับ และการจัดการความต้องการ

การเริ่มต้นวิเคราะห์ (Inception)

เป็นขั้นตอนแรกของกระบวนการวิเคราะห์ความต้องการ ขั้นตอนนี้จะเป็นการกำหนดภาพรวมว่าเราจะเริ่มต้นโครงการอย่างไร ในขั้นตอนเริ่มต้นนี้จะมีการตั้งคำถามพื้นฐานเกี่ยวกับวิธีการทำงานหรือขั้นตอนที่จำเป็นต้องทำเพื่อให้งานสำเร็จ ซึ่งจะช่วยให้เข้าใจปัญหาพื้นฐานและลักษณะของทางแก้ปัญหาได้ ในขั้นตอนนี้การสื่อสารที่มีประสิทธิภาพถือว่าสำคัญมาก เพราะเป็นรากฐานที่ชี้ชัดว่าจะทำอะไรต่อไป

สิ่งที่วิศวกรซอฟต์แวร์ต้องทำในการวิเคราะห์:

1. **เข้าใจปัญหาอย่างชัดเจน**

   วิศวกรซอฟต์แวร์ต้องทำความเข้าใจปัญหาหรือความต้องการที่แท้จริงของธุรกิจหรือผู้ใช้งาน เพื่อกำหนดเป้าหมายที่ชัดเจน

2. **ใครคือกลุ่มคนที่ต้องการทางแก้ปัญหานี้**

   วิศวกรซอฟต์แวร์ต้องระบุผู้มีส่วนได้ส่วนเสีย เช่น ลูกค้า ผู้ใช้ หรือฝ่ายอื่น ๆ ที่ได้รับผลกระทบหรือมีส่วนเกี่ยวข้องกับระบบ

3. **ลักษณะของทางแก้ปัญหา**

   วิศวกรซอฟต์แวร์ต้องกำหนดลักษณะหรือคุณสมบัติของระบบที่จะพัฒนา เช่น รูปแบบของระบบ ฟังก์ชันหลัก และขอบเขตคร่าว ๆ

4. **การสื่อสารและความร่วมมือระหว่างลูกค้าและผู้พัฒนา**

   วิศวกรซอฟต์แวร์ต้องสร้างช่องทางและกระบวนการสื่อสารที่มีประสิทธิภาพ เพื่อให้ทั้งแต่ละฝ่ายเข้าใจตรงกันและทำงานร่วมกันได้ดี

การเจาะลึกความต้องการ (Elicitation)

เป็นขั้นตอนที่มุ่งเน้นไปที่การรวบรวมความต้องการจากผู้มีส่วนได้ส่วนเสียในโครงการ ขั้นตอนนี้ต้องทำอย่างระมัดระวัง เพราะความต้องการเหล่านี้คือหัวใจสำคัญของวัตถุประสงค์โครงการ การเข้าใจความต้องการที่แท้จริงจากลูกค้าเป็นสิ่งที่สำคัญมากสำหรับนักพัฒนา เพราะถ้าไม่ถูกต้องหรือขาดบางส่วน อาจทำให้เกิดข้อผิดพลาดในระบบได้

ปัญหาต่าง ๆ ที่อาจพบในขั้นตอนนี้

1. **ปัญหาขอบเขต (Problem of Scope):** ความต้องการที่ได้รับอาจมีรายละเอียดที่ไม่จำเป็น ระบุไม่ชัดเจน หรือไม่สามารถนำไปใช้งานจริงได้
2. **ปัญหาความเข้าใจ (Problem of Understanding):** ไม่มีความเข้าใจที่ชัดเจนระหว่างนักพัฒนาและลูกค้าในการกำหนดความต้องการ บางครั้งลูกค้าอาจไม่แน่ใจว่าต้องการอะไร หรือนักพัฒนาอาจเข้าใจผิดเกี่ยวกับความต้องการนั้น ๆ
3. **ปัญหาความเปลี่ยนแปลงของความต้องการ (Problem of Volatility):** ความต้องการที่เปลี่ยนแปลงไปตามเวลา อาจทำให้การบริหารโครงการยากขึ้น และก่อให้เกิดการสูญเสียทรัพยากรและเวลา

เทคนิคที่ใช้

1. **การสัมภาษณ์ (Interview):** นิยมใช้มากที่สุด
2. **การแสดงลำดับเหตุการณ์ (Scenario):** เตรียมคำถามตามลำดับงานของผู้ใช้
3. **การสร้างต้นแบบ (Prototype):** เช่น ออกแบบจอภาพบนกระดาษ เพื่อทดสอบการยอมรับความต้องการในเบื้องต้น
4. **การประชุม (Meeting):** เป็นการเรียกกลุ่มบุคคลที่เกี่ยวข้องมาประชุม เพื่อขอความคิดเห็นและความต้องการ
5. **การสังเกต (Observation):** โดยตรวจสอบสภาพแวดล้อมการทำงานของผู้ใช้ เป็นวิธีที่ดี แต่ค่าใช้จ่ายสูง

การขยายความรายละเอียด (Elaboration)

เป็นขั้นตอนในการนำความต้องการของลูกค้า มาขยายและแจกแจงรายละเอียดเพิ่มเติม กิจกรรมนี้มุ่งจะพัฒนาแบบจำลองทางเทคนิคที่ละเอียดของหน้าที่การทำงาน รวมถึงลักษณะและข้อจำกัดของซอฟต์แวร์

กระบวนการ

1. **การแบ่งกลุ่มความต้องการ (Requirement Classification)**

   แบ่งกลุ่มความต้องการออกเป็น functional, non-functional, product & process หรืออาจแบ่งกลุ่มตามลำดับความสำคัญ

2. **การสร้างแบบจำลองความต้องการ (Requirement Modeling)**

   เป็นการสร้างแบบจำลองแนวคิด เพื่อจำลองความต้องการ ให้เห็นภาพรวมความต้องการ ทีมงานจะต้องเข้าใจความต้องการได้ตรงกัน เพื่อชี้ให้เห็นข้อผิดพลาดและแก้ไขข้อผิดพลาดนั้น

3. **การออกแบบสถาปัตยกรรม (Architectural Design)**

   เพื่อแสดงให้ผู้ใช้สามารถมองเห็นถึงส่วนประกอบต่าง ๆ ที่จะสนับสนุน และรองรับความต้องการได้ง่ายขึ้น

4. **การจัดสรรความต้องการ (Requirement Allocation)**

   เป็นการจัดสรรความต้องการให้เข้ากับองค์ประกอบในแต่ละส่วนของซอฟต์แวร์ เพื่อนำไปวิเคราะห์ในระดับรายละเอียดเพิ่มมากขึ้น

การเจรจาต่อรอง (Negotiation)

ลูกค้ามักต่อรองขอมากกว่าที่ระบบจะทำให้ได้ นักวิศวกรความต้องการ ต้องประสานความขัดแย้งเหล่านี้ผ่านกระบวนการเจรจาต่อรองกับลูกค้า และปรับเปลี่ยนความต้องการบางส่วน เพื่อให้ทุกฝ่ายบรรลุความพอใจ

การจัดทำข้อกำหนด (Specification)

เป็นขั้นตอนในการจัดทำเอกสารเพื่อทำการระบุข้อกำหนดในด้านต่าง ๆ ของระบบ โดยเอกสารจะต้องมีความชัดเจน ครอบคลุม และอธิบายความต้องการของระบบได้ทั้งหมด เพื่อให้ผู้ที่เกี่ยวข้องกับการพัฒนาระบบทุกคนเห็นภาพที่ตรงกัน

การจัดทำข้อกำหนดควรใช้ภาษาที่เป็นธรรมชาติและใช้คำศัพท์ง่าย ๆ โดยควรหลีกเลี่ยงศัพท์เทคนิคเฉพาะทาง และใช้รูปแบบที่สอดคล้องกันตลอดทั้งเอกสาร ซึ่งโดยทั่วไปในการจัดทำมักจะใช้มาตรฐาน IEEE 830-1998: IEEE Recommended Practice for Software Requirements Specifications (SRS) ซึ่งเป็นแนวปฏิบัติแนะนำของ IEEE สำหรับการจัดทำเอกสารข้อกำหนดด้านซอฟต์แวร์

ประเภทของข้อกำหนดที่ต้องระบุ

1. **ข้อกำหนดเชิงฟังก์ชัน (Functional Requirements)**

   ทำหน้าที่อธิบายว่าระบบควรทำอะไรได้บ้าง เช่น การตรวจสอบข้อมูลนำเข้า (input validation), การจัดเก็บข้อมูล (data storage) และส่วนต่อประสานกับผู้ใช้ (user interface)

2. **ข้อกำหนดที่ไม่ใช่เชิงฟังก์ชัน (Non-Functional Requirements)**

   ทำหน้าที่อธิบายว่าระบบควรทำงานได้ดีเพียงใด หรือคุณภาพของระบบควรเป็นอย่างไร เช่น ประสิทธิภาพ (performance), ความน่าเชื่อถือ (reliability), ความสามารถในการใช้งาน (usability), ความปลอดภัย (security) นอกจากนี้ยังรวมถึง ความสามารถในการบำรุงรักษา (maintainability), ความสามารถในการขยายระบบ (scalability), ความสามารถในการตรวจสอบ (testability), และความสามารถในการนำกลับมาใช้ใหม่ (reusability)

3. **ข้อจำกัด (Constraints)**

   ทำหน้าที่อธิบายถึงข้อจำกัดหรืออุปสรรคต่าง ๆ ที่ต้องพิจารณาถึงในการดำเนินการพัฒนาระบบ

4. **เกณฑ์การยอมรับ (Acceptance Criteria)**

   ทำหน้าที่ระบุเงื่อนไขที่ระบบต้องสามารถทำได้ถึงจะยอบรับได้ว่าระบบสมบูรณ์และพร้อมสำหรับการนำไปใช้งานจริง

การตรวจรับ (Validation)

เป็นขั้นตอนในการตรวจสอบว่าข้อกำหนดของระบบที่ได้จัดทำในขั้นการจัดทำข้อกำหนดมีความถูกต้องสมบูรณ์ ข้อกำหนดต่าง ๆ มีความสอดคล้องไม่ขัดแย้งกัน และต้องสามารถตอบสนองต่อความต้องการของผู้ใช้ได้จริง ๆ

สิ่งที่ต้องพิจารณาในการตรวจรับ

1. **ความเที่ยงตรง (Validity):** ความต้องการในข้อกำหนดจะต้องเป็นสิ่งที่ผู้ใช้ต้องการอย่างแท้จริง และต้องเป็นไปตามวัตถุประสงค์ของโครงการ
2. **ความสอดคล้อง (Consistency):** ความต้องการ และข้อกำหนดทั้งหมดต้องสอดคล้องกันหรือไม่มีความขัดแย้งกันเองในแต่ละข้อ
3. **ความครบถ้วนสมบูรณ์ (Completeness):** ข้อกำหนดจะต้องครอบคลุมความต้องการทุกด้านอย่างครบถ้วน
4. **ความเป็นไปได้ (Feasibility):** ข้อกำหนดจะต้องสามารถนำไปพัฒนาได้จริงในทางปฏิบัติ
5. **สามารถพิสูจน์ได้ (Verifiability):** ข้อกำหนดทั้งหมดจะต้องสามารถวัดผลได้

การจัดการความต้องการ (Requirement management)

เป็นขั้นตอนในการวางแผน จัดการ และควบคุมข้อกำหนด (Requirements) ของระบบหรือซอฟต์แวร์ ตลอดวงจรชีวิตของโครงการ โดยมีเป้าหมายเพื่อให้แน่ใจว่า สิ่งที่ผู้ใช้งานต้องการจะถูกแปลงเป็นระบบที่ตอบโจทย์จริง และสามารถปรับเปลี่ยนได้อย่างมีประสิทธิภาพหากมีความต้องการใหม่เกิดขึ้น

ประโยชน์ที่ได้รับ

1. ช่วยลดความเข้าใจผิดระหว่างผู้พัฒนากับลูกค้า
2. สามารถส่งมอบงานที่ตรงตามความต้องการ
3. ลดข้อผิดพลาดที่จะเกิดขึ้นและช่วยป้องกันการเปลี่ยนแปลงที่ไม่จำเป็น
4. ควบคุมคุณภาพของการทำงาน
5. สามารถส่งมอบงานได้ตรงเวลา

แบบจำลองการวิเคราะห์ (Analysis Model)

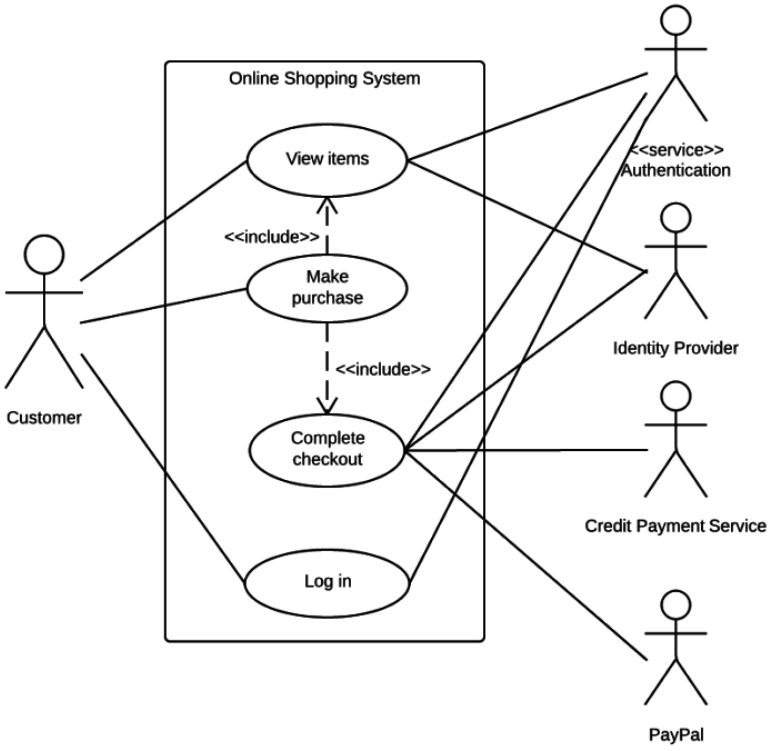
เป็นแบบจำลองที่เขียนขึ้นจากข้อกำหนดและความต้องการของระบบ เพื่อสะท้อนให้เห็นถึงหน้าที่การทำงานของระบบด้านต่าง ๆ ซึ่งแบบจำลองการวิเคราะห์สามารถแบ่งออกได้เป็น 4 ลักษณะ ได้แก่ ลักษณะเชิงฉากบรรยาย, ลักษณะเชิงคลาส, ลักษณะเชิงพฤติกรรม และลักษณะเชิงกระแส

ลักษณะเชิงฉากบรรยาย (Scenario-based element)

เป็นแบบจำลองที่ทำหน้าที่อธิบายให้นักวิศวกรซอฟต์แวร์ เห็นถึงลักษณะการใช้งานของผู้ใช้ที่จะมีปฏิสัมพันธ์กับระบบ โดยแบบจำลองนี้จะประกอบไปด้วยแผนภาพต่าง ๆ เช่น แผนภาพยูสเคส, แผนภาพกิจกรรม และแผนภาพสวิมเลน

แผนภาพยูสเคส (Use-Case Diagram)

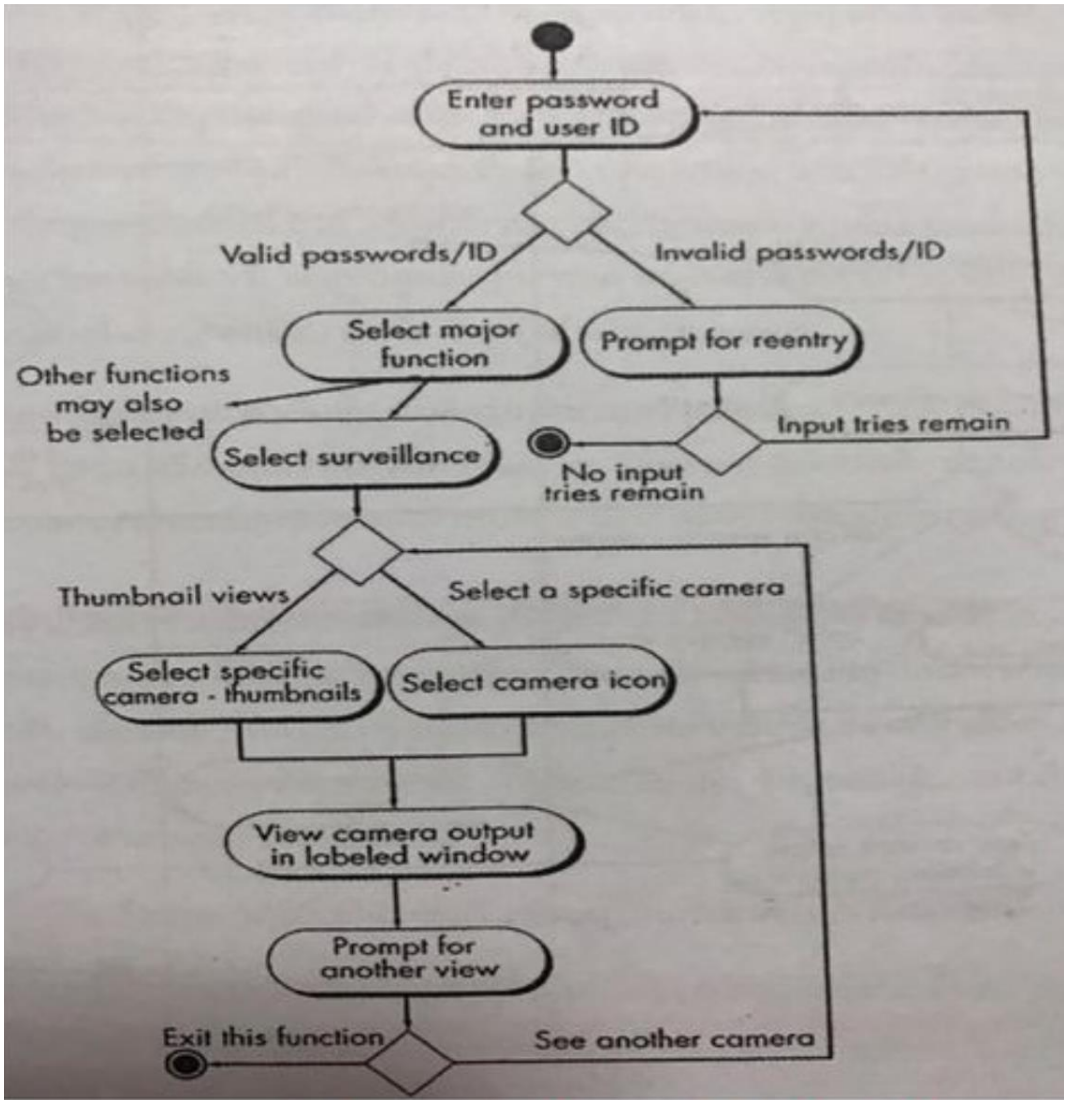
ใช้แสดงถึงปฏิสัมพันธ์ที่เกิดขึ้นระหว่างผู้ใช้กับตัวระบบ โดยแนวคิดของยูสเคสค่อนข้างจะเข้าใจง่าย โดยยูสเคสจะบรรยายถึงลักษระการใช้งาน เฉพาะด้วยภาษาที่ตรงไปตรงมาจากมุมมองของผู้กระทำ (Actor) ที่กำหนดไว้ โดยข้อมูลที่จะต้องใช้ในการสร้างยูสเคส ได้แก่ รายงานการพบปะสนทนากับลูกค้าและผู้ใช้งานอื่น ๆ ขอบเขตของปัญหาของระบบ รายการความต้องการเชิงหน้าที่และคำอธิบายสิ่งต่าง ๆ ที่จะถูกจัดการโดยระบบ



ภาพที่ 1 : ตัวอย่างแผนภาพยูสเคส

แผนภาพกิจกรรม (Activity Diagram)

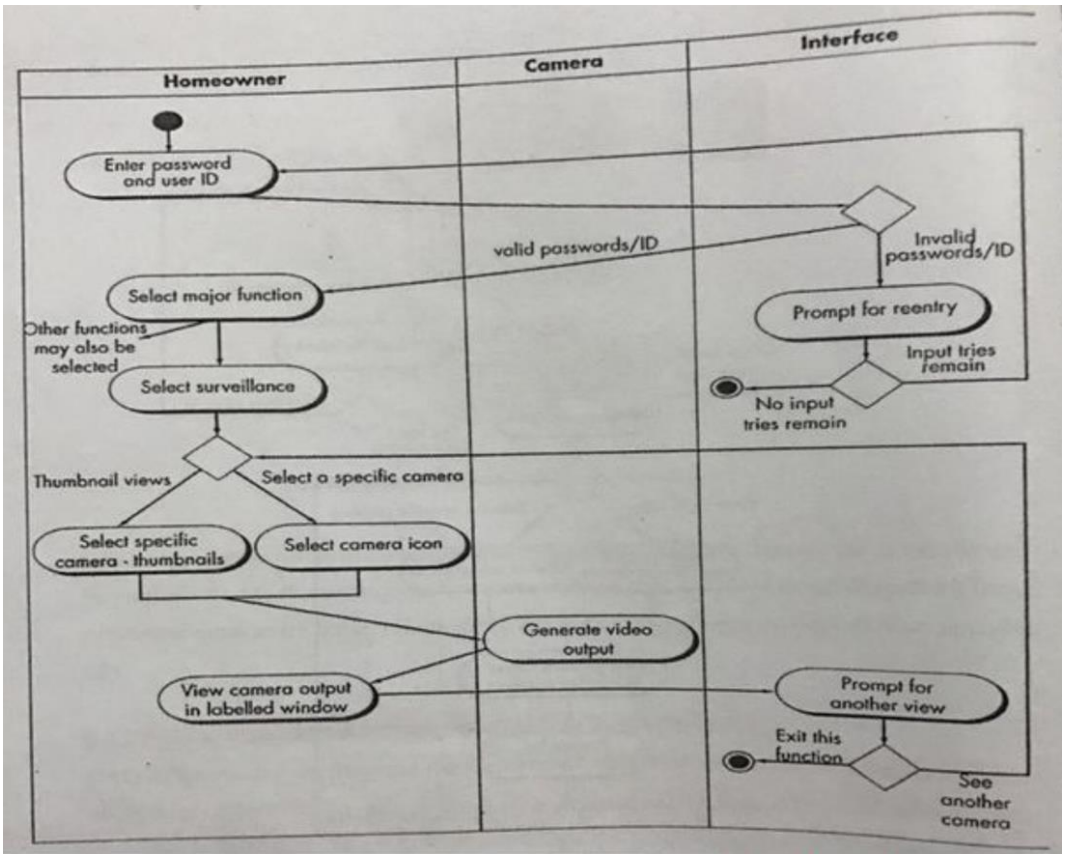
ทำหน้าที่ขยายยูสเคสให้ละเอียดขึ้น โดยจะแสดงรูปภาพของเส้นทางการกระทำที่เกิดขึ้นในฉากบรรยายหนึ่ง ๆ แผนภาพกิจกรรมจะมีลักษณะใกล้เคียงกับผังงาน (Flowchart)



ภาพที่ 2: ตัวอย่างแผนภาพกิจกรรม

แผนภาพสวิมเลน (Swimlane Diagrams)

เป็นแผนภาพกิจกรรมที่ใช้แสดงว่า Actor ใดทำหน้าที่รับผิดชอบต่อกิจกรรมใด โดยแบ่งแผนภาพออกเป็น "เลน" ตามแนวตั้ง เหมือนเลนในสระว่ายน้ำ เพื่อแยกความรับผิดชอบของแต่ละฝ่ายอย่างชัดเจนโดยส่วนบนของแผนภาพจะมีการระบุชื่อของ Actor ไว้



ภาพที่ 3: ตัวอย่างแผนภาพสวิมเลน

ลักษณะเชิงคลาส (Class-based element)

1. การระบุคลาสวิเคราะห์ (Identifying Analysis Classes)

วัตถุหรือคลาสอาจจะยากกว่าที่จะมองเห็นได้ เริ่มต้นด้วยการระบุคลาสโดยการสำรวจคำอธิบายปัญหาเราอาจจะหาคำนามหรือกลุ่มคำนาม และบันทึกไว้ในตาราง ให้ ระมัดระวังและบันทึกคำเหมือนกัน (Synonyms) ไว้ด้วย ถ้าหากว่าคลาสนี้จำเป็นสำหรับการใช้ในการสร้างระบบ คลาสนี้จะเป็นส่วนหนึ่งของคำตอบที่เราต้องการ อย่างไรก็ตาม ถ้าหากคลาสนี้จำเป็นเพียงเพื่ออธิบายคำตอบ มันก็จะเป็นส่วนหนึ่งของปัญหา

คลาสวิเคราะห์อาจจะแสดงตัวตนอยู่ในรูปแบบต่อไปนี้

- เอนทิตี้ภายนอก (External Entity) ได้แก่ ตัวระบบ ตัวเครื่องมือ ตัวบุคคล ผู้ผลิต หรือผู้บริโภค ข้อมูลที่ออกมาจากระบบ
- สิ่งของ (Thing) เช่น ตัวรายงาน ตัวแสดงผล ตัว จดหมาย ตัวสัญญาณซึ่งเป็นส่วนหนึ่งของโดเมนข้อมูลสำหรับปัญหานี้
- บทบาท (Roles) ได้แก่ ผู้จัดการ นักวิศวกร พนักงานขาย ซึ่งแสดงโดยบุคคลที่มีปฏิกิริยาโต้ตอบกับระบบ
- หน่วยองค์กร (Organization Unit) ได้แก่ แผนก กลุ่ม หรือทีมงานซึ่งเกี่ยวข้องกับตัวโปรแกรมประยุกต์
- สถานที่ (Places)

2. การระบุแอตทริบิวส์ (Specifying Attribute)

แอตทริบิวส์อธิบายคลาสที่ได้เลือกสรรแล้วว่าอยู่ในแบบจำลองการวิเคราะห์ แอตทริบิวส์คือคุณสมบัติของคลาส เป็นการให้รายละเอียดว่าคลาสนี้มีคุณสมบัติอะไรบ้าง การพัฒนาชุดของแอตทริบิวส์ที่มีความหมายสำหรับคลาสการวิเคราะห์ นักวิศวกรซอฟต์แวร์สามารถศึกษายูสเคสและเลือกสิ่งของที่มีเหตุที่ควรจะเป็นสมบัติของคลาสนั้น

3. การระบุโอเปอร์เรชั่น (Specifying Operation)

ตัวดำเนินการหรือโอเปอร์เรชั่น (Operation) เป็นตัวกำหนดพฤติกรรมของวัตถุ โอเปอร์เรชั่นอาจจะมีหลายชนิด แต่เราอาจจะแบ่งโอเปอร์เรชั่นออกเป็นชนิดใหญ่ๆ ดังนี้

3.1. โอเปอร์เรชั่นที่ท่าการจัดการข้อมูล

3.2. โอเปอร์เรชั่นที่ทำการคำนวณ

3.3. โอเปอร์เรชั่นที่ทำการสอบถามเกี่ยวกับสถานะของวัตถุ

3.4. โอเปอร์เรชั่นที่ทำหน้าที่ตรวจสอบวัตถุว่ามีเหตุการณ์บางอย่างเกิดขึ้นหรือไม่

4. การสร้างแบบจำลองซีอาร์ซี (Class Responsibility Collaborator)

- เป็นวิธีการง่าย ๆ สำหรับการระบุและจัดระเบียบคลาสที่เกี่ยวข้องกับระบบ
- เป็นการรวบรวมกระดาษแข็งจำนวนหนึ่งซึ่งแต่ละใบแสดงถึงคลาส ๆ หนึ่ง
- มีลักษณะเดียวกับบัตรดัชนี (Index Card) วัตถุประสงค์คือการจัดระเบียบคลาสให้ดี

เราจะแบ่งคลาสทั้งหลายออกเป็น 3 ประเภท ดังนี้

1) คลาสเอนทิตี้ (Entity Classes)

บางครั้งเรียกว่า คลาสแบบจำลอง (Model Classes) หรือ คลาสธุรกิจ (Business Classes) ได้มาจากประโยคข้อความของ ปัญหาโดยตรง ตัวอย่างเช่น FloorPlan หรือ Sensor คลาสเหล่านี้มักจะแสดงสิ่งของที่อาจจะถูกจัดเก็บอยู่ในฐานข้อมูลอย่างถาวรตลอดช่วงระยะเวลาการทำงานของแอพพลิเคชั่น

2) คลาสขอบเขต (Boundary Classes)

ใช้ในการสร้างอินเตอร์เฟสหรือหน้าต่างแสดงการติดต่อกับผู้ใช้งาน หรือทำการพิมพ์รายงานที่ผู้ใช้งานอาจจะเห็นหรือมี ปฏิสัมพันธ์ด้วยขณะที่กำลังใช้ซอฟต์แวร์ คลาสเอนทิตี้บรรจุข้อมูลที่จำเป็นสำหรับผู้ใช้งานแต่ไม่สามารถแสดง ตัวตนออกมาได้ คลาสขอบเขตถูกออกแบบมาเพื่อให้ รับผิดชอบจัดการกับวัตถุเอนทิตี้ โดยนำมาแสดงให้กับ ผู้ใช้งาน ตัวอย่างเช่น คลาสขอบเขตที่ชื่อว่า CameraWindow มีหน้าที่แสดงเอาท์พุตจากกล้อง

3) คลาสตัวควบคุม (Controller Classes)

ใช้จัดการ หน่วยงานตั้งแต่เริ่มต้นจนจบ โดยออกแบบมาให้จัดการกับ

- การสร้างหรือการปรับวัตถุข้อมูลให้เป็นปัจจุบัน
- การสร้างวัตถุขอบเขต เพื่อให้ดึงข้อมูลออกจากวัตถุเอนทิตี้
- การสื่อสารระหว่างชุดของวัตถุ

- การตรวจสอบความถูกต้องของข้อมูลที่สื่อสารระหว่าง ออบเจ็กต์หรือระหว่างผู้ใช้งานกับแอพพลิเคชั่น โดยทั่วไปแล้ว คลาสตัวควบคุมไม่ได้จัดอยู่ในขั้นการวิเคราะห์

5. ความสัมพันธ์และการขึ้นแก่กัน (Association and Dependencies)

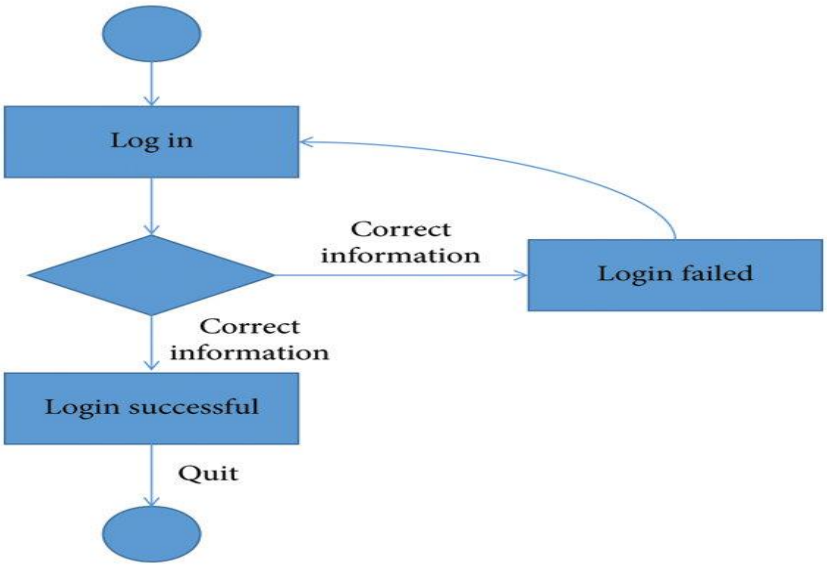
คลาสวิเคราะห์หลายๆ คลาสอาจจะมีความสัมพันธ์กันได้ ในบางลักษณะ เหมือนวัตถุข้อมูลที่อาจจะสัมพันธ์กันได้นั่นเอง ในสัญลักษณ์ UML เราจะเรียกว่า ความสัมพันธ์ ระหว่างคลาส (Association) ในหลาย ๆ เหตุการณ์ความสัมพันธ์แบบการให้และการรับบริการเกิดขึ้นระหว่างคลาสวิเคราะห์สอง ในกรณีนี้คลาสที่รับบริการจะขึ้นอยู่กับคลาสที่ให้บริการ และเกิดความสัมพันธ์แบบขึ้นแก่กัน (Dependency) ขึ้น ซึ่งอาจถูกกำกับด้วยสัญลักษณ์ตัวพิมพ์ Stereotype หมายถึง กลไกที่ขยายเพิ่มเติมขึ้นภายใน UML ซึ่งอนุญาตให้นักวิศวกรซอฟต์แวร์นิยามแบบจำลองพิเศษ

6. แพ็คเกจการวิเคราะห์ (Analysis Packages)

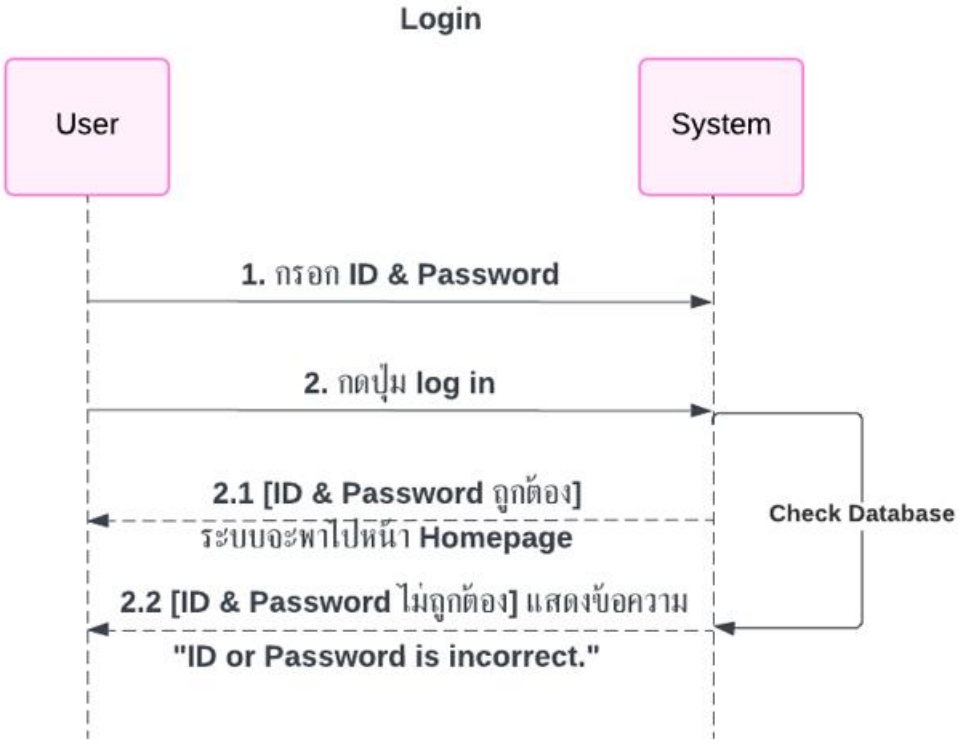
ส่วนสำคัญส่วนหนึ่งของแบบจำลองวิเคราะห์ คือ การจัดหมวดหมู่ นั่นคือแต่ละส่วนประกอบของแบบจำลอง การวิเคราะห์ ได้แก่ ยูสเคสหรือคลาสการวิเคราะห์ จะถูกจัดหมวดหมู่ในลักษณะที่เป็นการรวมกลุ่มหรือแพ็คเกจ โดยเรียกว่าแพ็คเกจการวิเคราะห์ (Analysis Package)

ลักษณะเชิงพฤติกรรม (Behavioral elements)

เป็นลักษณะของแบบจำลองที่แสดงให้เห็นถึงกิจกรรมและพฤติกรรมของระบบที่มีการเปลี่ยนแปลง เมื่อมีเหตุการณ์เกิดขึ้น รวมทั้งแสดงให้เห็นความสามารถของระบบที่ดำเนินการในบางหน้าที่ได้ โดยแบบจำลองนี้จะประกอบไปด้วยแผนภาพต่าง ๆ เช่น แผนภาพสถานะ (State Diagram) แผนภาพลำดับ (Sequence Diagram)



ภาพที่ 4: ตัวอย่างแผนภาพสถานะ



ภาพที่ 5: ตัวอย่างแผนภาพลำดับ

ลักษณะเชิงกระแส (Flow-oriented elements)

เป็นการนำแบบจำลอง (Models) มาใช้เพื่ออธิบายการไหลของข้อมูลภายในระบบซอฟต์แวร์ รวมถึงการประมวลผลข้อมูล และที่เก็บข้อมูลต่างๆ หัวใจหลักของ Flow-oriented elements คือการตอบคำถามว่า **"ข้อมูลเดินทางอย่างไรในระบบนี้"**

เครื่องมือหลัก

เครื่องมือหลักที่เราใช้ในการแสดง Flow-oriented elements ก็คือ Data Flow Diagram (DFD) โดยDFD เป็นแผนภาพที่แสดงให้เห็นว่าข้อมูลเข้าสู่ระบบอย่างไร ถูกประมวลผลที่ไหน เก็บไว้ที่ใด และข้อมูลเหล่านั้นส่งออกไปที่ไหนต่อบ้าง โดยไม่สนว่าเทคโนโลยีที่ใช้คืออะไร หรือการทำงานภายในเป็นโค้ดอย่างไร แต่เน้นที่การไหลของข้อมูลเชิงตรรกะ (Logical Flow)

ส่วนประกอบสำคัญของ DFD

DFD ประกอบด้วยสัญลักษณ์หลัก 4 อย่างที่ใช้ในการสร้างแผนภาพ ได้แก่

1. **External Entity (หน่วยงานภายนอก)**
- คือ: บุคคล, องค์กร, หรือระบบอื่น ๆ ที่ส่งหรือรับข้อมูลจากระบบแต่ไม่ได้เป็นส่วนหนึ่งของระบบที่กำลังวิเคราะห์
- ตัวอย่าง: ลูกค้า (ป้อนข้อมูลคำสั่งซื้อ), ธนาคาร (รับข้อมูลการชำระเงิน)
- สัญลักษณ์: มักเป็นรูปสี่เหลี่ยมผืนผ้า

2. **Process (กระบวนการ)**
- คือ: การกระทำหรือฟังก์ชันที่แปลงข้อมูลขาเข้าให้เป็นข้อมูลขาออก
- ตัวอย่าง: คำนวณราคา, ตรวจสอบข้อมูลลูกค้า, บันทึกข้อมูล
- สัญลักษณ์: มักเป็นรูปวงกลม หรือสี่เหลี่ยมมุมโค้งมน

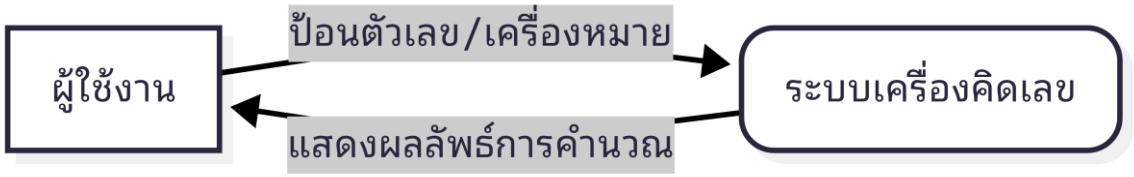
3. **Data Store (ที่เก็บข้อมูล)**
- คือ: การกระทำหรือฟังก์ชันที่แปลงข้อมูลขาเข้าให้เป็นข้อมูลขาออก
- ตัวอย่าง: คำนวณราคา, ตรวจสอบข้อมูลลูกค้า, บันทึกข้อมูล
- สัญลักษณ์: มักเป็นรูปวงกลม หรือสี่เหลี่ยมมุมโค้งมน

4. **Data Flow (กระแสข้อมูล)**
   - คือ: การเคลื่อนที่ของข้อมูลจากส่วนหนึ่งไปยังอีกส่วนหนึ่ง
   - ตัวอย่าง: ลูกค้า (ส่ง) ข้อมูลคำสั่งซื้อ (ไปที่) ระบบ, ระบบ (ส่ง) ผลลัพธ์การคำนวณ (ไปที่) หน้าจอ
   - สัญลักษณ์: ลูกศรที่มีชื่อกำกับทิศทางการไหลของข้อมูล

ระดับของ DFD

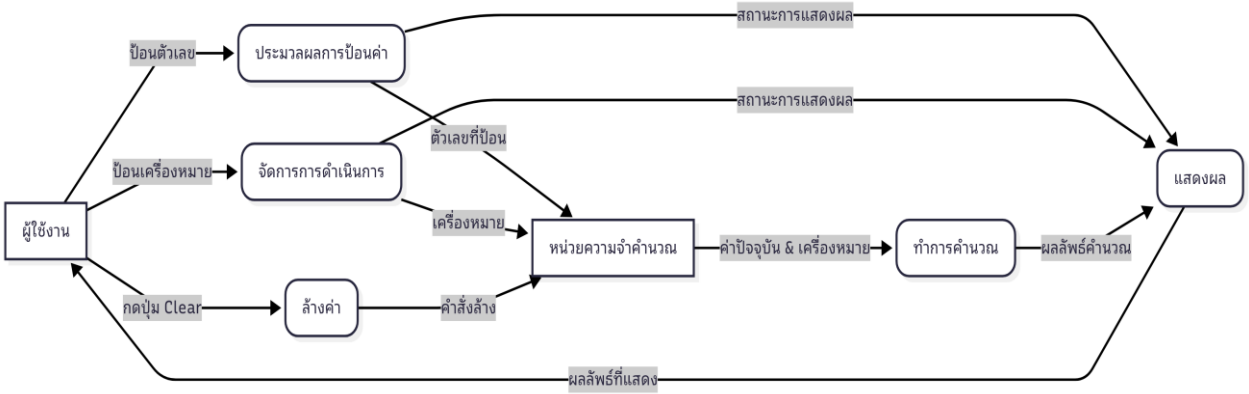
1. **Context Diagram (Level 0)**

- ภาพรวม: เป็น DFD ระดับสูงสุด ที่แสดงภาพรวมของ ทั้งระบบ เป็นเพียงกระบวนการเดียว (Single Process)
- เน้น: การปฏิสัมพันธ์กับ External Entities (ใครส่งอะไรให้ระบบ และใครได้รับอะไรจากระบบบ้าง)
- ประโยชน์: ช่วยให้เห็นขอบเขตและบริบทของระบบโดยรวม



2. **DFD (Level 1)**

- ภาพรวม: เป็น DFD ระดับสูงสุด ที่แสดงภาพรวมของ ทั้งระบบ เป็นเพียงกระบวนการเดียว (Single Process)
- เน้น: การปฏิสัมพันธ์กับ External Entities (ใครส่งอะไรให้ระบบ และใครได้รับอะไรจากระบบบ้าง)
- ประโยชน์: ช่วยให้เห็นขอบเขตและบริบทของระบบโดยรวม



3. **Lower-level DFDs (Level 2, 3, ...)**

- ภาพรวม: เป็น DFD ระดับสูงสุด ที่แสดงภาพรวมของ ทั้งระบบ เป็นเพียงกระบวนการเดียว (Single Process)
- เน้น: การปฏิสัมพันธ์กับ External Entities (ใครส่งอะไรให้ระบบ และใครได้รับอะไรจากระบบบ้าง)
- ประโยชน์: ช่วยให้เห็นขอบเขตและบริบทของระบบโดยรวม

กฎพื้นฐานของ DFD

ในการสร้าง DFD เรามีกฎสำคัญที่เรียกว่า Balancing Principle (หลักความสมดุล) ซึ่งระบุว่า ***"ข้อมูลเข้าและออกของกระบวนการใน DFD ระดับล่าง จะต้องตรงกับข้อมูลเข้าและออกของกระบวนการเดียวกันใน DFD ระดับบนเสมอ"*** เพื่อให้แผนภาพทุกระดับมีความสอดคล้องกัน

ประโยชน์หลักของการใช้ DFD

1. **เข้าใจการไหลของข้อมูล:** ช่วยให้เห็นภาพรวมและรายละเอียดของข้อมูลที่เคลื่อนที่ในระบบ
2. **สื่อสารได้ดีขึ้น:** เป็นภาษาภาพที่ช่วยให้ผู้มีส่วนได้ส่วนเสีย (ผู้ใช้งาน, นักพัฒนา) เข้าใจระบบร่วมกัน
3. **ระบุปัญหา:** ช่วยหาจุดที่ข้อมูลหายไป หรือการทำงานที่ไม่สมเหตุสมผล
4. **พื้นฐานการออกแบบ:** เป็นข้อมูลสำคัญสำหรับการออกแบบฐานข้อมูลและโครงสร้างโปรแกรมในขั้นตอนต่อไป

ข้อจำกัดของ DFD

แม้ DFD จะมีประโยชน์มาก แต่ก็มีข้อจำกัดที่ควรทราบ ได้แก่

1. **ไม่แสดงลำดับเวลา:** DFD ไม่ได้บอกว่าอะไรเกิดขึ้นก่อน-หลัง หรือมีการตัดสินใจอย่างไร
2. **ไม่แสดง Flow ควบคุม:** DFA ไม่ได้แสดงตรรกะการตัดสินใจ (IF-ELSE) หรือเงื่อนไขการทำงาน

3. **ต้องใช้ร่วมกับ Diagram อื่น:** เพื่อให้เข้าใจระบบได้สมบูรณ์ มักต้องใช้ร่วมกับ Use Case Diagram (หน้าที่ของระบบ), Class Diagram (โครงสร้างข้อมูล), หรือ Activity Diagram (ลำดับการทำงาน)

สรุป

Flow-oriented elements โดยเฉพาะ Data Flow Diagram (DFD) เป็นเครื่องมือที่ทรงพลังในการวิเคราะห์ระบบซอฟต์แวร์ ช่วยให้เราทำความเข้าใจการไหลของข้อมูล ได้อย่างชัดเจน ตั้งแต่ภาพรวมไปจนถึงรายละเอียด การเข้าใจ DFD จะเป็นพื้นฐานสำคัญในการพัฒนาระบบที่มีคุณภาพและตรงตามความต้องการ

อ้างอิง

- ดร. เกตุกาญจน์ โพธิจิตติกานต์. (2555). **ความต้องการด้านซอฟต์แวร์.** สืบค้นเมื่อ 22 กรกฎาคม 2568, จาก http://www.rmuti.ac.th/user/kedkarn/2012/software_en/requirement_eng.ppt
- นิธิพร รอดรัตษะ. (2556). **วิศวกรรมความต้องการ.** สืบค้นเมื่อ 22 กรกฎาคม 2568, จาก https://ajnitiporn.wordpress.com/wp-content/uploads/2012/08/rm-ch7.pdf
- มหาวิทยาลัยราชภัฏนครปฐม. (2568). **การวิเคราะห์ซอฟต์แวร์.** สืบค้นเมื่อ 22 กรกฎาคม 2568, จาก https://view.officeapps.live.com/op/view.aspx?src=http%3A%2F%2Fcourseware.npru.ac.th%2Fadmin%2Ffiles%2F20180108114230_4385316f8e51ea74f44a1b77b705ba55.pptx&wdOrigin=BROWSELINK
- Alexandra Jonker, Alice Gomstyn. (2568). **What is requirements management?.** สืบค้นเมื่อ 22 กรกฎาคม 2568, จาก https://www.ibm.com/think/topics/what-is-requirements-management
- Geeks for Geeks. (2567). **Software Requirement Tasks - Software Engineering.** สืบค้นเมื่อ 22 กรกฎาคม 2568, จาก https://www.geeksforgeeks.org/software-engineering/software-engineering-software-requirement-tasks/
- Geeks for Geeks. (2568). **Requirements Engineering Process in Software Engineering.** สืบค้นเมื่อ 22 กรกฎาคม 2568, จาก https://www.geeksforgeeks.org/software-engineering/software-engineering-requirements-engineering-process/
- GitHub . (2568). **GitHub Copilot Chat (Jul 2025 Version) [Large language model].** สืบค้นเมื่อ 23 กรกฎาคม 2568, จาก https://docs.github.com/en/copilot/
- Google. (2568). **Gemini (Version 2.5) [Large language model].** สืบค้นเมื่อ 23 กรกฎาคม 2568, จาก https://gemini.google.com/

- Sucianty Gunawan. (2562). **การสร้างแบบจำลองการวิเคราะห์ (Building the Analysis Model).** สืบค้นเมื่อ 22 กรกฎาคม 2568, จาก https://slideplayer.in.th/slide/15148494/
- Visure Solutions. (2568). **What is Requirements Engineering: Process for Software and Systems.** สืบค้น เมื่อ 22 กรกฎาคม 2568, จาก https://visuresolutions.com/alm-guide/requirements-engineering/